Olympus America

iOS/Mobile Development
Process and Guidelines

02/19/2013

Purpose:

The purpose of this document is to provide a guideline for the development of mobile applications developed by Olympus America, Inc. or third party vendors they contract/employ.

Audience:

Anyone involved in the development of mobile applications for Olympus America, business unit personnel, designers, developers and support technology staff.

Mobile Development Process

The mobile development process or the software development life cycle (SDLC) consist of several phases. These phases can occur sequentially, iteratively or in parallel depending on the development method selected for each project, such as Ad Hoc, Waterfall, Agile, etc.

In general, the phases are:

- Business Requirements Gathering/Analysis
- Application Functional Workflow
- Design Wireframe
- Prototyping
- Development
- Testing/Quality Assurance/Control
- Deployment
- Maintenance/Support

Process Responsibilities

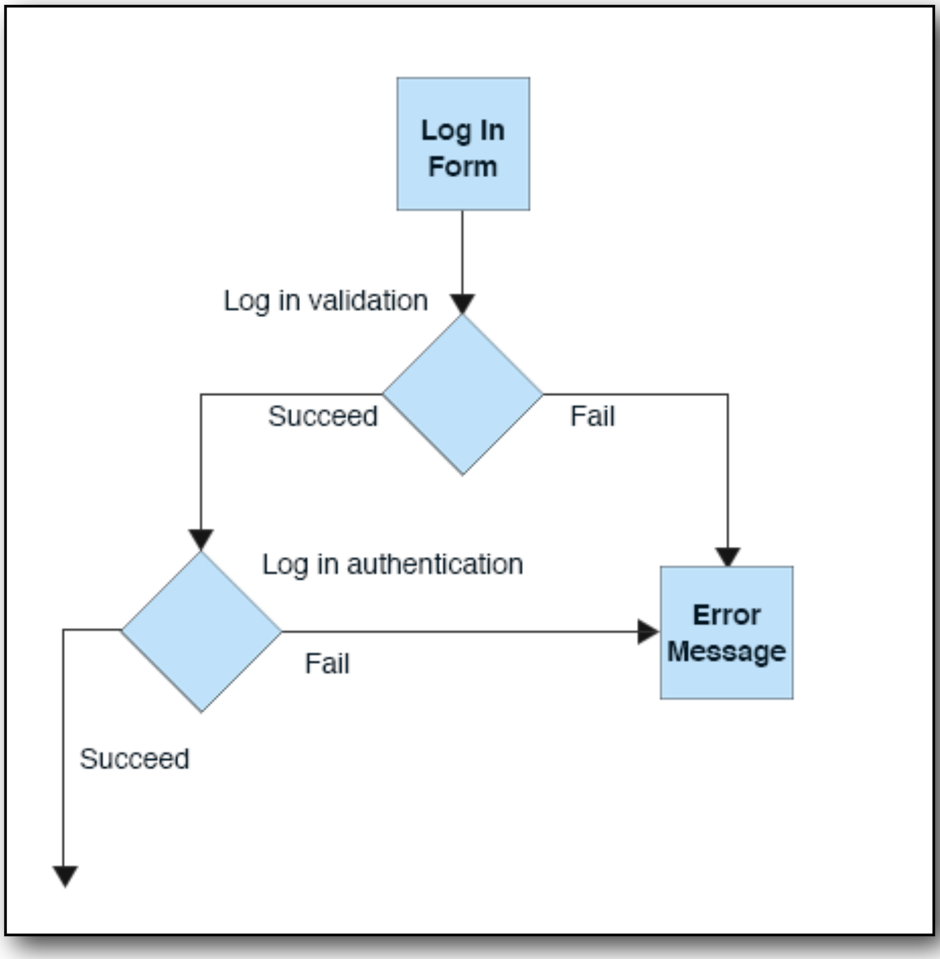
Process	Primary Responsibility	Secondary Responsibility
Business Requirements Gathering/Analysis	Business Analyst/Project Stakeholder	All
Application Functional Workflow	Architect/Lead Developers	Business Analyst, User Interface Specialist
Design Wireframe	User Interface Specialist	Architect/Lead Developer
Prototyping (Hi and Low Fidelity)	User Interface Specialist/Lead Developer	Architect
Development	Lead Developer	Development Team
Testing/Quality Assurance/Control	QA Lead/Regulatory Affair Lead	Lead Developer/Project Stakeholder
Deployment	ITS	Lead Developer/Project Stakeholder
Maintenance/Support	ITS	Lead Developer/Project Stakeholder

