Using local Frameworks in Cocoa Plugins

The Problem

Ok, so there you are with your wonderful idea for a plugin for Mail or Address Book or any number of other apps under OSX and you sit down and knock out a quick bit of code to see if you can get something to work. And there is this great framework some cleaver bod put together that does pretty much everything you want to do so you figure you'll just use the framework, include it in the plugin bundle and everything will work, right? I mean, it works for Apps, doesn't it... Unfortunately, life is not that simple.

You see, when an Application is loaded, the dynamic loader (dyld) takes a look at the frameworks it needs and goes off to find and load them (if they are not already loaded). It does this by looking in the location the framework said it was installed into when it was compiled. If you are stuck trying to get a bundle framework to load you have likely come across the `@executable_path/../Frameworks` string you are supposed to use when compiling a framework for use within an application bundle; this is translated by the dynamic loader into just that - the path of the executable it's trying to load replaces the `@executable_path` part of the string. This all works fine, for Applications.

So what goes wrong when you try to do the same for a plugin bundle? The dynamic loader loads a plugin just like it would a framework or a nib. It considers the plugin to be part of the host application and, as such, will continue to use the host application executable path rather than the plugin path when referring to framework locations. When your plugin asks the dynamic loader to load a framework from it's frameworks folder, the loader actually looks in the *application's* frameworks folder. Unsurprisingly, it then can't find the framework.

The good news is that a little programming and some typically hacky missuse of technology will get everything working, as long as your plugin is only supposed to work in OSX 10.2 upwards.

NOTE: It seems that this hacksy doesn't work with C global vars or (possibly) C function calls. Therefore, at least while getting stuff working, I strongly recommend working ONLY with ObjectiveC methods. It also, therefore, stands to reason that this hack also won't work with C frameworks. I have not tested this so please let me know your experiences!

The Solution

Overview

Fundamentally, the task at hand is to convince the dynamic loader to load a framework from a folder it was never told about nor would ever consider to look in if left to it's own devices. It turns out that this stage is surprisingly simple, thanks to Cocoa's wonderful dynamic loading system (yes, I know, that same wonderful system that landed you in your current mess...). To the dynamic loader a nib, a plugin, and a framework are all just bundles. As such, you can load them. Whenever you want. Quite neat really.

However, it's all well and good saying you can just load the framework once your plugin has started but it is the fact that your plugin doesn't load in the first place that is currently vexing you. What we need is a way to load your plugin without the dynamic loader complaining about lost frameworks and yet, once the plugin has loaded, let it use those very frameworks the loader couldn't find. It turns out that those smart bods at Apple (who have seemed to have totally missed our current predicament) managed to give us this ability without telling us, in the process of dealing with framework backwards compatibility. They were trying to solve what should happen when code compiled to use a later version of a framework tries to load with an earlier version which does not contain all the same symbols (being older and less feature-rich). Their solution is "weak methods" which can be used by your program like normal but are not actually matched up at loading, only when first used. A good explanation of what this "weak loading" is supposed to be used for is available here. But the good news for us is that an entire framework can be set to weakly link and load! And our problem is solved.

HowTo

There are two parts to the solution, each matching one of the problems in the above section. First we need to setup Xcode to set our problem framework to weakly link our plugin during the build and then we need to write the code to actually load the framework before it is first used. If we don't do this, when it is first used the loader will throw the same error it would normally have thrown trying to load your plugin.

Configuring Xcode

Normally, you will notice that all the frameworks you use will appear in the Frameworks & Plugins subsection of the Target build for your plugin. These are your normally bound, normally loaded frameworks. We need to remove the framework you need to weakly load from this list so do so - either by deleting it or by un-ticking it (if you have the Target Membership column visible).

Next you need to explicitly add the framework to the build process as a weak framework. To do this, click on your plugin target and bring up it's properties (apple-i). In the build properties, find the "Other Linker Flags" entry (about 1/5 the way down the list, for me) and add the string "-weak_framework frameworkname" to whatever was there before (which is often empty). Framework name should not include the .framework extension.

Last, as with any locally bundled resource, you need to make sure your framework is copied to the build bundle so, if you have not already done so, add a copy stage to the build process and set it to the "Frameworks" sub-folder of the bundle. Then drag the weakly-linked framework into this stage so it is copied when you press build.

