Migrating iPhone 3.x apps to iPad and iOS 4.0

Adrian Kosmaczewski

2010-07-26

Right now, creating Universal Applications[1] for the iPod touch, the iPhone and the iPad is not really a straightforward task. The current panorama of iOS-compatible software and hardware platforms is getting more and more complex, and this blog post is a small guide (by no means exhaustive) of tips and tricks that have helped me get my apps running in as many platforms as possible, with as few headaches as possible.

First of all, a few warnings:

- I assume your current applications run in iPhone OS 3.1 as a minimum requirement. Apple does not accept any more applications targeting the 2.x frameworks these days, and the developer tools do not include those anymore.
- I assume your projects compile and link without warnings[2]; many API calls have been deprecated between 2.x and 4.0, so you'd better have code using the latest methods and no deprecated ones.

Panorama

Waiting for an unification of the iPad and the iOS 4 frameworks (probably later this year[3]), the current panorama of software and hardware iOS platforms looks as follows:

	iPhone			iPod touch		iPad
	3G	3GS	4	2nd gen	3rd gen	
iPhone OS 3.1	x	x		x	x	
iPhone OS 3.2						x
iOS 4.0		(ls)	x	(ls)	x	
Year Released	2008	2009	2010	2008	2009	2010
RAM (MB)	128	256	512	128	256	256
CPU (MHz)	620	833	1 GHz	620	833	1 GHz

(ls): limited support, particularly for multitasking.

[1] http://devimages.apple.com/iphone/resources/introductiontouniversalapps.pdf
[2] /blog/objective-c-compiler-warnings/
[3] http://adage.com/digital/article?article_id=144670

For practical purposes, I've omitted in the table above the fact that the iPhone 3G is (theoretically) able to run iOS 4. Something that has been empirically proved to be a bad idea[4].

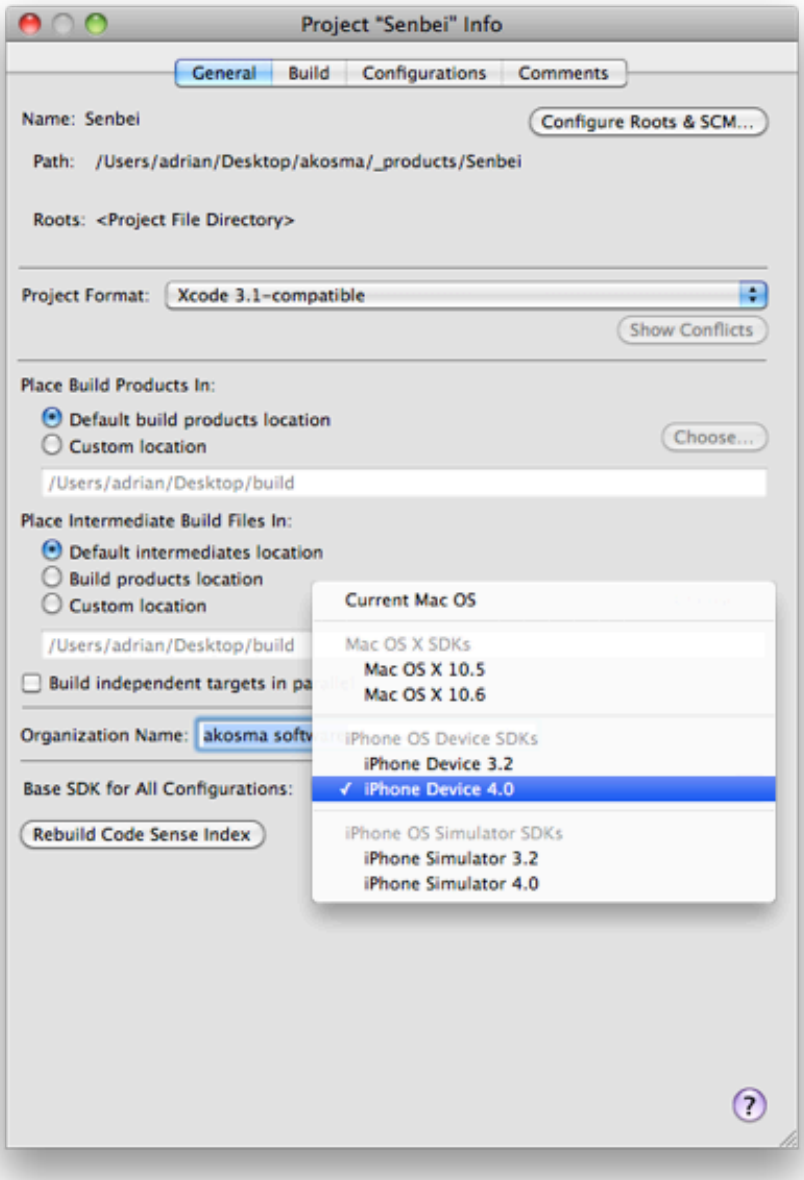
Upgrading Xcode Projects to iOS 4

Your Xcode 3.1 iPhone project can be migrated to iOS 4 as follows:

