Configuring a native iOS application with the MobileFirst Platform SDK

Overview

Unlike MobileFirst Hybrid applications, which get autogenerated with the MobileFirst JavaScript SDK built-in, and can thus communicate with MobileFirst Server with no special customization, native apps are not bundled with the MobileFirst Native SDK by default.

In this tutorial, you learn how to add the MobileFirst Native SDK to a native iOS application and set it up with additional required configuration files.

With the MobileFirst Native SDK, you can implement various features, such as connecting to the MobileFirst Server instance, calling adapter procedures, implementing authentication methods, certificate pinning, using push notifications, and more.

For a request from a native app to be recognized by MobileFirst Server, the client application must be registered to the MobileFirst Server instance.

In this tutorial, you learn how to deploy the server-side entity for the native iOS application.

Prerequisites

- Proficiency with Apple Xcode IDE
- If you haven't gone through the Command Line Interface tutorial (../../advanced-client-side-development/using-cli-to-create-build-and-manage-mobilefirst-project-artifacts/), do so now.

Topics

- Adding the MobileFirst Native SDK to an application
- Deploying the server-side entity to MobileFirst Server
- Inside a native application containing the MobileFirst Native SDK
- Configuring a Swift application
- Tutorials to follow next

Adding the MobileFirst Native SDK to an application

The MobileFirst Native SDK can be added in several ways:

- From a remote CocoaPods repository: Ensure that you always use the latest available version.
- Locally: Use either the MobileFirst Studio plug-in for Eclipse or the MobileFirst CLI tool.

The preferred method is by using CocoaPods, as explained next. For the "Classic" local method, see the local method section.

1. Create a Xcode project or use an existing one.
2. If CocoaPods (http://guides.cocoapods.org) is not installed in your development environment, install it as follows:
    1. Open **Terminal**.
    2. Run the command: `sudo gem install cocoapods`
    3. Run the command: `pod setup` - **Note:** This command may take several minutes to complete.

3. Change directory to the location of the Xcode project.
4. Run the command: `pod init`. This creates a `Podfile`.
5. Using your favorite editor, open the `Podfile` file, located at the root of the project.
6. Comment out or remove the contents of the file.
7. Add the following lines and save the changes:

```
source 'https://github.com/CocoaPods/Specs.git'
pod 'IBMMobileFirstPlatformFoundation'
```

8. Run the command: `pod install`. This command adds the MobileFirst Native SDK, generates the Pod project, and integrates it with the Xcode project.
   Note: This command may take several minutes to complete.
9. > **Important**: From here on, use the `[ProjectName].xcworkspace` file in order to open the project in Xcode. Do **not** use the `[ProjectName].xcodeproj` file. A CocoaPods-based project is managed as a workspace containing the application (the executable) and the library (all project dependencies that are pulled by the CocoaPods manager).
10. Open the Xcode project by double-clicking the **.xcworkspace** file.
11. Right-click the project and select **Add Files To [ProjectName]**, select the `worklight.plist`, located in the root folder of the Xcode project.
12. Whenever you want to use the MobileFirst Native SDK, make sure that you import the framework:

```
#import <IBMMobileFirstPlatformFoundation/IBMMobileFirstPlatformFoundation.h>
```

> If you use Xcode 7 and iOS 9, read the ATS and Bitcode (file:////home/travis/build/MFPSamples/DevCenter/_site/blog/2015/09/09/ats-and-bitcode-in-ios9/) page.

Deploying the server-side entity to MobileFirst Server

Now that the native application is set up with the MobileFirst Native SDK, it is also required that MobileFirst Server recognize it after the SDK is used in the application. For this, a MobileFirst Server instance and a MobileFirst project are needed.

Creating a project and server instance

In **Terminal**, navigate to the location where you want to create the MobileFirst Backend project and run the following commands:

```
mfp create your-project-name-here
cd your-project-name
mfp start
```

After the project is created and the server is running, the server-side entity can be created and deployed to the server.
The next step adds the following files to the native project, as explained in section Inside a native application containing the MobileFirst Native SDK: `worklight.plist`, `application-descriptor.xml`, `mobilefirst` folder.

Deploying the server-side entity

1. In **Terminal**, navigate to the Xcode project.
2. Run the command: `mfp push`

   The command asks to which MobileFirst Server instance you want to deploy the artifacts and update the `wlclient.properties` file accordingly with the server address.

   An Interactive menu guides you, prompting you to provide the required values: application name, initial version number, and which bundle ID to use. You can edit these values again later in the `worklight.plist` and `application-descriptor.xml` files. You can also edit them by using the CLI command `mfp config`.