When you have done all this, your plugin should build without any (additional) errors or warnings and you can check by hand to make sure your bundle contains the weakly linked framework in the Contents/Frameworks folder.

The following pictures give an idea of the changes I made in my GPGKey plugin for Apple's Address Book. The first picture shows the GPGME framework (which I wanted to weak-link) where you would expect it to be - in the Frameworks build stage. The second shows it removed from the Frameworks build stage and added to a copy stage that is set to copy files to the build bundle's Frameworks folder. The third picture shows the setup of the Target Build settings to include the GPGME framework as a weakly linked framework.

Loading the framework

Now you have your framework setup to load weakly, you actually need to load it. If you don't, when it is first needed it will load as normal and fail, as it did before. However, as I mentioned above, frameworks are just another type of bundle so can be loaded with NSBundle (handy, don't you think!). You will need to get the framework loaded asap to make sure nothing tries to use it before you get it loaded in so I recommend loading it in the `+(void)initialize` method of the primary class in your plugin. This way, it will be loaded before any instances of the primary class are created and so, hopefully, before you ever need to use the methods in the framework.

Put simply, the following lines of code create an NSBundle instance pointing to your framework (working out where the framework is based on NSBundle's class info which does know we are in a plugin) then asks the instance to load, thereby loading in the framework and it's symbol tables:

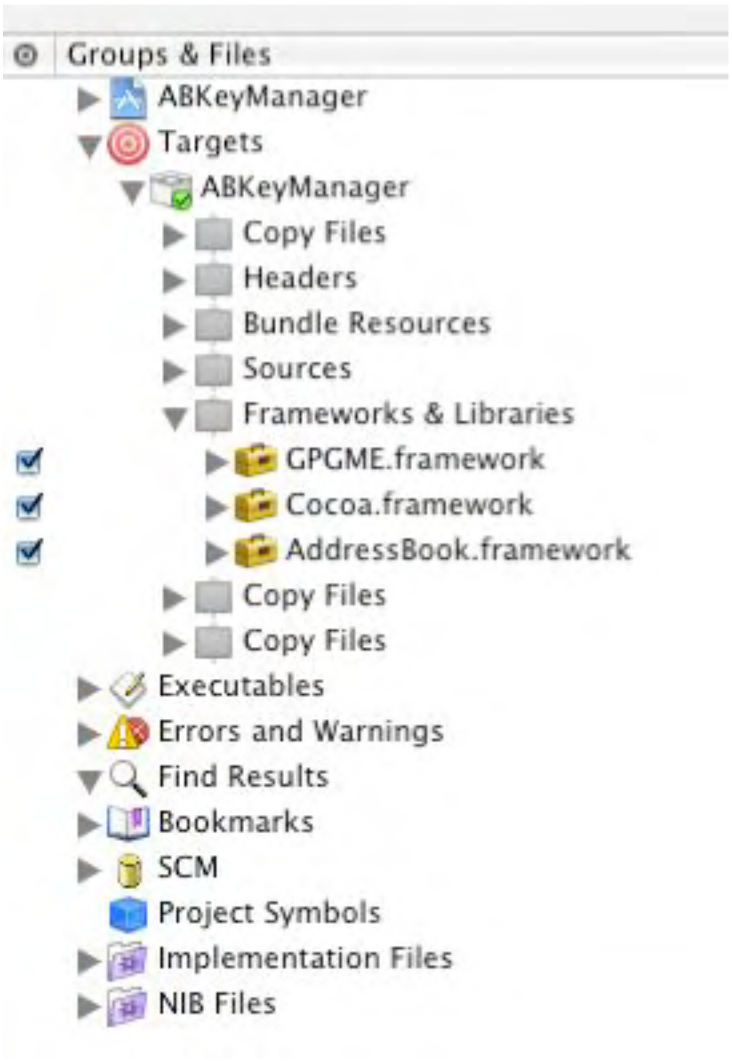
```
NSString *frameworkPath=[[[NSBundle bundleForClass:[self class]] bundlePath]
        stringByAppendingPathComponent:@"Contents/Frameworks/yourframework.framework"];

NSBundle *framework=[NSBundle bundleWithPath:frameworkPath];

if([framework load])
    NSLog(@"Framework loaded");
else
{
    NSLog(@"Error, framework failed to load\nAborting.");
    exit(1);
}
```

