Linking Native Libraries

Xamarin.iOS supports linking with both native C libraries and Objective-C libraries. This document discusses how to link your native C libraries with your Xamarin.iOS project. For information on doing the same for Objective-C libraries, see our Binding Objective-C Types document.

Building Universal Native Libraries (i386, ARMv6, and ARMv7)

It is often desirable to build your native libraries for each of the supported platforms for iOS development (i386 for the Simulator and ARMv6/ARMv7 for the devices themselves). If you've already got an Xcode project for your library, this is really trivial to do.

To build the i386 version of your native library, run the following command from a terminal:

/Developer/usr/bin/xcodebuild -project MyProject.xcodeproj -target MyLibrary -sdk iphonesimulator -configuration Release clean build

This will result in a native static library under MyProject.xcodeproj/build/Release-iphonesimulator/. Copy (or move) the library archive file (libMyLibrary.a) to someplace safe for later use, giving it a unique name (such as libMyLibrary-i386.a) so that it doesn't clash with the armv6 and armv7 versions of the same library that you will build next.

To build the ARMv6 version of your native library, run the following command:

/Developer/usr/bin/xcodebuild -project MyProject.xcodeproj -target MyLibrary -sdk iphoneos -arch armv6 -configuration Release clean build

This time the built native library will be located in MyProject.xcodeproj/build/Release-iphoneos/. Once again, copy (or move) this file to a safe location, renaming it to something like libMyLibrary-armv6.a so that it won't clash.

Now build the ARMv7 version of the library:

/Developer/usr/bin/xcodebuild -project MyProject.xcodeproj -target MyLibrary -sdk iphoneos -arch armv7 -configuration Release clean build

Copy (or move) the resulting library file to the same location you moved the other 2 versions of the library, renaming it to something like libMyLibrary-armv7.a.

To make a universal binary, you can use the lipo tool like so:

lipo -create -output libMyLibrary.a libMyLibrary-i386.a libMyLibrary-armv6.a libMyLibrary-armv7.a

This creates libMyLibrary.a which will be a universal (fat) library which will be suitable to use for all iOS development targets.

Linking Your Library

Any third-party library that you consume needs to be statically linked with your application. This is required because iOS does not support shared libraries for user applications.

If you wanted to statically link the library "libMyLibrary.a" that you got from the Internet or build with Xcode, you would need to do a few things:

- Bring the Library into your project
- Configure Xamarin.iOS to link the library
- Access the methods from the library.

To **bring the library into your project**, Select the project from the solution explorer and press Command (Apple)+Option+A. Navigate to the libMyLibrary.a and add it to the project. When prompted, tell Xamarin Studio or Visual Studio to copy it into the project. After adding it, find the libFoo.a in the project, right click on it, and set the Build Action to none.

To **Configure Xamarin.iOS To Link the Library**, on the project options for your final executable (not the library itself, but the final program) you need to add in "iOS Build"'s Extra argument (these are part of your project options) the "-gcc_flags" option followed by a quoted string that contains all the extra libraries that are required for your program, for example:

-gcc_flags "-L${ProjectDir} -lMylibrary -force_load ${ProjectDir}/libMyLibrary.a"

The above example will link libMyLibrary.a

You can use the -gcc_flags to specify any set of command line arguments to pass to the GCC compiler used to do the final link of your executable. For example, this command line, also references the CFNetwork framework:

-gcc_flags "-L${ProjectDir} -lMylibrary -lSystemLibrary -framework CFNetwork -force_load ${ProjectDir}/libMyLibrary.a"

If your native library contains C++ code you must also pass the -cxx flag in your "Extra Arguments" so that Xamarin.iOS knows to use the correct compiler. For C++ the previous options would look like:

-cxx -gcc_flags "-L${ProjectDir} -lMylibrary -lSystemLibrary -framework CFNetwork -force_load ${ProjectDir}/libMyLibrary.a"

Accessing C Methods from C#

There are two kinds of native libraries available on iOS:

- Shared libraries that are part of the operating system and
- Static libraries that you ship with your application.

To access methods defined in either one of those, you use Mono's P/Invoke functionality which is the same technology that you would use in .NET, which is roughly:

- Determine which C function you want to invoke
- Determine its signature
- Determine which library it lives in
- Write the appropriate P/Invoke declaration

When you use P/Invoke you need to specify the path of the library that you are linking with. When using iOS shared libraries, you can either hardcode the path, or you can use the convenience constants that we have defined in our Constants class , these constants should cover the iOS shared libraries.

For example, if you wanted to invoke the UIRectFrameUsingBlendMode method from Apple's UIKit library which has this signature in C:

```
void UIRectFrameUsingBlendMode (CGRect rect, CGBlendMode mode);
```

Your P/Invoke declaration would look like this:

```
[DllImport (Constants.UIKitLibrary,EntryPoint="UIRectFrameUsingBlendMode")]
public extern static void RectFrameUsingBlendMode (RectangleF rect, CGBlendMode blendMode);
```

The Constants.UIKitLibrary is merely a constant defined as "/System/Library/Frameworks/UIKit.framework/UIKit", the EntryPoint lets us specify optionally the external name (UIRectFramUsingBlendMode) while exposing a different name in C#, the shorter RectFrameUsingBlendMode.

Static Libraries

Since you can only use static libraries on iOS, there is no external shared library to link with, so the path parameter in the DllImport attribute needs to use the special name "__Internal" as opposed to the path name.

This forces DllImport to look up the symbol of the method that you are referencing in the current program, instead of trying to load it from a shared library.