Inside a native application containing the MobileFirst Native SDK

In addition to the reconfigured project, now containing the MobileFirst Native SDK, three artifacts were added: the `worklight.plist` file, the application descriptor, and the `mobilefirst` folder.

worklight.plist

The `worklight.plist` file, located at the root of the project, 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 Mobile Test Workbench (deprecated feature) to identify it as a MobileFirst application.
- `wlPlatformVersion` – The MobileFirst Studio version.

Application descriptor

The `application-descriptor.xml` file, located at the root of the project, is a metadata file that you use to define various aspects of the application, such as user identity realms and push notifications support, security settings that MobileFirst Server enforces, and more.

The mobilefirst folder

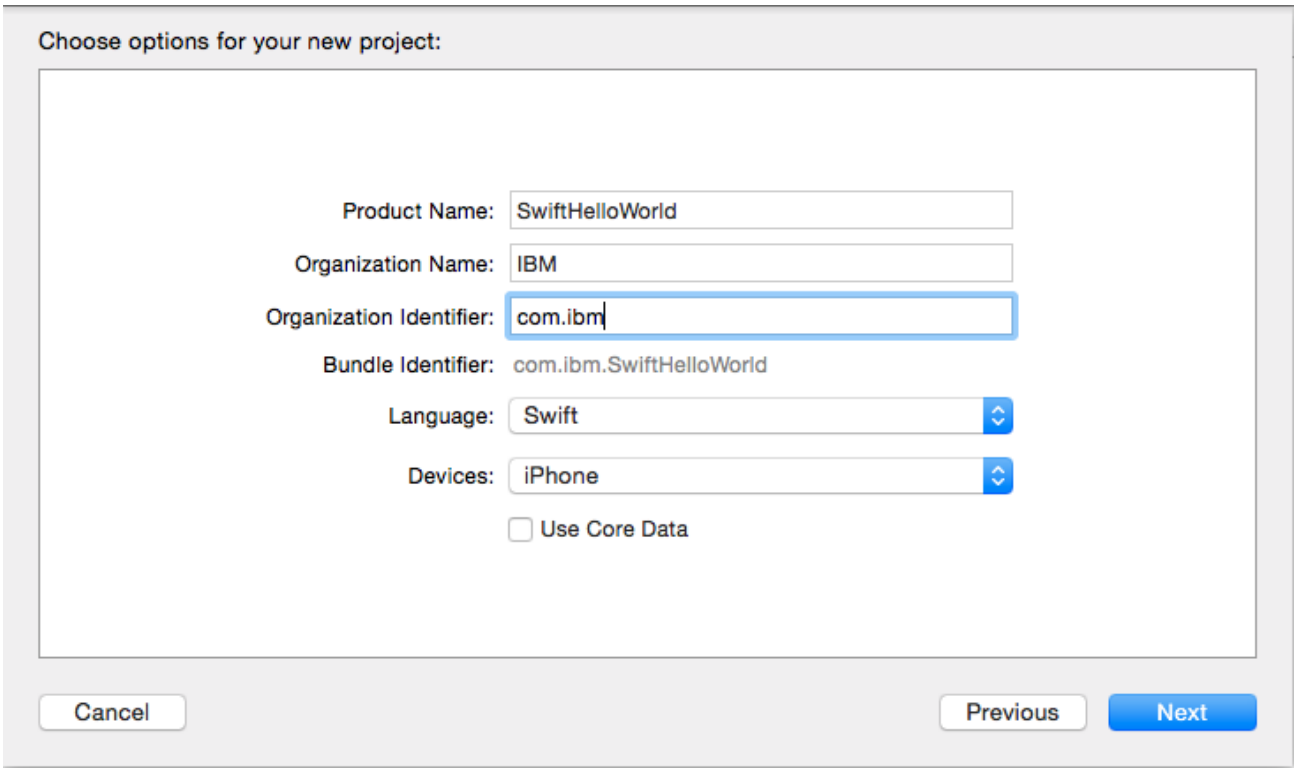
The `mobilefirst` folder, located at the root of the project contains `.wlapp` files. These files are the server-side entities that are deployed to the MobileFirst Server.

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.

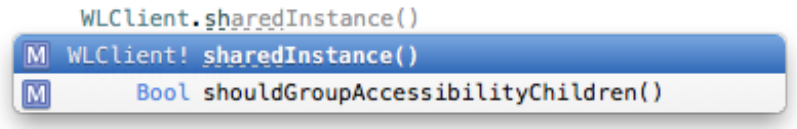
1. Create a Swift project and follow the same steps, as described at the beginning of the tutorial, to

install the MobileFirst SDK into an iOS native application.