Business Requirements Gathering/Analysis

- Determine Project Goals
- Determine User Needs
- Set Up Project Requirements
- Perform Competitive Analysis (Not the term I want to use - check to see if an off the shelf app already exists or if we have already created something similar.

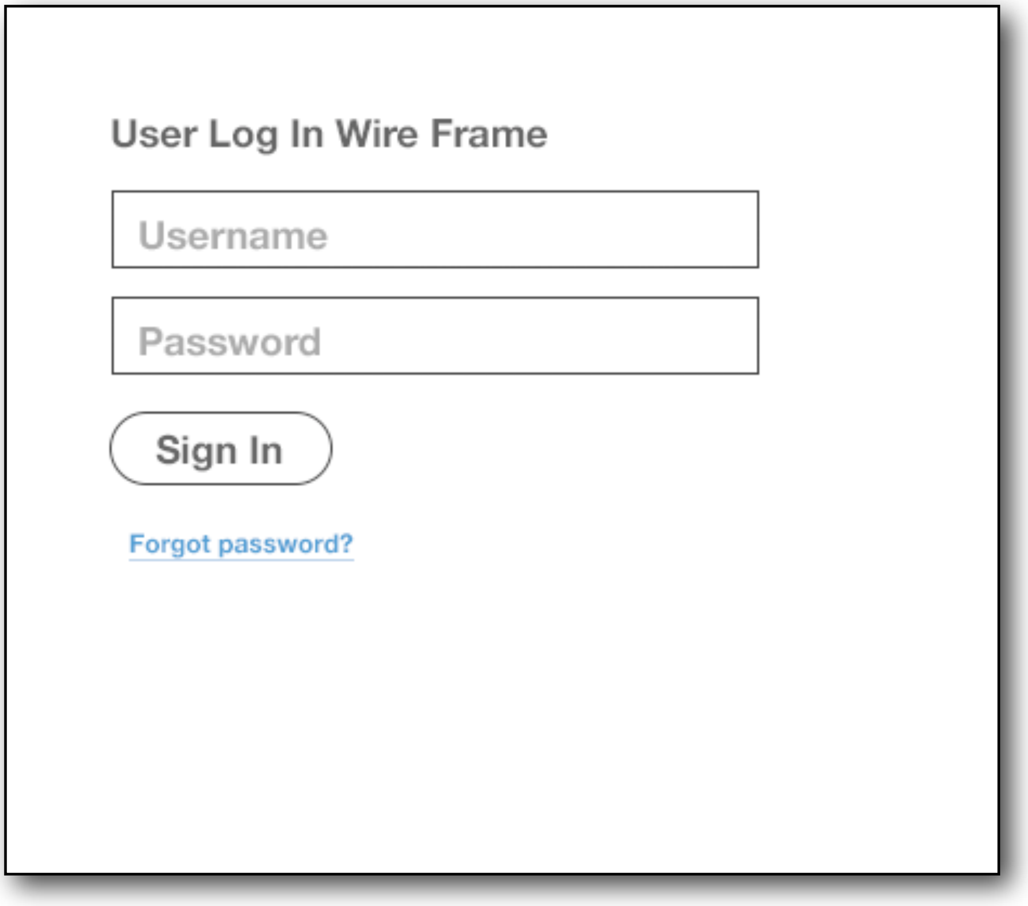
Application Functional Workflow

The application functional workflow determine when and where events happen in regards to code logic - see the example below for a user login:



