Research Report
Use of objects in Graphic User Interfaces in Cocoa and Java Swing Foundation classes

Wayne Buchner,6643140

13-11-2011

1 Introduction

The Graphic User Interface allows the user to interact with a program by manipulating properties of objects there by sending information from the View to the Controller. Both Objective C and Java accomplish this in similar ways. This research report focuses on the differences in producing Graphic User Interfaces using Objective C's Cocoa and Java's Swing packages. Inherently the greatest point of difference in developing on the aforementioned platform is Java Swing's ability to be a truly cross platform framework (Sparke 2006). Where as Apples Cocoa is a powerful platform centric framework constructed purely for the Mac OS and iOS platforms. This provides a focused set of frameworks.

Dias & Oliveira (2011) states that writing code can be a daunting experience for novice programmers and that designing in an interface using direct manipulation of objects is a powerful learning tool to what was once a steep learning curve. Research undertaken for this document will provide proof that advances in Graphic User Interface builders significantly reduce code writing time, increasing efficiency and reduce errors. Removing unnecessary debugging time from the development lifecycle by reducing errors and increasing code quality as GUI builders improve, may reduce development times and allow developers to concentrate on other areas. Tests results generated for this paper are the results of timed tests between Apple's native integrated development environment, XCode Oracles Netbeans. Results are the comparison in improvements to code completion and build times.

Section 2 will introduce the Graphic User Interface architecture and explain at a high level the structure of the class hierarchies including an introduction to the Model View Controller (MVC) principals. **Section 3** compares available platform generic frameworks. **Section 4** explains in detail the differences between information passing and event handling across the two platforms while.

2 Graphic User Interface Architecture An Introduction

A Graphic User Interface allows the to user interacts with a system. A well designed interface should leverage a users existing knowledge there by improving the effectiveness of the users interaction. Both Cocoa and Swing allow for the layering of objects to be built in a hierarchical manor in order for the user to perform tasks, or in programming terminology trigger events. Although they achieve this in different ways, the semantics are minute in terms of actual designing, where Cocoa generally moved towards a design process based more on WYSIWIG allowing for more productive workflow, other development languages have been slow to implement or follow this lead. Developing a Graphic User Interface in Swing can be achieved either through a plain text file containing the code for a window or via a WYSIWIG editor found in IDE's such as BlueJay, Netbeans or Eclipse.

Both platforms isolate the view from the data by implementing the Model View Controller design pattern (further discussed in Section 4) and the similarities do not end there. Both inherit all core functionalities from a root Object class, NSObject for Cocoa and the JComponent class for Swing. Briefly, although the terminology may be different, both frameworks create an interface in a similar manor. Table 1 below shows the counterpart for main GUI objects that are shared across both frameworks.

As you can see in Table 1 both platforms fundamentally build a Graphic User Interface using similar constructs. The class names may be different however the principal behind them is similar. Where Java Swing endeavors to supply a cross platform framework, Cocoa is tailored to the MacOS and iOS platforms offering a more rigid, extensible framework build foremost with the platforms specifications.

Comparative Objects	
Cocoa - NSObject	**Swing - JComponent**
Window	JRootPanel
UIView	Container, View
Subview	JPanel
UIScrollView	JScrollPane
TabBar	JTabbedPane

Table 1: Average time of 20 build cycles for a simple Hello World Program

2.1 View Hierarchy in Cocoa

Figure 1 shows Cocoa's UIKit Class Hierarchy. Note that all objects inherit directly from the root class of NSObject. Also being more platform specific it has platform centric objects targeted at specific hardware devices. Building a Graphic User Interface using Cocoa has been further simplified by the employment and development over countless iterations of the Integrated Development Environment, XCode. XCode's interface builder offers detailed information on objects and offers only objects appropriate for use in the particular project, saving the developer time implementing unsupported user interface elements. As an aside, the newest iteration of XCode allows for storyboarding of windows to further simplify navigation between windows.

