# AppPL Library for iOS

Version 0.9.9

23rd May, 2011

## Introduction

The App Performance Library (AppPL) enables developers of mobile applications on iOS, and other mobile platforms, to collect performance data of their apps “in the wild.” AppPL collects performance data in apps and communicates these to a remote web service, which collects and aggregates the data, and can provide periodical reports, identify problems, and optionally send alerts. Typically, AppPL is used to track performance, and errors, of API calls, as in most apps these calls affect the actual performance the most.

Watchmouse provides a default web service endpoint for logging data, but the specification of the acceptor service is open and you can develop your own back end for the library if needed.

## Requirements

The AppPL library depends on the SystemConfiguration and CoreTelephony frameworks, but has no other external dependencies. It makes use of features present in iOS 3.x and therefore requires that your iOS Deployment Target is set to iOS 3.1.3 or above. The library is distributed as a “fat” binary for armv6, armv7 and x86. You can safely include it in your own project and without affecting you ability to test your code in the iPhone or iPad simulator.

## Installing AppPL in your Xcode Project

You can download a binary distribution of AppPL from http://AppPL.watchmouse.com/. Alternatively if you want to inspect the source code, or build the library directly from the source, you can checkout the latest version of the source code from out Mercuial repository hosted on BitBucket at http://bitbucket.org/watchmouse/appperflib/.

### Binary Distribution

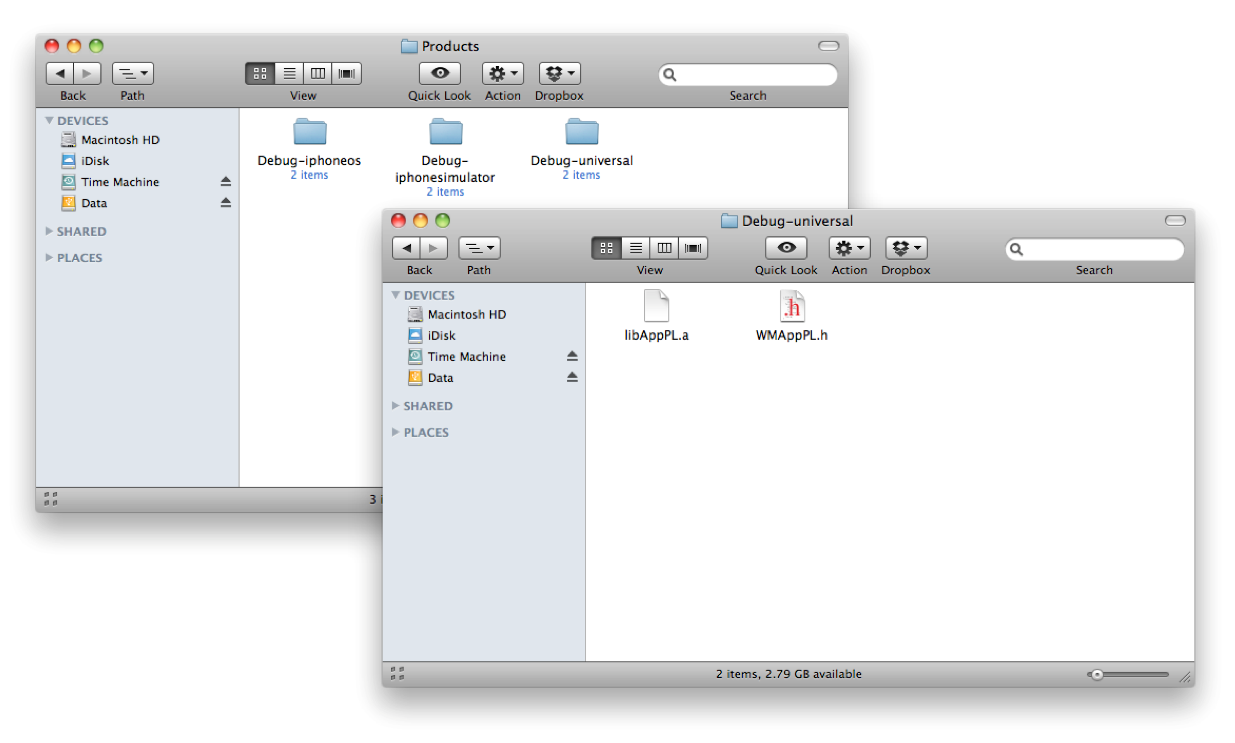
Drag and drop the libAppPL.a static library and WMAppPL.h header file into your Xcode project. At this stage you should also add the CoreTelephony and SusyemConfiguration frameworks to your project as the library depends on these frameworks. Finally you should add “-ObjC” and “-all\_load” to the “Other Linker Flags” in your Project Build Settings for both the project and target.