2. Use `import "IBMMobileFirstPlatformFoundation/IBMMobileFirstPlatformFoundation.h"` in any class that needs to use the MobileFirst SDK.

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) category to learn more about authentication and security, server-side development, advanced client-side development, notifications, and more.

"Classic" local method to add the MobileFirst Native SDK to an application

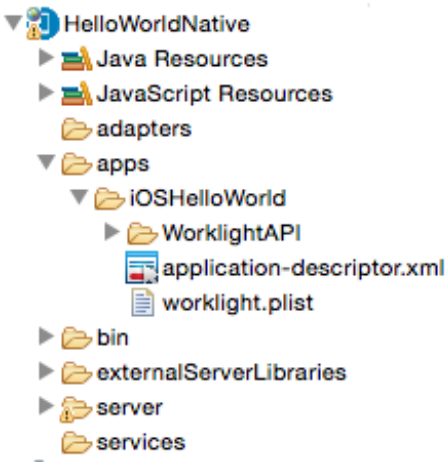
If you use the local method, SDK files are generated from the existing local resources in the installed version of MobileFirst Platform Foundation. In the "Classic" method, the SDK files are referred to as "NativeAPI". Follow the instructions below for either the MobileFirst Studio plug-in for Eclipse or the MobileFirst CLI.

Using either Studio or CLI, create and deploy the NativeAPI server-side entity:

1. ## Studio

    1. In MobileFirst Studio, create a MobileFirst project and add a MobileFirst Native API.

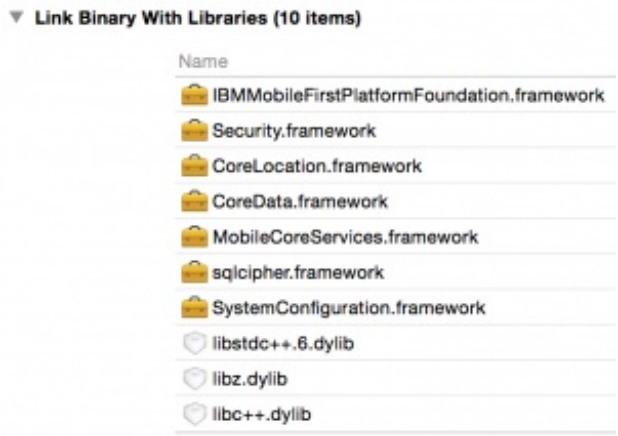
2. In the **New MobileFirst Native API** dialog, enter your application name and select **iOS** for the **Environment** field.
3. Right-click the generated `NativeAPI` folder (located in `your-projects/apps/your-nativeapi-app-name`) and select **Run As > Deploy Native API**.
   This action is required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



2. CLI

1. Using the CLI (../../advanced-client-side-development/using-cli-to-create-build-and-manage-mobilefirst-project-artifacts/), create a new MobileFirst project: `$ mfp create HelloWorldNative`
2. Navigate to the newly created project directory: `$ cd HelloWorldNative/`
3. Add a new iOS native API: `$ mfp add api iOSHelloWorld -e ios`
4. Navigate to the Xcode project and run the command: `$ mfp push`.
   Note: This action is required for MobileFirst Server to recognize the application if it attempts to connect.

3. Copy the files to the native iOS application:
   1. Copy the previously-created `WorklightAPI` folder and the `worklight.plist` file from Eclipse or the CLI-generated folder to the root of the native project in Xcode.



   2. In the Xcode Build Phases section, link the following libraries in your native iOS application:
      - SystemConfiguration.framework
      - MobileCoreServices.framework
      - CoreData.framework
      - CoreLocation.framework
      - Security.framework
      - sqlcipher.framework *(from WorklightAPI/Frameworks)*
      - IBMMobileFirstPlatformFoundation.framework *(from WorklightAPI/Frameworks)*
      - libstdc++.6.dylib
      - libz.dylib

- libc++.dylib

Note: If you use Xcode 7, instead of `libstdc++.6.dylib`, `libz.dylib`, and `libc++.dylib`, link to `libstdc++.6.tbd`, `libz.tbd`, and `libc++.tbd`.

3. In Xcode **Build Settings**:
   - In **Framework Search Paths**, add the following path: `$(SRCROOT)/WorklightAPI/Frameworks`
   - 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, see the topics about developing native applications for iOS, in the user documentation.