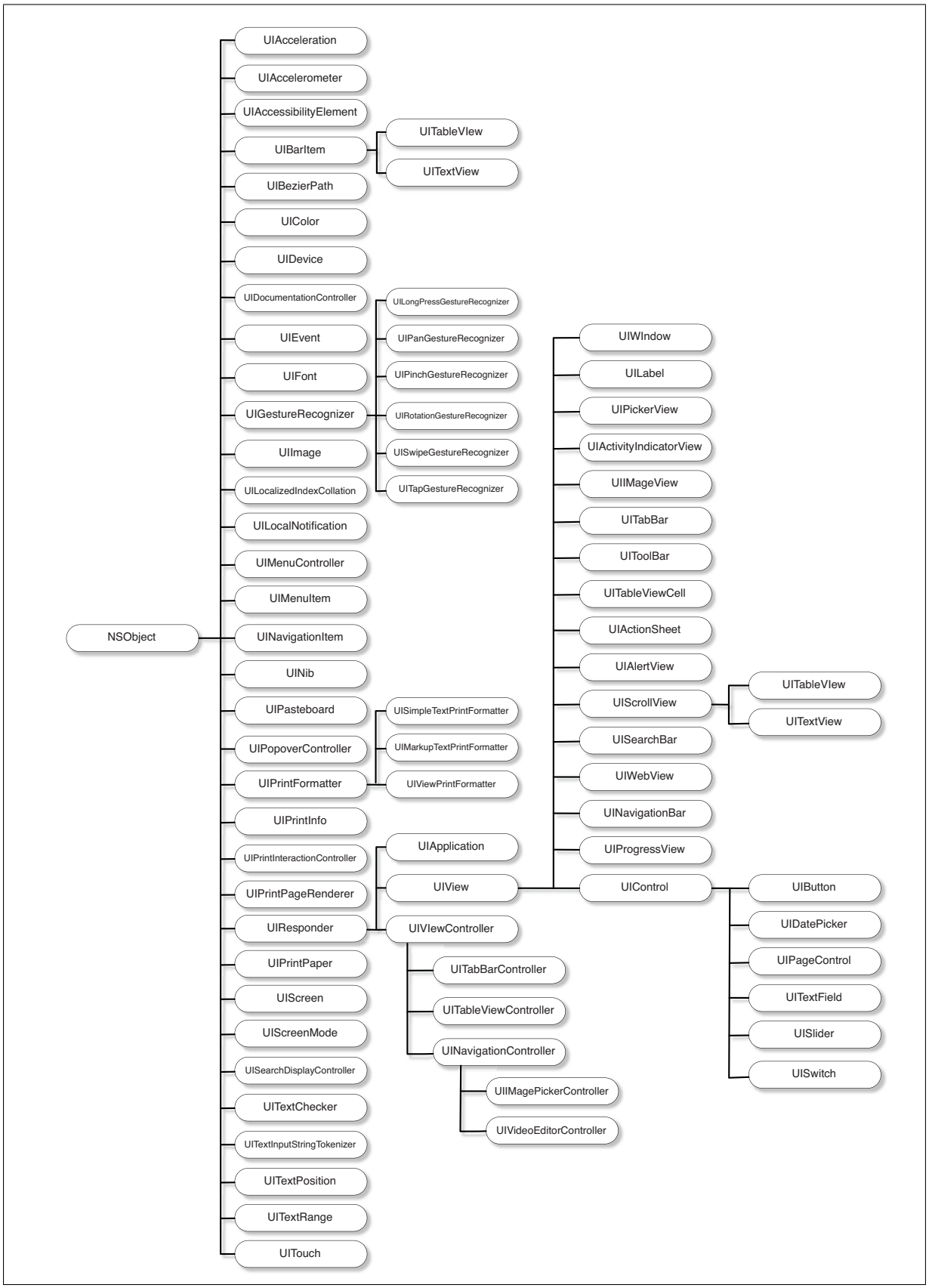

Figure 1: Cocoa UIKit Class Hierarchy. Apple (2010)

2.2 View Hierarchy in Swing

In Figure 2 you can see that Swing's hierarchy is configured only slightly differently, with JComponent inheriting from the Container class, whose root class is Object. Java Swing is a lightweight framework built on top of Java's awt framework. Due to it's completely Java construction it can be easily mixed with other Java Component class objects such as standard Java AWT labels, text fields and scrollbars.