Design Wireframing

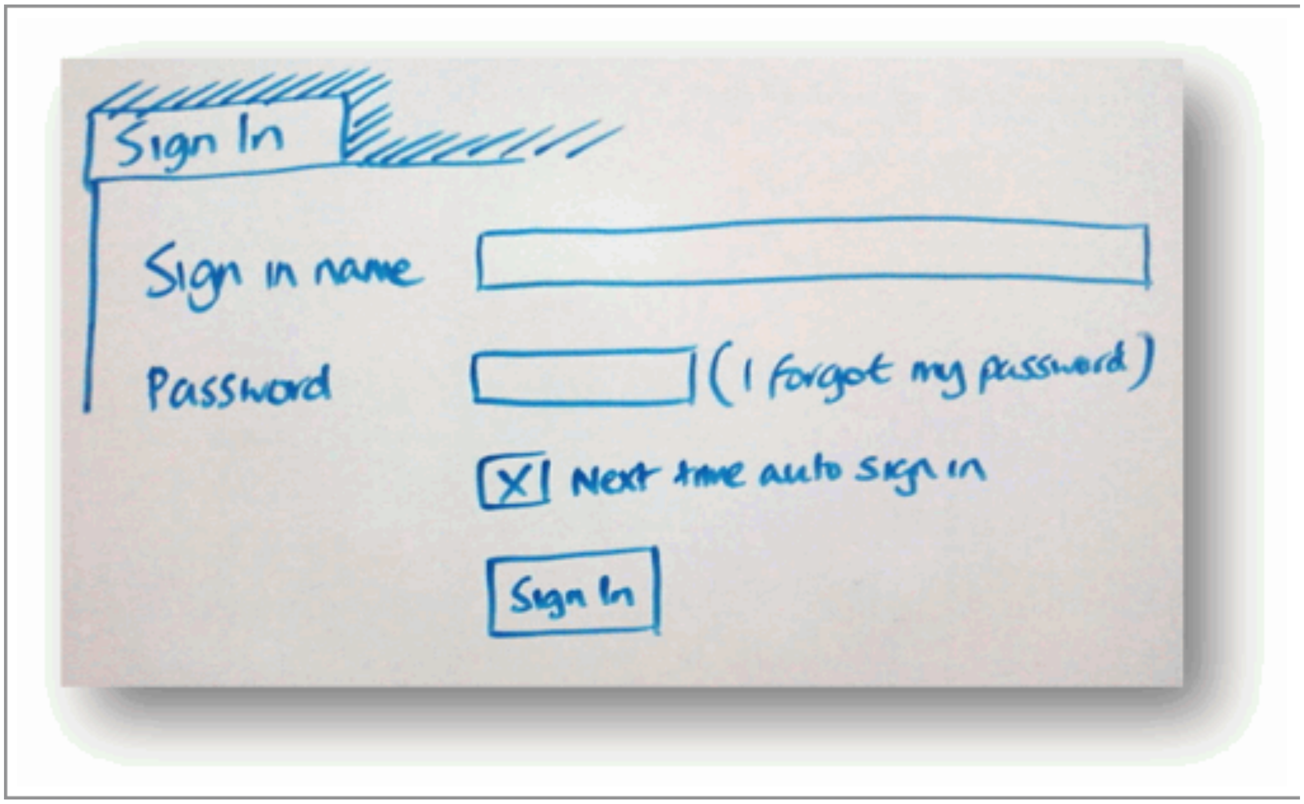
Wireframing is the design equivalent of the functional workflow. It allows the designers to provide a pseudo layout of the major elements of the application. These are created from a high level view so that the design team can then work in detail.