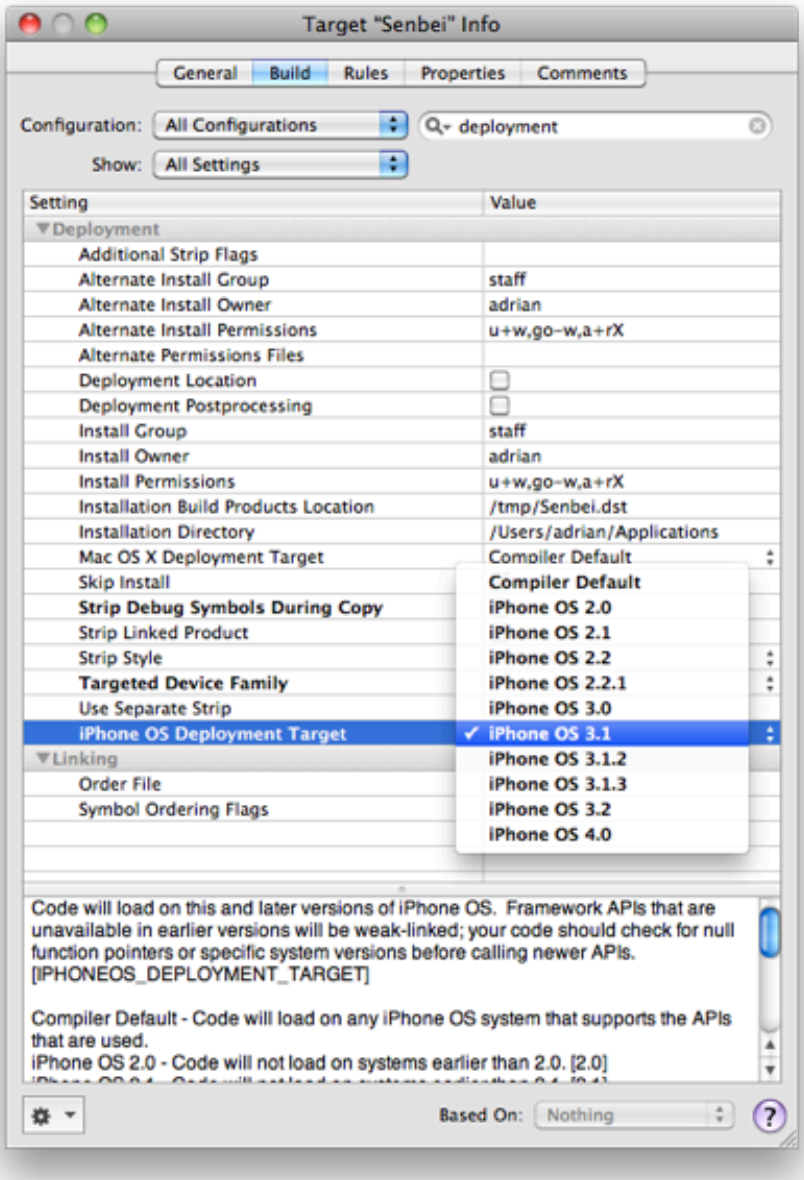
- If you are using an SCM with branching support, create a new branch for all of the operations below; you don't want to impact your "trunk", or main branch with all these changes, as more urgent bugs might appear while you are doing this.
- Download the latest SDK from Apple[5]. This is required by Apple, actually. And the latest version of Xcode (3.2.3 at the time of this writing) allows you to write applications compatible with all the platforms in the table above.
- Open your project with Xcode 3.2.3; select the "Project / Edit Project Settings…" menu and change the "Base SDK value to iPhone Device 4.0 (yes, even if your project is meant to target iPhone OS 3.1… stay with me!)