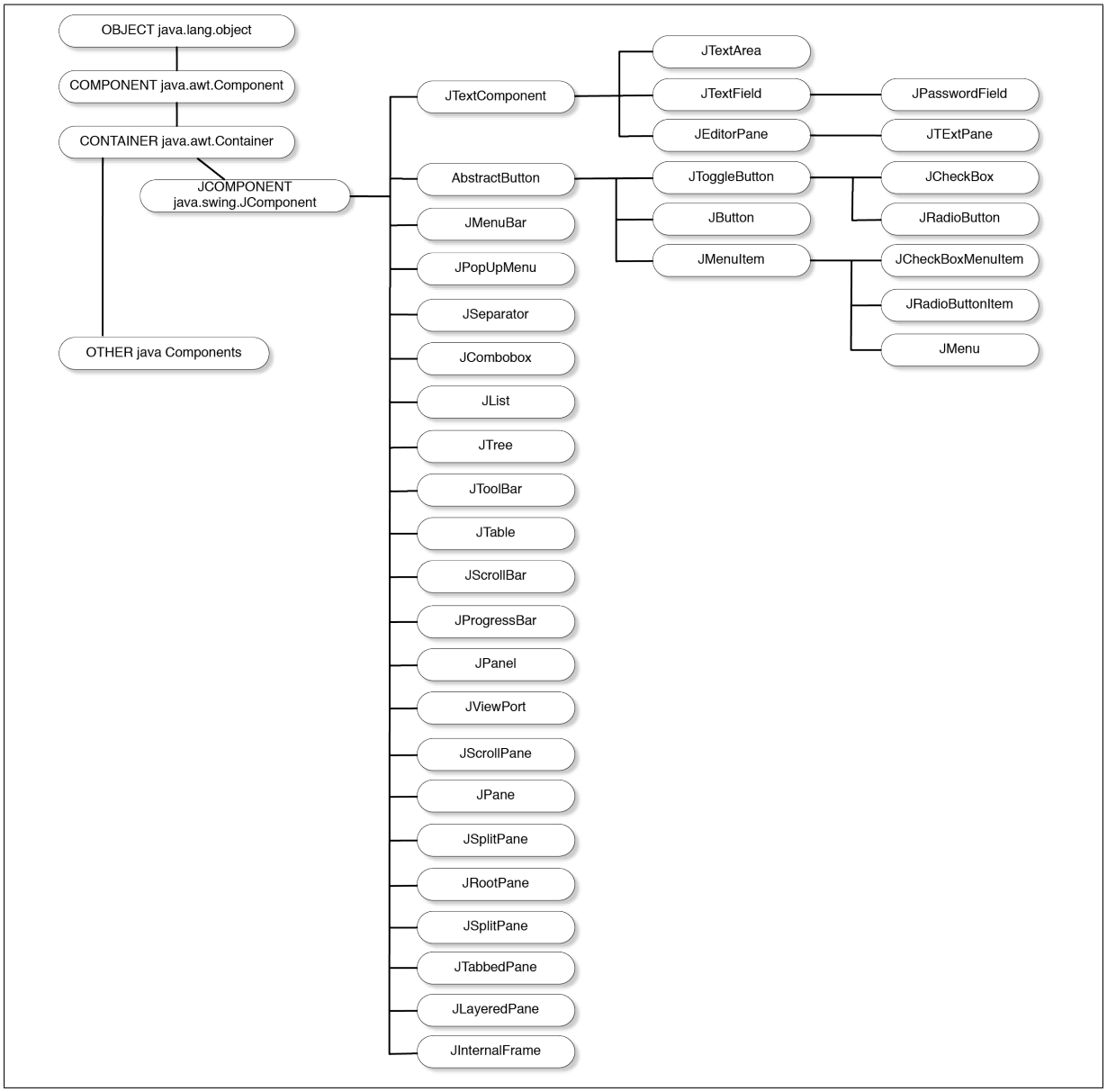

Figure 2: Swing Class Hierarchy. Oracle (2010)

2.3 Hello World

Moving from text based user interaction to a Graphic User Interface the understanding of how input is received changes. A programmer must shift from taking the result of a direct prompt to listening for user interactions.