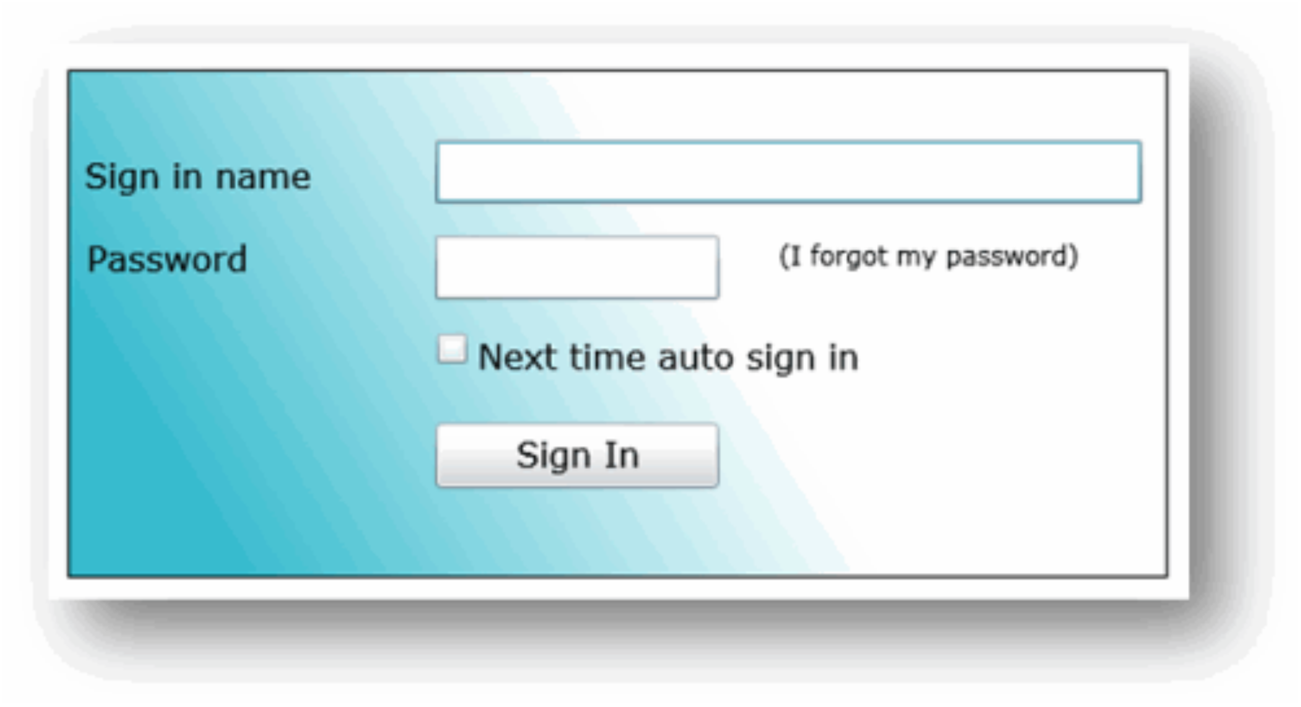
Prototyping

Prototyping is the process of taking the functional workflow and the design wireframes and use them to design the user interface for the application. They allow for rapid, iterative design changes and the ability to conduct meaningful user test for evaluation and determining application specifics. Prototypes come in three levels, low, medium and high.

Low fidelity Prototypes: These are the back of the envelope sketches that get created during brain storming sessions. Depending on the complexity of the project the design wireframe and the low fidelity prototype can be interchangeable.



Medium Fidelity Prototypes: A progression past the sketch level, where the project is defined throughly enough where detailed visuals are needed. Medium fidelity prototypes are generally created in imaging software such as Photoshop or a layout program such as Adobe Indesign or Apple's Pages.



High Fidelity Prototyping: These are pseudo functional emulations of the final developed application designed either with HTML and displayed through a web browser or with a prototyping tool such as iRise or FluidUI.

Design

Design is where the functional workflow and wire frames are merged and the application begins to come to life. Design can be implemented in all three facets of prototyping and obviously is one half of the success of the live app.

Reference Appendix A. for the Olympus Mobile Design Style Guide.

Development

Since Olympus has made a commitment to iOS development will be in XCode, Apple's Integrated Development Environment (IDE). The purpose of this guide will not to rigidly set up development standardization so as to limit a developer's creativity and programming skills but rather to ensure that throughout Olympus a development style is followed so that any iOS developer can open up a project started by someone else and have little learning curve to get up to speed.

Reference Appendix B. for the Olympus Mobile Development Style and Standardization Guide.

Quality Assurance/Testing

There are four types of QA/Testing for app development.

