Configuring a native iOS application with the MobileFirst Platform SDK

fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/7.0/hello-world/configuring-a-native-ios-with-the-mfp-sdk.html) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new)

Overview

To serve a native iOS application, MobileFirst Server must be aware of it. For this purpose, IBM MobileFirst Platform Foundation provides a Native API library, which contains a set of APIs and configuration files.

This tutorial explains how to generate the iOS Native API and how to integrate it with a native iOS application. You must generate the iOS Native API so that you can use it later on for tasks such as connecting to the MobileFirst Server instance, invoking adapter procedures, implementing authentication methods, and so on.

Prerequisite: Developers must be proficient with the Apple Xcode IDE.

This tutorial covers the following topics:

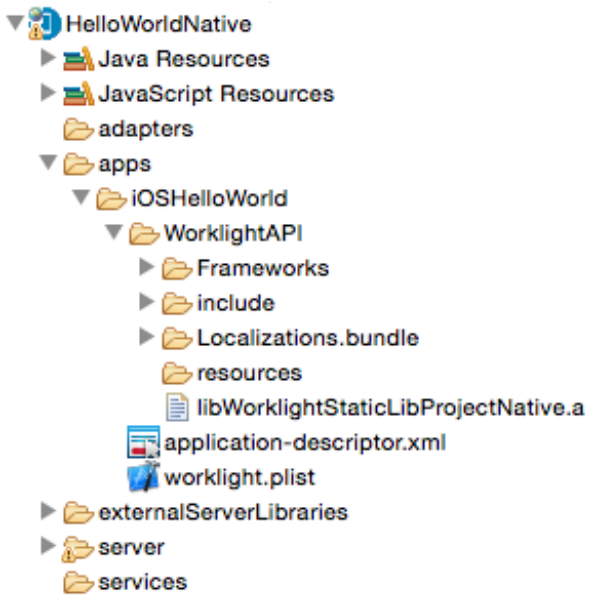
- Creating and Deploying a MobileFirst Native API
- Inside a MobileFirst Native API
- Configuring an iOS native application
- Configuring a Swift application
- Tutorials to follow next

Creating and Deploying a MobileFirst native API

CLI

1. In the terminal with the CLI (../../advanced-client-side-development/using-cli-create-build-manage-project-artifacts/) installed, create a new MobileFirst project using: `$ mfp create HelloWorldNative`.
2. Go to the newly created project directory: `$ cd HelloWorldNative/`.
3. Add a new iOS native API using `$ mfp add api iOSHelloWorld -e ios`.
4. Build and deploy the application using `$ mfp bd`. *This action is required for MobileFirst Server to recognize the application if it attempts to connect.*

Studio



Create

1. In MobileFirst Studio, create a MobileFirst project and add a MobileFirst Native API.
2. In the **New MobileFirst Native API** dialog, enter your application name and select **iOS** for the **Environment** field.

Deploy

3. Right-click the generated NativeAPI folder (located in `your-projects/apps/your-nativeapi-app-name`) and select **Run As > Deploy Native API**.
   This action is required for MobileFirst Server to recognize the application if it attempts to connect.

Inside a MobileFirst Native API

The MobileFirst native API contains several components:

- The `WorklightAPI` folder is an IBM MobileFirst API library that you must copy to your native iOS project.
- You use the `application-descriptor.xml` file to define application metadata and to configure security settings that MobileFirst Server enforces.
- The `worklight.plist` file contains connectivity settings that a native iOS application uses. You must copy this file to your native iOS project.
- As with any MobileFirst project, you create the server configuration by modifying the files that are in the `server\conf` folder.