This is called event based programming and that is how Graphic User Interface's work. Both Swing and Cocoa work in this manor. In both frameworks, **Event Listeners** respond to interaction with specific elements of the UI.

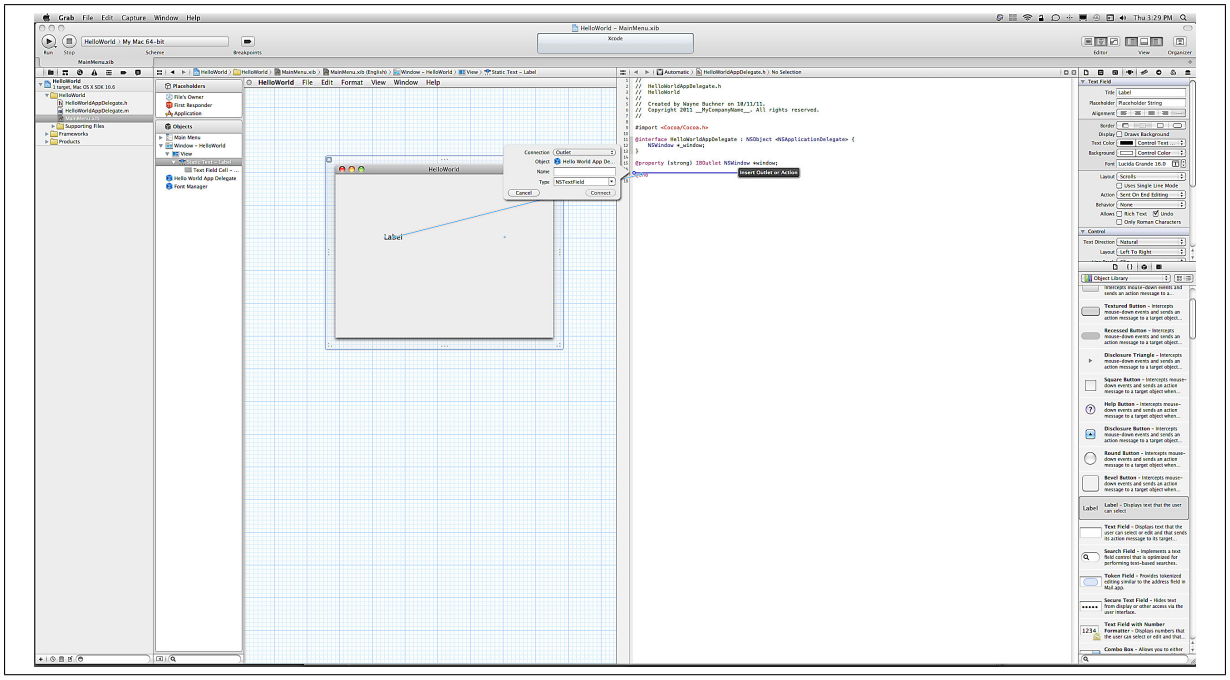

Figure 3: Interface Builder, XCode 4.1

Figure 3 shows the simplicity of **Hooking Up** an object created in interface builder to the view controller. Once an object has been created and Hooked up to a view controller or a delegate, it then becomes available to manipulate throughout the class via generated code placed directly into the corresponding header and implementation files. XCode places the correct getter and setter methods for the properties of the object via the @synthezise declaration. Java's design view works in a similar way Figure 4 shows the Design View for a Java Swing window. Selecting the object in the design view and adding an **Event** places an event listener method into the class ready for code in a similar way that XCode places the method into the implementation file.

Constructing the user interface with the assistance of a Graphic User Interface builder proved extremely fast on both platforms. Table 2 compares the user interface construction and build times for the test Hello World program using hand coding versus the IDE's assisted user interface builder. Based on the results constructing the Hello World program, developing time for a complex user interface like the interface shown in **??** will be dramatically improved using an Graphic User Interface builder. The figures in Table 2 would indicate there is no noticeable difference in build time for either platform when comparing between the programs hand coded versus auto generated code. Where the figures return an indication of difference lie in the ability to speedily outlay a user interface.