1. Do the app methods work the way they are suppose to. ie - if it is suppose to add 1+1 do we get 2 or 3?

2. Does the app deliver what the client asked for and does it meet their UI/UX requirements.

3. Does the app meet our regulatory guidelines (if necessary).

4. Does the app work on the various devices and OS's we currently support.

Reference Appendix C for the Olympus Mobile Development Testing Guidelines

Deployment

Olympus uses Afaria from Sybase/SAP as it's MDM/MAM tool to deploy in-house and vendor apps.

Reference Appendix D for the Olympus Mobile Development Deployment Guidelines

Support And Maintenance

Reference Appendix E for the Olympus Mobile Development Support And Maintenance Guidelines

Appendix A

Olympus Mobile Design Style Guide

This appendix is to be used as a mobile design style guide for development of Olympus iOS/ mobile apps. The mobile design style guide is broken into two section. The first section discusses standards in presenting the "story" of each app. The second provides guidance on color use, object designs and placement.

Presenting The App "Story"

Every Olympus app will be unique but also will contain some standardized items which will help to ensure branding and recognition across the enterprise and customers (when we get around to building apps for the consumers).

Application Icon

The application icon must follow Apple's guidelines for application icon design (see Table 1). Additionally they must adhere to the following Olympus guidelines:

1. Olympus logo must appear on top of icon.
2. Icon must follow Olympus color coding specs for type of app (Calculator, Marketing App, etc.)*
3. Icon must follow Olympus icon image specs for type of app. *

* To be established.

Application Display Naming Convention

There is little guidance from Apple in this regards. There are no limit on the number of characters as the device will just abbreviate the name of the app. For example RealLongApplicationName would appear as RealLo...onName or something similar. What is displayed is not regulated by the number of characters but rather by the space allocated to the name. Attempting to pixel count here will lead you down the path to madness as each iOS device can potentially have different space available to it (such as iPad2, iPad3, iPhone5, etc.) and what is displayed is reliant on the characters in the name (W takes much more space than I, numbers more than letters). You will have to go by a rule of thumb in this case and twelve seems to be a safe number of characters to use.

Within that limitation the app name should, in some way, be reflective of the app's functionality. For example, if the app is a calculator for determining break even points on EBUS purchases it could be titled EBUSBreakEven, which is 13 letters with no spacing.

If you must eliminate spacing from an application name use camel case format - myAppName.

Splash Screen and Launch Image:

Each app will open with the same Olympus launch image and splash screen, which will then fade into the title screen of the app.

(The Launch Image is a screen capture of the app (in our case it will just be a capture of the splash screen) which is presented to the user while the app loads. It is not as important for small apps with quick load times as it is for load intensive ones (think of WIRED magazines app which downloads almost 300MB of data per issue).

Title Screen

The title screen let's the user know, in clear terms, what app they are in. Depending on the app it can fade out to an app start screen or can be the start screen itself.

Start Screen

Where the user will begin interacting with the app. It will contain initial navigation and/or instructions if necessary. This could potentially be the title screen.

Table 1 - Apple Recommended Icon Sizes

Description	Hi Res iPad	iPad	iPhone 5 (and hi res phones)	iPhone/iPod Touch
App Icon	144x144	72x72	114x114	57x57
Launch Image	1536 x 2008 (portrait) 2048 x 1496 (landscape)	768 x 1004 (portrait) 1024 x 748 (landscape)	640 x 1136 (iPhone 5) 640 x 960 (hi res phone)	320 x 480
Small icon for Spotlight search results and Settings (recommended)	100 x 100 (Spotlight search results) 58 x 58 (Settings)	50 x 50 (Spotlight search results) 29 x 29 (Settings)	58 x 58	29 x 29

http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/mobilehig/IconsImages/IconsImages.html

Design Objects

There are a wide variety of design objects in the development of any app. I use the term design objects to refer to both native objects (such as iOS buttons) and objects created by the designer, such as a top bar to hold the Olympus logo and app title.

Appendix B

Olympus Mobile Design Style and Standardization Guide

Development

Since Olympus has made a commitment to iOS development will be in XCode, Apple's Integrated Development Environment (IDE). The purpose of this guide will not to rigidly set up development standardization so as to limit a developer's creativity and programming skills but rather to ensure that throughout Olympus a development style is followed so that any iOS developer can open up a project started by someone else and have little learning curve to get up to speed.

