SDKs and Deployment Targets

Xcode has two build settings that are very important when supporting multiple versions of the OS (whether we're talking about the Mac or the iPhone, it's all the same to Xcode) with a single application binary. These settings are `SDKROOT` (a.k.a *Base SDK*) and `MACOSX_DEPLOYMENT_TARGET`[1] (a.k.a. *Mac OS X Deployment Target*).

Base SDK

Most people are familiar with the Base SDK setting. This setting defines which SDK to build against, and therefore, which APIs are available for your use. If, for instance, you want to build an application that uses an API that was introduced in 10.5, you *must* have a Base SDK set to `macosx10.5` or later[2].

Deployment Target

Were Base SDK the only setting, you would be limited to three options:

1. Build multiple versions of your application, one fore each OS version that you want to support
2. Build a single application that uses features from the newer OS, and ignore users of the previous version.
3. Build a single application that runs on the previous version of the OS, but miss out on the new features of the newer OS.

None of those choices seems particularly appealing. Fortunately, that is where the deployment target setting comes in. The deployment target setting, which seems to confuse people a bit, is much more subtle, and is often overlooked. It tells Xcode the version number of the earliest OS you wish to support, and adjusts the linkage of your application to make it possible.

Take the following example:

- There are two versions of a hypothetical OS: OS 1.0 and OS 2.0
- I have an application that I've been developing for OS 1.0, but I *really* want to take advantage of a new, amazing OS 2.0 feature.
- I have enough users that still run OS 1.0, and I don't want to deprive them of my application.

All I need to do to accomplish this is to set my Base SDK to "OS 2.0", which will allow my application to use the new feature, and set my deployment target to "1.0″, which will allow my application to launch on OS 1.0.

At Runtime

But there's a catch: I can *launch* on 1.0, but if I use a function, class or method that doesn't exist (like those required to use the new, amazing feature), my application will crash. So, the last thing that I need do is to check, at runtime, for the availability of the particular function (or class, or method, etc.).

Functions

For functions, I can accomplish this by comparing the address of the function to NULL:

```
if(APIForAmazingNewFeature != NULL) {
//I can call APIForAmazingNewFeature() here
} else {
//I can't call it here, but I can fallback to doing something sensible
}
```

Classes

Since Objective-C is more dynamic than C, I can use higher-level constructs when the feature I'm testing for is a class:

```
Class myClass = NSClassFromString(@"AmazingNewClass");
```

if(myClass) {
//I can use "myClass" in place of AmazingNewClass when calling class methods:
AmazingNewClass *instance = [[myClass alloc] init];
…
} else {
//The class doesn't exist
…
}

Added 11. Feb, 2010:

> If you're developing for an OS that supports Weak-import classes (currently only iPhone OS 3.1 and later), dealing with classes that may or may not exist is even easier—you can even subclass such classes (something that isn't possible in previous versions of the Mac and iPhone operating systems).

Methods

…or a method:

```
if([someObject respondsToSelector: @selector(methodAddedInVersion2)) {
[someObject methodAddedInVersion2];
} else {
…
}
```

In Conclusion

Using these two build settings you can easily build a single binary of your application that supports the latest and the greatest, yet gracefully degrades when running on older OSes. The basic rules of thumb:

- Set the Base SDK to the lowest value that will still support all of the features that you use
- Set the deployment target to the lowest OS version on which you plan to run

- Set the deployment target to the lowest OS version on which you plan to run.
- Check, at runtime, for any features that you use that don't exist in the version of the OS that matches your deployment target.

1. or `IPHONEOS_DEPLOYMENT_TARGET`/*iPhone OS Deployment Target* [↵]
2. There is one exception to this. If you leave this setting blank, it will build against the host system. So such an application would be properly built if, and only if, you were building it on a Mac running 10.5. Needless to say, it is impossible to build an iPhone application in this manner [↵]

June 23rd, 2009 in Uncategorized | tags: iPhone, Macintosh, Xcode

8 Comments

Chris SuterJune 23rd, 2009 at 17:31

One of the newer compilers (probably clang, but possible gcc-llvm—I forget), spews a warning when you do:

if(APIForAmazingNewFeature != NULL) {

The warning complains that the test is always TRUE, which suggests, although I haven't checked, that it's getting optimised away.

The warning appears to go away if you do:

fn = APIForAmazingNewFeature;
if (fn) {
// Do amazing stuff
}

Cédric LuthiJune 25th, 2009 at 07:51

Also good to know: the SDKROOT setting translates to gcc's -isysroot option, MACOSX_DEPLOYMENT_TARGET to -mmacosx-version-min and IPHONEOS_DEPLOYMENT_TARGET to -miphoneos-version-min.

Chris RylandJune 25th, 2009 at 11:40

@Chris Suter: Yes, but the new static analysis clang-based engine Xcode 3.2 would probably see those two as identical, and still give you a warning. I wonder if there's a better way.

Chris HansonJuly 1st, 2009 at 12:10

@Chris Suter: The compiler should only emit a warning for that if the declaration of APIForAmazingNewFeature isn't tagged with `__attribute__((weak_import))`, which is what indicates to the compiler that it can be NULL; otherwise, assuming that functions aren't NULL is a valid optimization.

This attribute (and `__attribute__((unavailable))` of course) are what the macros in AvailabilityMacros.h (old) and Availability.h (new) are for; they're set up to tag declarations with the appropriate attributes based on the combination of minimum required (deployment target) and maximum allowed (SDK) OS that you've chosen.

Emanuele VulcanoJuly 2nd, 2009 at 06:04

The correct way to prevent a compiler from optimizing away, anyway, is to use volatile. Otherwise, a cunning compiler may infer (and is allowed to) optimize away the use of fn.

BenjohnJuly 22nd, 2009 at 04:38

Thanks for this brilliantly useful info, I think my users are likely to appreciate it too. Some additional magic sauce that I've just found is that you need to weakly link in the "base sdk" frameworks that may not exist on the "deployment platform". You can read how to do that here: http://blog.dearprakash.com/2009/06/20/iprogrammer-tip-6-weak-linking-additional-libraries/