### Building from Source

### Macintosh HD:Users:aa:Desktop:Screen shot 2011-05-23 at 13.28.20.png

*Figure 1. The AppPL source code inside Xcode.*

Open the AppPL.xcodeproj file in Xcode and build the library. Xcode will automatically build both the arm and x86 versions of the library and combine them into a universal “fat” library during a custom build stage. In the build directory three separate sub-directories will be created, see Figure 2.



*Figure 2. The three built versions of the library.*

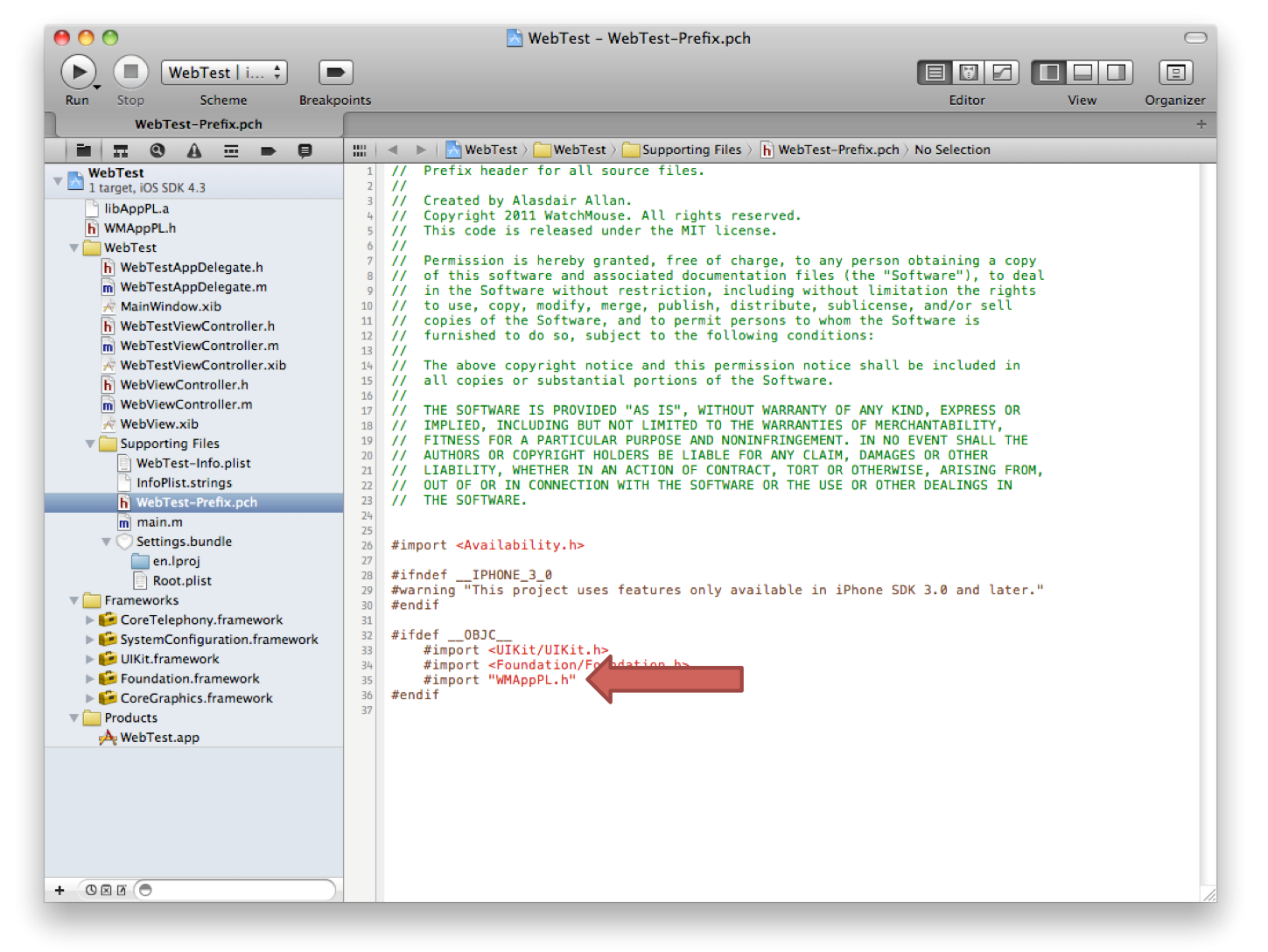
Go ahead and drag and drop the libAppPL.a static library and WMAppPL.h header file into your Xcode project, and then follow the directions outlined above for a binary distribution.