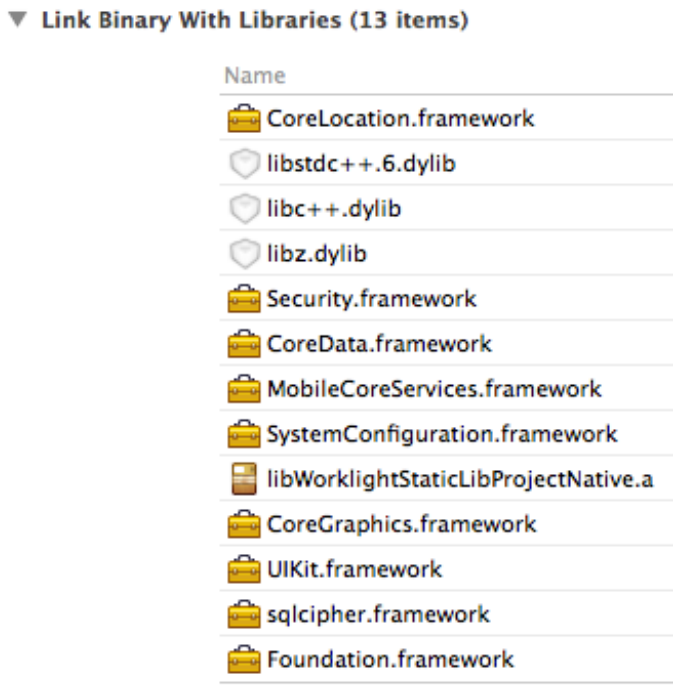
The worklight.plist file

The `worklight.plist` holds server configuration properties and is user-editable:

- `protocol` – The communication protocol to MobileFirst Server, which is either `http` or `https`.
- `host` – The hostname of the MobileFirst Server instance.
- `port` – The port of the MobileFirst Server instance.

- `wlServerContext` – The context root path of the application on the MobileFirst Server instance.
- `application id` – The application ID as defined in the `application-descriptor.xml` file.
- `application version` – The application version.
- `environment` – The target environment of the native application (“iOSnative”).
- `wlUid` – This property is used by MTW (Mobile Test Workbench, an Eclipse plug-in (../../advanced-topics/testing-mobilefirst-platform-applications-mobile-test-workbench)), the test workbench, to identify this as a MobileFirst application.
- `wlPlatformVersion` – The MobileFirst SDK version.

Configuring an iOS native application



1. Create an Xcode project or use an existing one.
2. Copy the previously-created `WorklightAPI` folder and the `worklight.plist` file from Eclipse to the root of the native project in Xcode.
3. In the Xcode Build Phases section, link the following libraries in your native iOS application:

   ### Required

   - libWorklightStaticLibProjectNative.a (found in **WorklightAPI**)
   - sqlcipher.framework (found in **WorklightAPI/Frameworks**)
   - MobileCoreServices.framework
   - SystemConfiguration.framework
   - CoreLocation.framework
   - Security.framework
   - libz.dylib
   - libc++.dylib
   - libstdc++.6.dylib

   ### Optional

- CoreData.framework

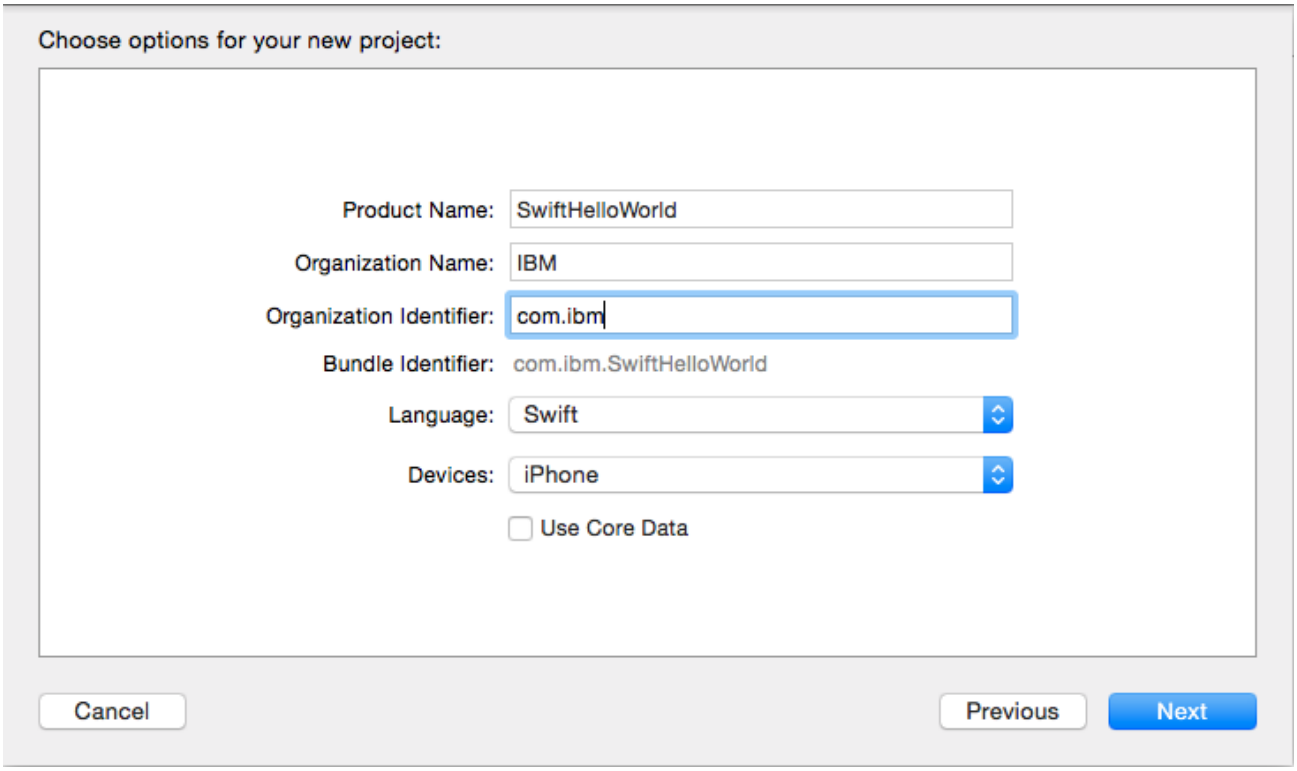
4. In Xcode **Build Settings**:
   - In Header Search Paths, add the following path: `$(SRCROOT)/WorklightAPI/include`
   - In **Other Linker Flags**, add `-ObjC`.
   - In the **Deployment** section, for the iOS **Deployment Target** field, select a value that is greater than, or equal to, 6.0.