[4] http://www.tuaw.com/2010/07/22/ios-4-and-iphone-3g-is-a-match-made-in-whats-the-opposite-of/

[5] http://developer.apple.com/iphone/

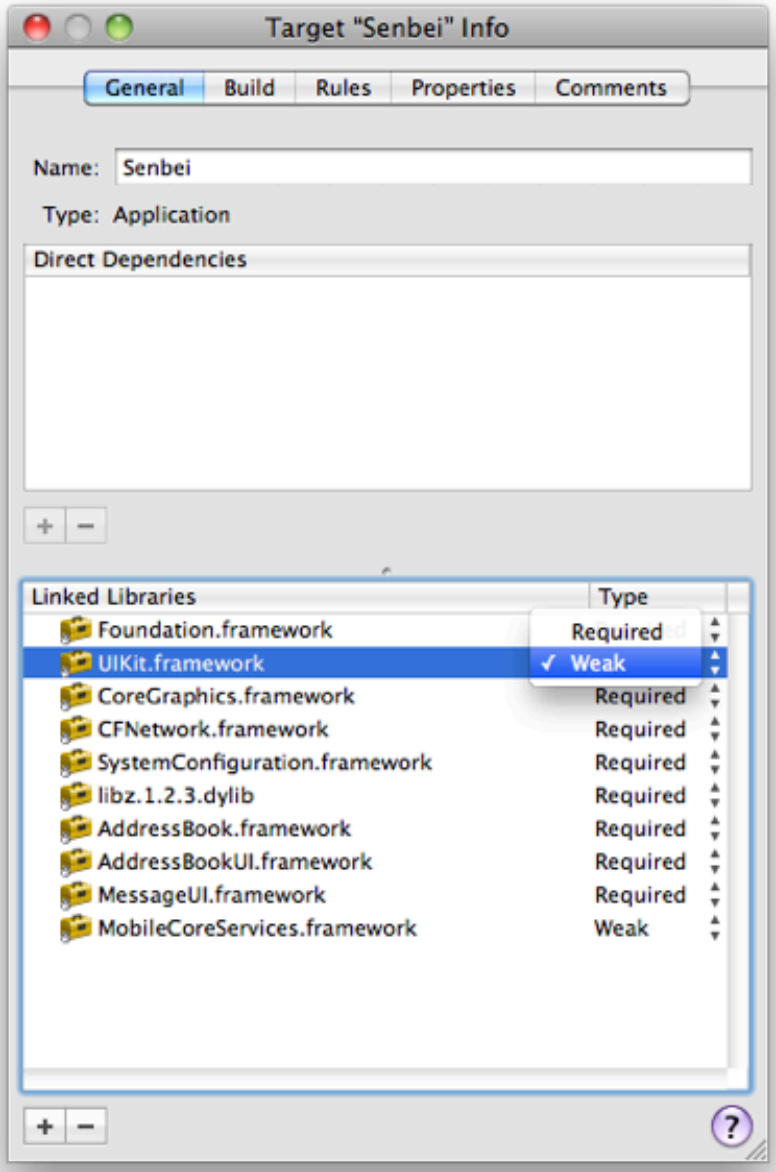


- In the "Targets" group of your Xcode project, open the "Info" panel, select the "Build" tab and change the "iPhone OS Deployment Target" entry for all the configurations of your target to "iPhone OS 3.1". This will allow your code to be loaded in earlier versions of iOS, even if your application uses iOS 4 as a base SDK for all configurations:



- Set the UIKit framework, as well as any new framework provided by iOS 4, to link as "Weak"; for example, in the screenshot below, taken from the settings of my open source Senbei application[6], you can see that UIKit is weakly linked, as well as the MobileCoreServices framework (which does not exist in iPhone OS 3). This prevents the application from crashing at startup in versions of iOS earlier than 4, because newer versions of UIKit bring new classes and APIs that were not available before:

[6] http://github.com/akosma/Senbei



- Change your Entitlements.plist file, because iOS 4 applications submitted to the App Store require a new format in the Entitlements.plist file. In most cases, you can just delete the previous file, and create a new one from Xcode ("File / New File…", then select "Code Signing / Entitlements" in the dialog that appears).

Old Entitlements.plist:

```
<plist version="1.0">
```

```
<dict>
    <key>get-task-allow</key>
    <false></false>
</dict>
</plist>
```

New Entitlements.plist:

```
<plist version="1.0">
<dict>
    <key>application-identifier</key>
    <string>$(AppIdentifierPrefix)$(CFBundleIdentifier)</string>
    <key>keychain-access-groups</key>
    <array>
        <string>$(AppIdentifierPrefix)$(CFBundleIdentifier)</string>
    </array>
</dict>
</plist>
```

- Update your icons and graphics for the new "Retina" display of the iPhone 4; Apple has published a special guide[7] that provides all the information required in terms of sizes, formats and naming conventions.

After doing the steps above, you should have an application running on iPhone OS 3.1, 3.2 (as an iPhone application) and 4.0, offering exactly the same functionality and ready to be sent to the App Store.