## Using AppPL

The AppPL library sub-classes the NSURLConnection and UIWebView classes with custom WMURLConnection and WMWebView it then intercepts all the delegate messages for these classes and logs information at each stage of the process of connecting to the remote service.

## Basic Usage

The easiest method to make sure AppPL is available throughout your project is to import the WMAppPL.h header file in your projects Prefix File, see Figure 3.



*Figure 3. Importing WMAppPL.h into the Prefix.pch file.*

The inside your Application Delegate initialize the library as below,

- (BOOL)application:(UIApplication \*)application

didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions {

WMAppPL \*performanceLibrary = [WMAppPL sharedWMAppPL];

performanceLibrary.libraryDebug = YES;

performanceLibrary.waitForWiFi = YES;

performanceLibrary.token = @"XXXXXXXXXXXX";

self.window.rootViewController = self.viewController;

[self.window makeKeyAndVisible];

return YES;

}

Here we obtain a reference the WMAppPL singleton object, turn on debugging for verbose output to the console (and stderr) during development, and ask the library to dispatch logging information to the remote web service only when the device is attached to a wireless rather than a wwan network connection. After this brief initialization step, the library is setup and ready for use.

To make use of the library you now need to replace all references to NSURLConnection, UIWebView and UIWebViewDelegate in your own code to with references to the library’s own WMURLConnection, WMWebView and WMWebViewDelegate classes. You should remember to change the class of any UIWebView created using Interface Builder to the WMWebView custom class using the identity inspector in the Utilities panel in Interface Builder.

## Persistent Queue

The AppPL library makes use of a FIFO queue to handle pending notifications to the remote web service. By default this queue will be lost if the application terminates, however the library can make this queue persistent. To do so you should modify the following lifecycle methods in your Application Delegate to archive the queue when it enters the background or terminates.

- (void)applicationDidEnterBackground:(UIApplication \*)application {

/\* Use this method to release shared resources \*/

[[WMAppPL sharedWMAppPL] archiveQueue];

}

- (void)applicationWillTerminate:(UIApplication \*)application {

/\* Called when the application is about to terminate. \*/

[[WMAppPL sharedWMAppPL] archiveQueue];

}

Making sure you also restore the queue when the application opens or enters the foreground.

- (BOOL)application:(UIApplication \*)application

didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions {

WMAppPL \*performanceLibrary = [WMAppPL sharedWMAppPL];

performanceLibrary.libraryDebug = YES;

performanceLibrary.waitForWiFi = YES;

performanceLibrary.token = @"XXXXXXXXXXXX";

[performanceLibrary restoreQueue];

self.window.rootViewController = self.viewController;

[self.window makeKeyAndVisible];

return YES;

}

- (void)applicationWillEnterForeground:(UIApplication \*)application {

[[WMAppPL sharedWMAppPL] restoreQueue];