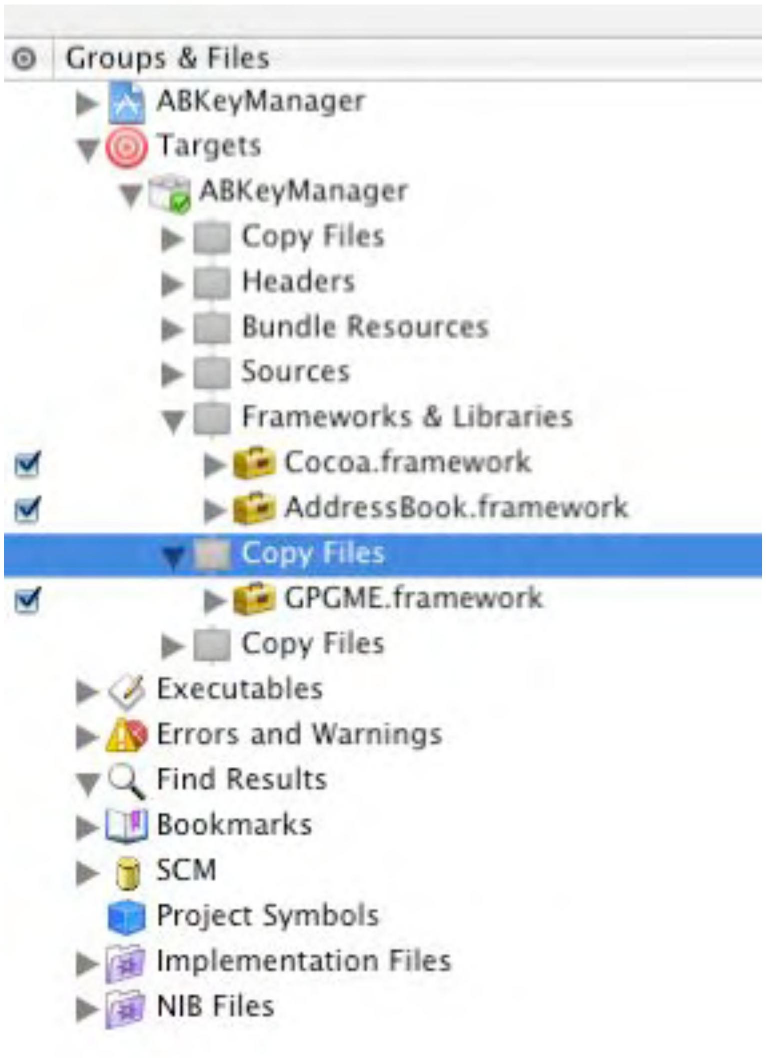
Once the framework is loaded, all the symbols are available in the app symbol tables and the dynamic loader is not called to load the framework again. As a result, you can now use the framework to your heart's content :)