> For more information review the "developing native applications for iOS" topic in the user documentation

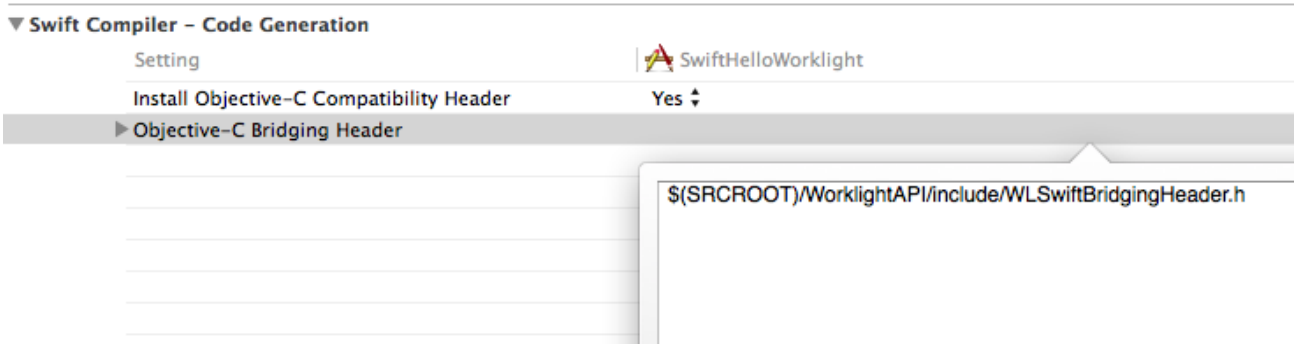
Configuring a Swift application

Because Apple Swift is designed to be compatible with Objective-C, you can use the MobileFirst SDK from within an iOS Swift project, too.

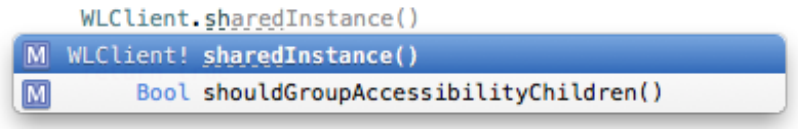
1. Create a Swift project and follow the same steps, as described at the beginning of the tutorial, to install the MobileFirst SDK into an iOS native application.



2. After all the normal steps for an iOS application, go to **Build Settings > Swift Compiler - Code Generation**.
   In **Objective-C Bridging Header**, add this path:
   `$(SRCROOT)/WorklightAPI/include/WLSwiftBridgingHeader.h`
   However, if you already have your own Bridging Header for other purposes, include the Worklight Bridging Header inside your own Bridging Header instead.



All the MobileFirst classes are now available from any of your Swift files.
XCode provides code autocompletion converted to the Swift style.



Tutorials to follow next

Now that your application contains the Native API library, you can follow the tutorials in the Native iOS development (../../ios-tutorials/) section to learn more about authentication and security, server-side development, advanced client-side development, notifications and more.