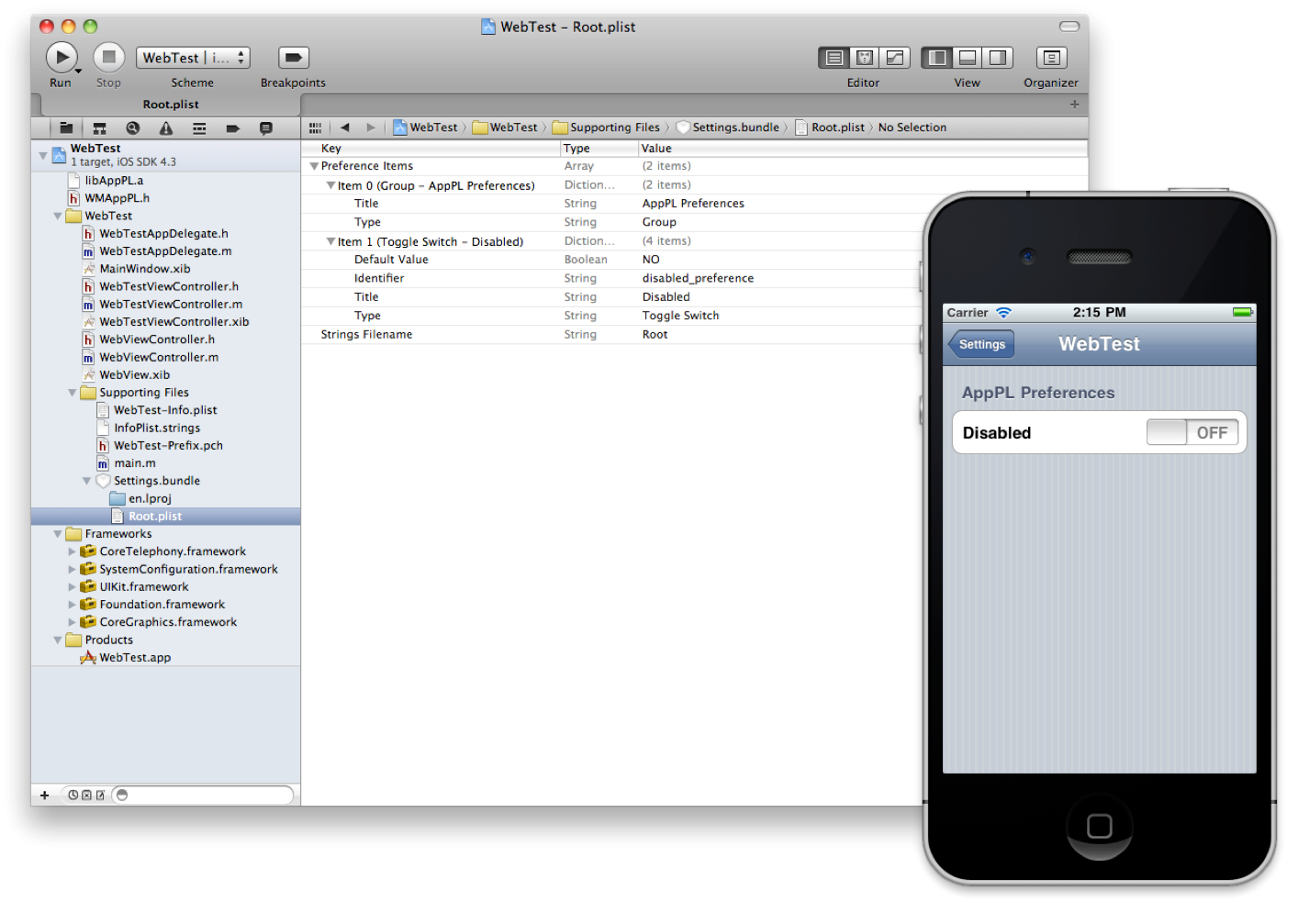
}

The libraries queue should now be persistent and, at least in theory, no web services calls will be lost due to network or other outages, such as problems with the remote acceptor web service.

## Preferences

You can expose the ability to turn the library on and off to your application’s users by adding a preference panel to your application and querying your application’s NSUserDefaults to check the state of this preference when the application is opened or enters the foreground.

In Xcode add a Settings.bundle file to your project, and modify it as in Figure 4.



*Figure 4. The Root.plist file in the Settings.bundle and in the iPhone’s Settings app*

Then in your Application Delegate add the following method that will be run when the application is initialized,

+ (void)initialize{

NSUserDefaults \*defaults = [NSUserDefaults standardUserDefaults];

NSDictionary \*appDefaults = [NSDictionary dictionaryWithObject:@"NO" forKey:@"disabled\_preference"];

[defaults registerDefaults:appDefaults];

[defaults synchronize];

}

and then modify the relevant lifecyle methods as below.

- (BOOL)application:(UIApplication \*)application

didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions {

WMAppPL \*performanceLibrary = [WMAppPL sharedWMAppPL];

performanceLibrary.libraryDebug = YES;

performanceLibrary.waitForWiFi = YES;

performanceLibrary.token = @"XXXXXXXXXXXX";

NSUserDefaults \*defaults = [NSUserDefaults standardUserDefaults];

BOOL disabled = [defaults boolForKey:@"disabled\_preference"];

performanceLibrary.libraryOff = disabled;

[performanceLibrary restoreQueue];

self.window.rootViewController = self.viewController;

[self.window makeKeyAndVisible];

return YES;

}

- (void)applicationWillEnterForeground:(UIApplication \*)application {

[[WMAppPL sharedWMAppPL] restoreQueue];

NSUserDefaults \*defaults = [NSUserDefaults standardUserDefaults];

[defaults synchronize];

[WMAppPL sharedWMAppPL].libraryOff = disabled;

}

## The Delegate Protocol

The AppPL library provides a delegate protocol that allows you to be notified each time the library dispatches a log message to the remote acceptor web service. You should declare the class you wish to receive these delegate messages as a WMAppPLDelegate as below,

@interface MyObject : NSObject <WMAppPLDelegate> {

// insert code here

}

and call the WMAppPL class method to set the library delegate in an appropriate place, e.g. for a view controller class this could be done in the viewDidLoad method.

[WMAppPL setDelegate:self];

Finally you can implement the two (optional) delegate methods,

- (void)flushCompletedWithResponse:(NSString \*)json;

- (void)flushFailedWithError:(NSError \*)error andResponse:(NSString \*)json;

which will pass the acceptor response as a JSON string to your own code.