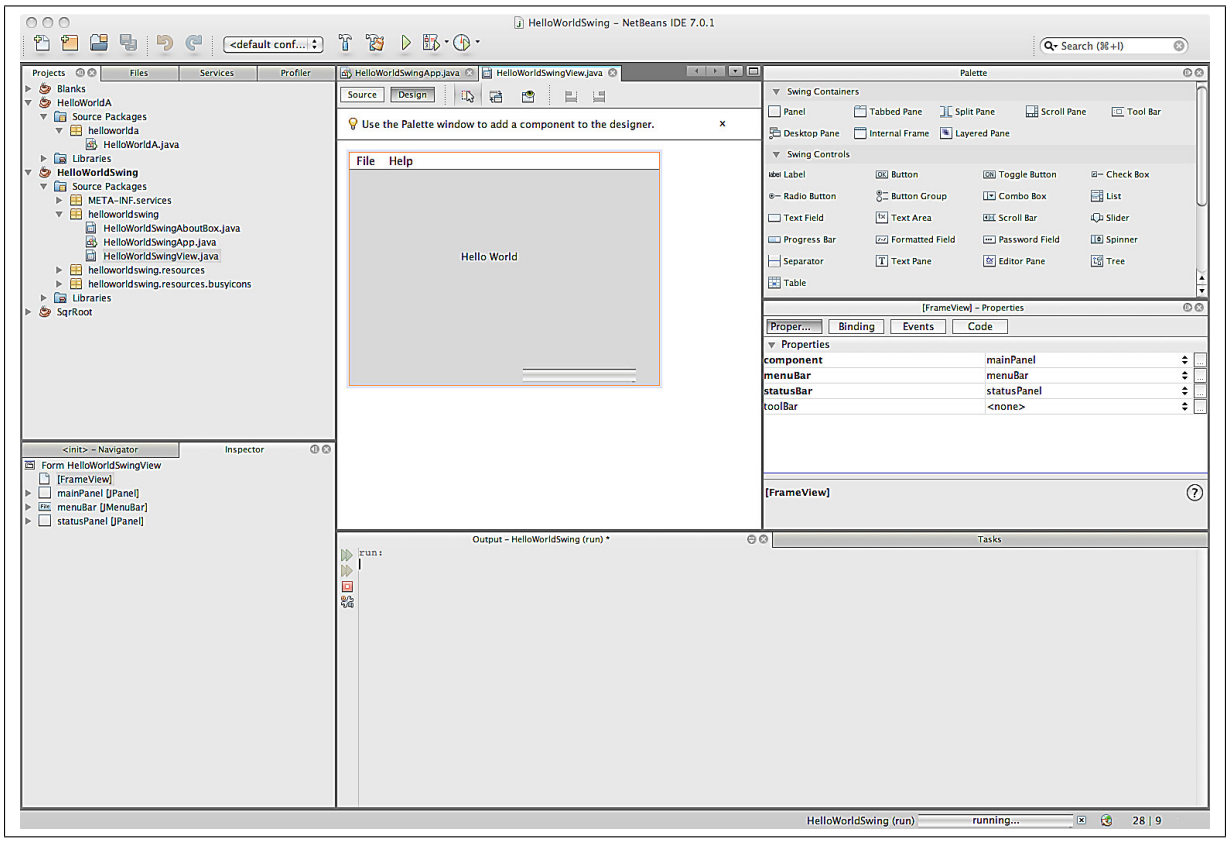

Figure 4: Design View, NetBeans 7.01

	X-Code	**X-Code** Interface builder	**X-Code** iOS	**NetBeans** (No gui builder)	**NetBeans** GUI Builder
Code Time	01:56.2	00:42.0	01:37.0	02:34.9	00:47.7
lines of code	5	1	4	10	1
Average Build Time	38.7ms	31.085ms	57.165ms		16.395ms