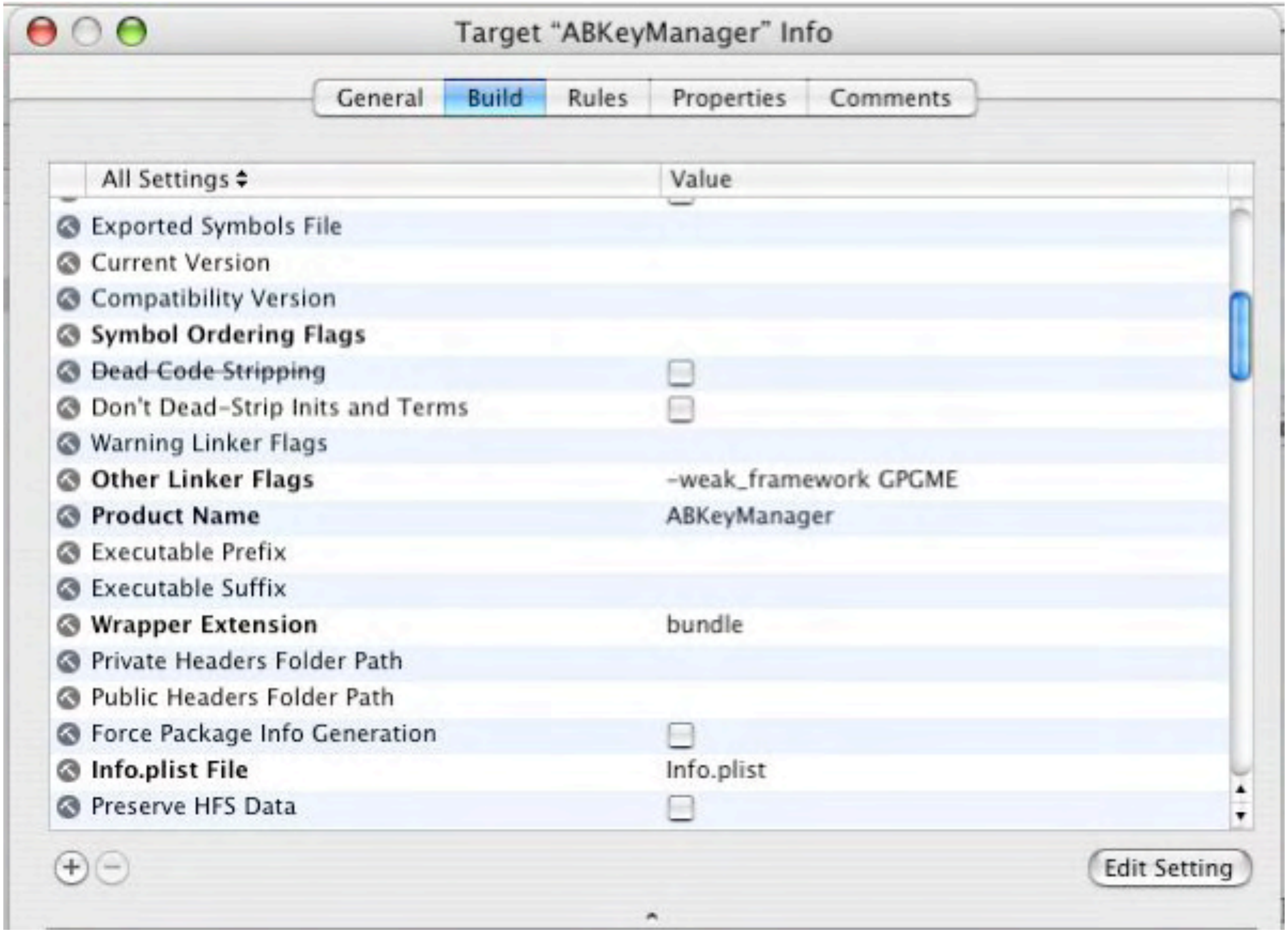
Potential Problems

- This only works in OSX versions that support weak binding. Earlier versions ignore the weak binding info and just crash out as you are currently experiencing.
- I've been told that you need to be using a native Xcode project, not an imported ProjectBuilder project for you to be able to see the dialogues you need to see to set up weak linking.

Happy Coding!


Groups & Files
ABKeyManager
Targets
ABKeyManager
Copy Files
Headers
Bundle Resources
Sources
Frameworks & Libraries
GPGME.framework
Cocoa.framework
AddressBook.framework
Copy Files
Copy Files
Executables
Errors and Warnings
Find Results
Bookmarks
SCM
Project Symbols
Implementation Files
NIB Files


Groups & Files
ABKeyManager
Targets
ABKeyManager
Copy Files
Headers
Bundle Resources
Sources
Frameworks & Libraries
Cocoa.framework
AddressBook.framework
Copy Files
GPGME.framework
Copy Files
Executables
Errors and Warnings
Find Results
Bookmarks
SCM
Project Symbols
Implementation Files
NIB Files



All Settings	Value
Exported Symbols File	
Current Version	
Compatibility Version	
Symbol Ordering Flags	
~~Dead Code Stripping~~	☐
Don't Dead-Strip Inits and Terms	☐
Warning Linker Flags	
Other Linker Flags	-weak_framework GPGME
Product Name	ABKeyManager
Executable Prefix	
Executable Suffix	
Wrapper Extension	bundle
Private Headers Folder Path	
Public Headers Folder Path	
Force Package Info Generation	☐
Info.plist File	Info.plist
Preserve HFS Data	☐