Create a Universal iPhone and iPad application

Creating an iPad-compatible application from your iPhone app requires much more than just adapting your source code; the user experience of iPad apps is a completely different beast than that of iPhone apps, so I strongly recommend

[7] http://developer.apple.com/iphone/library/qa/qa2010/qa1686.html

you ask a graphic and/or UX designer for help[8]. I can't stress this too much; iPad apps aren't just "big" iPhone apps. They are much, much more.

Technically speaking, in the simplest of terms, you should be able to migrate most UIKit-based application following the standard path provided by Apple:

- Select your target in Xcode and choose the "Project / Upgrade Current Target for iPad..." menu.
- Use some compile-time and runtime tricks to get different parts of your code to execute in different environments, as explained by Jeff LaMarche[9]:

```
#if __IPHONE_OS_VERSION_MAX_ALLOWED >= 30200
    if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad)
        NSLog(@"iPad Idiom");
    else
#else
        NSLog(@"iPhone Idiom");
#endif
```

Testing your code in different devices

My advice is the following: don't upgrade your old iPhone to iOS 4, particularly if you own a 3G. Keep it running under 3.1 and get the new iPhone 4 as soon as you can. This has two major advantages:

- You will be sure that there aren't crashes or unexpected situations in iPhone OS 3.1, particularly when you are using new APIs in your code.
- The iPhone 3G and 3GS are slower than the new iPhone 4, but they are still (probably not for long) the most widely deployed, particularly in those countries where the iPhone 4 has not yet been released. This will help you create code that uses low memory, and that runs faster in newer models.

In my personal situation, I have my old iPhone 3G running iPhone OS 3.1.3, and a second generation iPod touch running iOS 4. And of course, my iPad runs iPhone OS 3.2.1.

Adapting your code for different platforms

Although Apple recommends using the UI_USER_INTERFACE_IDIOM macro[10], sometimes you need a tighter control in your code. For this, I have used three different techniques:

- Use a different class as the app delegate in your iPad application; instead of making your code a mess of "if" and "else" statements or preprocessor routines, make your iPad application create an instance of a different class, one that might or might not be a child class of the existing application delegate. This way, you can completely separate the code of both platforms, and each can load a different UI control hierarchy, without impacting

[8] http://www.smashingmagazine.com/2010/04/16/design-tips-for-your-ipad-app/

[9] http://iphonedevelopment.blogspot.com/2010/04/few-more-notes-on-creating-universal.html

[10] http://developer.apple.com/iphone/library/documentation/General/Conceptual/iPadProgrammingGuide/StartingYourProject/StartingYourProject.html

the existing iPhone OS 3.1 or iOS 4.0 infrastructure. You can specify a different class for your iPad app delegate directly in your MainWindow-iPad.xib file, created for you when you selected the "Project / Upgrade Current Target for iPad..." menu (one more advantage of using Interface Builder files in your projects!):

- Check for symbols at runtime, taking advantage of the dynamic nature of Objective-C; for example, the code below only executes in iOS 4 or later, because the ADBannerView is not available in previous versions of the iPhone OS. This code will run without problems in iPhone OS 3.1! You can also use NSObject's "respondToSelector:" method to perform runtime checks of the capabilities of your current platform, which might help you branch your code in different directions depending on the context:

```
- (void)showAdvertising
{
    Class klass = NSClassFromString(@"ADBannerView");
    if (klass)
    {
        ADBannerView *adView = [[ADBannerView alloc] initWithFrame:CGRectMake(0.0, 416.0,
        adView.currentContentSizeIdentifier = ADBannerContentSizeIdentifier320x50;
        adView.delegate = self;
        [self.view addSubview:adView];
    }
}
```

- Include and use in your project the UIDevice extensions by Erica Sadun[11]. This very handy set of categories provides a wealth of options, making your code much more readable and easier to understand. Most importantly, it is actively maintained by Erica, and this means that future versions of the categories will be able to detect more features of iOS.

Finally, MPMoviePlayerController requires special attention, because it's widely

[11] http://github.com/erica/uidevice-extension

used in many applications and also because it has been greatly improved (and changed) between iPhone OS 3.1, iPhone OS 3.2 and iOS 4. Those changes include, in some cases, deprecated APIs, so here I'll just link to this excellent article by John Muchow[12] in the iPhone Developer Tips blog, which explains all the problems and the possible solutions.

Conclusion

As you could see, creating universal apps is not easy, but it isn't impossible either; it requires a bit of attention and lots of testing. Do you have any tips or links that you would like to share? Feel free to add more information in the comments below.

[12] http://iphonedevelopertips.com/video/getting-mpmovieplayercontroller-to-cooperate-with-ios4-3-2-ipad-and-earlier-versions-of-iphone-sdk.html