Standardizations - By following some basic coding standardizations we can ensure that all iOS developers are working in the same style. This style guide, though in and of itself not a panacea to big ball of mud development, will help to manage that from happening.

Organizing Code

1. **Comments**: Probably the one simple thing a programmer can do to help clarify his/her code is to place comments throughout explaining what they are doing and how.

    a. Comment above any methods to explain what the method will do, it's dependencies and if it returns data what the returned data is used for.

    b. Comment out blocks of common code at the beginning - as an example:

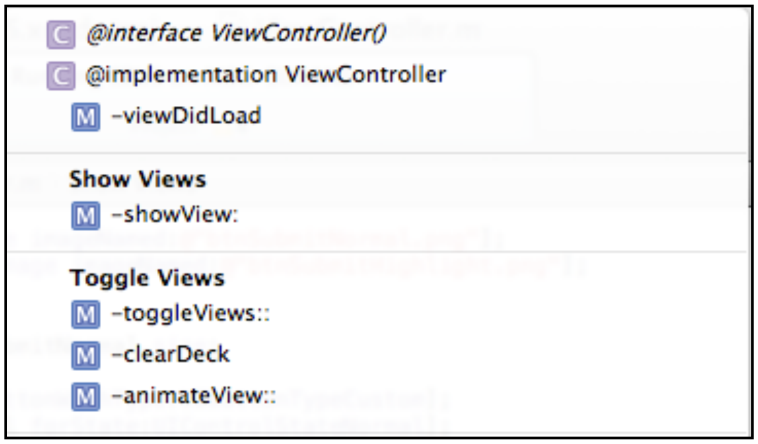
    ```
    //this is the username label
    ```

    c. Comment out each line of complex or unique code to help walk other programmers through your process.

2. **Pragma Marks**: Pragma marks are like comments for your IDE. Essentially they are an organizational tool that allows you to provide headings in the methods drop down list so you can group like methods. For example:

    ```
    #pragma mark - Show Views
    ```

    This would place the title Show Views in bold with an line beneath it (the hyphen is necessary for line) in the drop down list of methods in XCode.



```
C @interface ViewController()
C @implementation ViewController
    M -viewDidLoad
  Show Views
    M -showView:
  Toggle Views
    M -toggleViews::
    M -clearDeck
    M -animateView::
```

3. **Indentation**: Indenting code allows for easier reading and therefore easier debugging. As an example, the top snippet of code is much harder to read than the same code with indentation. :

```
//loop
for(UIView* thisSubView in mySubViews) {

//find the objectWrapper and set alpha to 0.0
 if (thisSubView.tag == 101) {

 [UIView animateWithDuration:0.5 delay:0.0 options:UIViewAnimationOptionCurveEaseIn

 animations:^{
thisSubView.alpha = 0.0;
 }

completion:^ (BOOL finished) {
}
];
}
}
```

```
//loop
for(UIView* thisSubView in mySubViews) {

        //find the objectWrapper and set alpha to 0.0
        if (thisSubView.tag == 101) {

                //animate view
                [UIView animateWithDuration:0.5 delay:0.0 
options:UIViewAnimationOptionCurveEaseIn

                        animations:^{
                                thisSubView.alpha = 0.0;
                        }

                        completion:^ (BOOL finished) {
                        }
                ];
        }
} //end loop
```

Class And Variable Naming

1. **Classes:** Class names should be written as OAI_MyClassName. OAI (Olympus America, Inc.) is used to ensure there are no namespacing collision (where you name a class the same as an existing class name in a framework you have included in your project) and the camel formatting is for ease of reading.

This does not entirely ensure that no namespacing collisions will occur but it does account for the majority of them.

2. **Variable Names:** I like to break variables into two types - data variables and object variables. In Objective-C data variables are known as primitives - int, float, double, char, short, long, etc. Object variables are references to Objective-C objects, such as UILabels, NSObject, etc.

