Introducing Universal Applications for iPhone OS

Introduction

With the introduction of iPad, iPhone SDK 3.2 now supports development of three types of applications: iPhone apps, iPad apps and Universal apps.

- **iPhone Applications**. iPhone applications are optimized to run on iPhone and iPod touch. These applications run on iPad in their original resolution or can be optionally set to be pixel- doubled to accommodate for the larger display.

- **iPad Applications**. iPhone SDK 3.2 supports the development of iPad applications that are optimized to take advantage of iPad features, but run only on iPad.

- **Universal Applications**. iPhone SDK 3.2 supports the development of Universal applications. A Universal app is optimized to run on all iPhone OS devices—it's essentially an iPhone app and an iPad app built as a single binary.

This document covers the guidelines you should follow to create well-designed Universal applications.

A Universal app can determine which device it's running on and provide the best experience for that device. Well-designed Universal apps leverage a device's unique hardware features, provide the right choice of user interface elements, and use only the functionality that is supported by that device.

Apple recommends that iPhone OS developers design, code and build their applications as Universal applications. Building apps as Universal apps makes it easier for everyone. It's easier for developers because there's only one app to manage on the App Store. It's easier for users because they'll know that it runs on any device they own.

Design for Universal

Whether you're thinking of optimizing an existing iPhone application or creating a brand new one that takes advantage of iPad, you should take the opportunity to design your apps as Universal apps.

An important approach to designing a Universal application for iPhone OS is to think about how user interaction can be separated from the underlying application code. The iPhone SDK's classes and APIs leverage a model-view-controller (MVC) paradigm that encourages a clean separation of your application data and logic from the views used to present that data.

The first step in making a Universal app is to create user interface designs for each of the form factors—one design for iPad devices and another for iPhone/iPod touch devices. Much of your design will be affected by the features you want to expose in each of the different form factors.
The following are some examples of user interactions to think about as you separate your iPad designs from your iPhone/iPod touch designs:.

Orientation	Using the accelerometer, it's possible for an application to detect the current orientation of an iPhone OS device. With that information it's possible for an app to optimize the user experience using alternative layouts to accommodate for the orientation. This is supported by a number of iPhone applications, including Mail, iPod and Photos. While it's not always desirable to support multiple orientations for an iPhone app, it's very important that all applications support multiple orientations when running on iPad.
Layout	Use the larger iPad screen to give people access to more information in one place, to increase focus on the content that people want, to flatten the hierarchy of screens in your app, to drastically reduce full-screen transitions, and to add realism, physicality, and stunning graphics to your application's user interface.
Gestures	The larger size of iPad's screen makes a wider variety of gestures possible. For example, it's conceivable that a user could perform a gesture with four fingers on iPad, but would probably not find it easy on iPhone or iPod touch. A well-designed Universal app design would accommodate differences in gesture-based input.

Split views	A split view is a unique capability of iPad which provides a new way for an application to present information side-by-side. It's important to carefully consider how to take advantage of the flexibility that split view offers on iPad, while still providing users access to the same information when the application is running on an iPhone or iPod touch.
Popovers	Similar to split views, popovers present a new way to present options to users. A well-designed Universal application will present the same options on iPhone and iPod touch using existing iPhone application user interface paradigms.
Hardware features	iPad, iPhone and iPod touch share much of the same hardware functionality. If you take advantage of a hardware feature that's not available on other devices, you should ensure that the lack of hardware on the other devices is handled gracefully. For example, an application running on iPad or iPod touch would not present the option to take a picture with a camera, but it could present them choices of images from the user's Photo library.

Conditional Coding

In order to achieve your design goals for a Universal application, you will need to use conditional coding to determine the availability of features when your app is running. Conditional coding allows you to make sure you're loading the right resources, using functionality that's supported by the device and properly leveraging hardware that's available.

Remember to make conditional code decisions at the most granular level you can. Try to avoid making assumptions based on device type or an OS version number. If you need to follow alternate code paths based on a method, then check for that method, not whether you're currently running on a specific device type. Making code path decisions at the lowest possible level allows your app to automatically pick up new functionality when that method or class is introduced in an OS software update, your app will now have the functionality without you having to create an update.

These are the categories of things that you may need to conditionally code for:

Resources	**Interface Builder files (nibs)**. In your code, you need to recognize which platform you're running on and load the appropriate resources for each. For example, to ensure you present the right interface for your application you may need two nibs for a view, one that contains the controls for an iPhone interface and the other for the iPad interface. Your code will need to check to see which device you're running on and choose the appropriate nib file. **Graphics**. And you will probably want to load different resources for each device. For example, your iPad graphics might be larger than your iPhone graphics.
Classes	Some Cocoa Touch classes are only available on certain devices. Before you use a class, you need to check whether the class is available. For example, iPhone OS 3.2 allows your app to share documents with other applications. That functionality is only available for iPad. Your code would test for the existence of the `UIDocumentInteractionController` class using `NSClassFromString()` which will return a valid class object if this class exists or nil if it doesn't. If the class does exist, your code can use it and share documents. If not, you'll need to work around it so the app works properly on devices that don't support sharing documents.
Methods and functions	Some devices support different methods and functions. If you are using functions on one device that don't exist on the other you need to weak-link to those functions and perform runtime checks before you call those methods. The base class for most Cocoa objects, `NSObject`, has built in support for runtime checking of method availability. The methods `instancesRespondToSelector` and `respondsToSelector` let you determine if an object or its' superclasses have the method you want to use on the device you're running on. For example, if you want to search an `NSString` using a regular expression you would test the `NSString` class to see if it responds to `NSRegularExpressionSearch`. If it does then you would invoke the search in the `NSString` class. If the result is `NO`, then you would switch code paths to use the regular expression searching code you are using on iPhone prior to iPad.

New APIs in existing Frameworks	New APIs added to frameworks work in a similar fashion. For example, UIGraphics for iPad adds support for PDF destinations. In this case UIGraphics exists on all version of iPhone OS, so you would not be checking for the framework. Your code would check for the specific PDF functions that you want to use. Your code would compare a function name like `UIGraphicsBeginPDFPage` against `NULL`, if it's equal to `NULL` then you know that the function is not available on this device. If it's not `NULL`, your code continues to use that function.
Hardware capabilities	Sometimes it's necessary to detect whether certain hardware is available. You would use the same methodology you use to see if a software method or function is available, for example if you were trying to determine if the device you were on had a camera, you would query the `UIImagePickerController` for the `UIImagePickerControllerSourceTypeCamera` source, and make a programatic decision from there if this returns false.

The SDK Compatibility Guide contains detailed information on weak-linking and building code that takes advantage of different frameworks and available APIs. The code sample MailComposer demonstrates the actual code you can use to make your app configure itself dynamically to take advantage of the OS features you have available at runtime.

Building a Universal app will reward your customers with a great experience no matter what device they run your application on, and will make management of your apps on the App Store simpler. Design well, write good conditional code, and build great apps!