Table 2: Average time of 20 build cycles for a simple Hello World Program

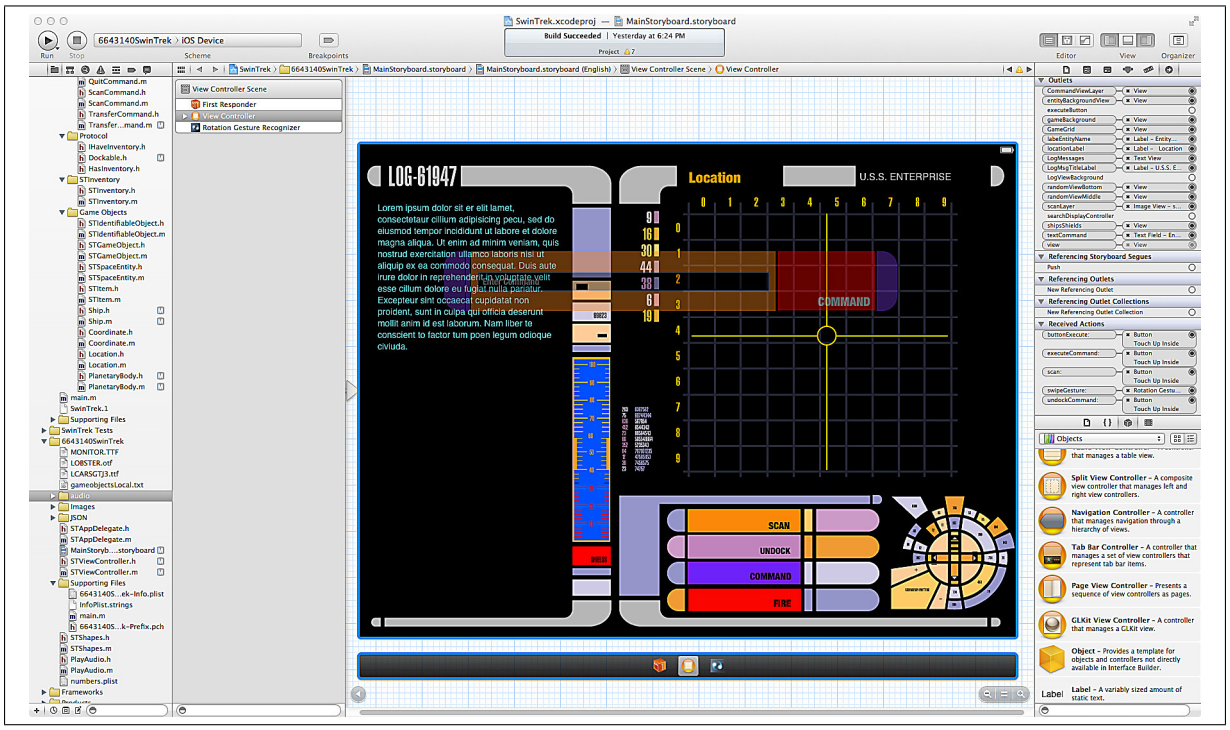

Figure 5: SwinTrek User Interface, XCode

Getting and **Setting** objects properties created using GUI builders in these two instances are no different over the two platforms that creating objects in code. Object properties can be manipulated through method calls or dot notation in both platforms. Infact there is only a small difference between the two in this case and that is to change a simple property on an object in Java, an event must be called or the getter or setter for that object must be called from within an event. Apple set up a series of methods where code can be inserted and ran. Where as Java, all *** HERE set up the differences between the two platforms and the way objects are manipulated ***

3 Framework specific objects

Both frameworks have developed comprehensive lists of classes that are available to a programmer to perform almost any required tasks

4 Object Information Passing

References

Apple (2010), *Cocoa Fundamentals*, Apple Inc, chapter 1, p. 46.

Dias, P. & Oliveira, S. (2011), Meet and greet programming using graphical languages and tangible interfaces, *in* 'Information Systems and Technologies (CISTI), 2011 6th Iberian Conference on', pp. 1 –4.

Oracle (2010), Java 2 sdk se developer documentation, *in* 'Hierarchy For Package javax.swing'.
URL: *http://download.oracle.com/javase/1,5,0/docs/api/javax/swing/package-tree.html*

Sparke, G. (2006), *The Java way : an introduction to Java programming / Gerard Sparke*, Pearson Education Australia, Frenchs Forest, N.S.W. :.