   Naming data variables: Use the following format:

   ```
   float myFloatVariable;
   ```

   Naming object variables: I use a code for naming object variables so I can tell what type of object it is from the variable name. For example:

   ```
   UILabel* lblMyLabel
   ```

   I've used lbl as my prefix to distinguish this as a UILabel object. Here's the list of prefixes:

   UIView: vMyView
   UILabel: lblMyLabel
   UITextField/TextView: txtMyTextField
   UITableView: tblMyTableView
   UIScrollView: scrMyScrollView
   UISegementedControl: scMySegementedControl
   UIButton: btnMyButton
   UIImageView: ivMyImageView
   UIImage: imgMyImage

   NSString: strMyString
   NSArray/NSMutableArray: arrMyArray
   NSDictionary/NSMutableDictionary: dictMyDictionary
   NSNumberFormatter: nfMyNumberFormatter
   NSNumber: numMyNumber

Appendix C

Olympus Mobile Development Testing Guidelines

Quality Assurance, Quality Control, Testing and Regulatory Control

Olympus mobile development requires a high level of quality control on the app developed either in-house or through a vendor. The standards listed below for QA, QC and testing will be for either in-house standards, vendor standards or both and be designated as such. Regulatory Control will be entirely within Olympus.

Though Quality Assuance, Quality Control and Testing are all closely related in software development there are differences between the three and it is important to distinguish those differences. There are times when the lines between each will blur but we should look at each as an independent part of the over all process.

Quality Assurance:

QA is process oriented. It provides a set of activities that ensure the requirements of the project are being met within the guidelines established. In a nutshell, the right things are being done in the right way. QA is in place to help manage the development process and attempt to correct bugs and errors before the product is built.

There are two facets of Olympus QA. The first is ensuring that the app is being developed following the guidelines within this document. The second is ensuring that the developers are following the Development Style Guide (Appendix B).

Quality Control:

QC is product oriented. It ensures the results of what you've done are what you expected. Quality Control is an attempt to find bugs and errors after the product (or a portion) is built but before it is released. QC can also be seen as System Testing (Step 3 below).

Testing:

There are realistically hundreds of testing paradigms in software development. One of the main types of testing in software development is unit testing. Typically unit testing is an automated process to test a component (method or class) of the larger software product. The next level up of testing for software development is integration testing - testing the results of when two components need to work together.

We will not do either of these. Gasp! Here's the reasoning:

Most of the code written in a Cocoa or Cocoa Touch application is controller class code. These classes link things together, their whole purpose is coupling objects. Since controller classes are so common in iOS applications (View Controllers, UIScroll Controller, UITableView Controller, UIImageView Controllers, etc.) and their purpose is to join different objects together unit and integration testing suffers from serious problems in the application development.

1. Requires huge amounts of mocking code and other fakery (time consuming) to implement.
2. The resulting tests are far removed from the integrated reality, making false positives and false negatives highly likely and leaving large holes that are simply not tested.

The best approach to testing iOS applications is to do system testing. **Test the complete program in the exact manner (or as close as possible) in which you expect it to be used.** Because we are going to take a slightly unconventional approach to testing doesn't mean we will not institute a thorough testing paradigm. System testing can be composed of multitude levels of testing. We will test the following:

User Interface Testing:

User interface testing is typically done using a matrices. A spreadsheet is created with the columns indicating the different steps in operating the program, (pushing buttons, selecting menu items, etc.), the rows are the different devices to test on. Each cell should, after testing, contain a pass/fail result.

Example:

	Account Setting Button	Account Screen Close Button
	Clicking this button should reveal the account setting screen.	Clicking this button will hide the account screen.
iPad3 - iOS5	☺	☺
iPad3 - iOS6	☺	☺
iPad4 - iOS6	☺	☺

Regression Testing:

Regression testing is testing a software application that has been shown not to contain any bugs, when additional code is added to it.

Performance Testing:

Determine how the application will perform in terms of responsiveness and stability under different scenarios.

Acceptance Testing:

Acceptance testing is done by the project manager/stakeholder to ensure the app meets the requirements initially established for the project.

Regulatory Assurance:

Should be the responsibility of the project manager to ensure the app meets regulatory requirements.

Appendix D

Olympus Mobile Development Deployment Guidelines

Appendix E

Olympus Mobile Development Support And Maintenance Guidelines