iPass SmartConnect Getting Started Developer Guide for iOS

FEBRUARY 2017 Version 1.6.0.41

Contents

Introduction

This document covers the initial implementation of the iPass Smart Connect SDK for iOS. iPass Smart Connect provides end-to-end connectivity framework for authenticating to iPass-supported WiFi networks across the globe.

IMPORTANT

This is a preliminary document for an API or technology in development. iPass is supplying this information to help you plan for the adoption of the programming interfaces described in this guide. This information is subject to change and software implemented according to this document should be tested with the latest version of the iPass SDK and accompanying documentation.

System Requirements

Operating System

- iOS 9.0 and onwards

Hardware Requirements

- Devices based on armv7, armv7s, arm64 architecture

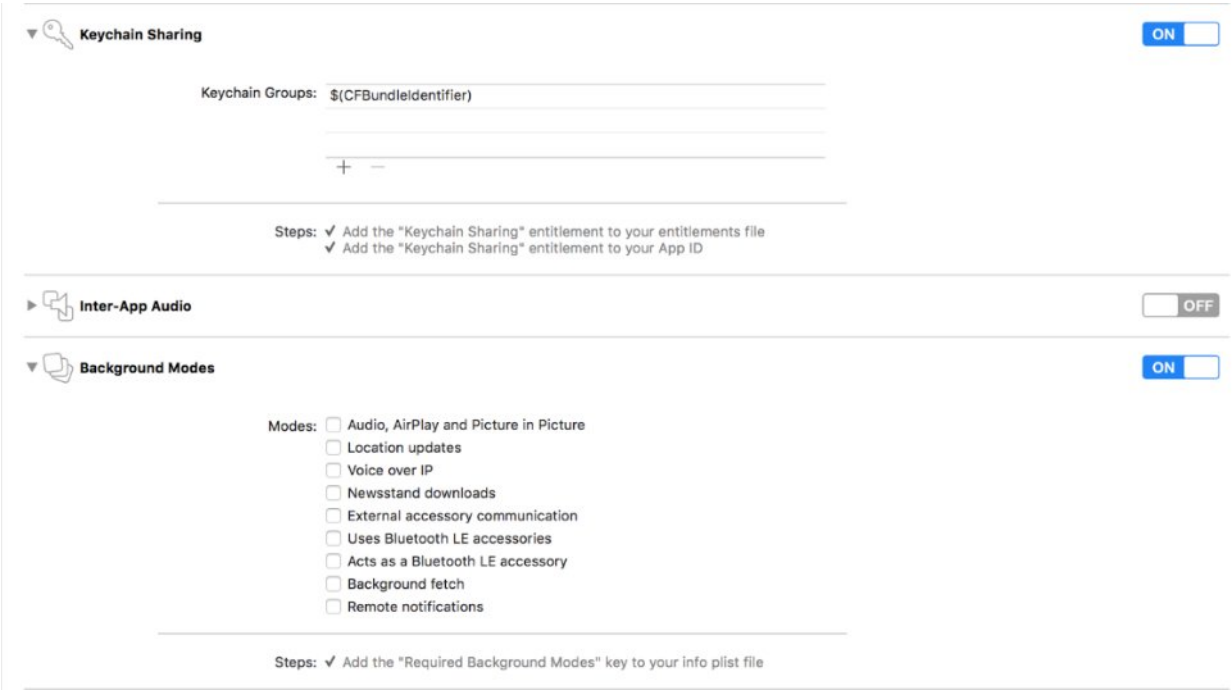
Creating Provisioning Profiles and Configuring SmartConnect enabled apps

Prerequisite

1. You must obtain the entitlements required for **Network Extension APIs** before using the SDK on a real device: send an email to networkextension@apple.com to obtain the entitlements to access WiFi APIs.
2. Select "**Network Extension**" as the entitlement when creating a provisioning profile on the Apple developer portal in your developer account:

This option is enabled by Apple after obtaining required entitlements. You will also need the NEHotspotHelper NEHotspotHelper API. To properly use NEHotspotHelper, the com.apple.developer.networking.HotspotHelper entitlement is required.

3. After obtaining the entitlements, create an entitlements file in XCode under the project using the SDK. Use the Capabilities pane under the project target to enable any of the services. This will create an entitlement file. To create the file manually:
   - From the file menu, select New File
   - Select iOS > Resource > Property List
   - Name the new file "foo.entitlements" (typically, "foo" is the target name)
   - Click the (+) next to "Entitlements File" to add a top-level item (the property list editor will use the correct schema due to the file extension)
   - Set your target's CODE_SIGN_ENTITLEMENTS build setting to be the path to entitlements file you just added.
   - Add these capabilities settings to the target:



4. Set the following value in the entitlements file:

   com.apple.developer.networking.HotspotHelper YES

Key	Type	Value
▼ Entitlements File	Dictionary	(4 items)
application-identifier	String	$(AppIdentifierPrefix)$(CFBundleIdentifier)
get-task-allow	Boolean	YES
com.apple.developer.networking.HotspotHelper	Boolean	YES
▼ Keychain Access Groups	Array	(1 item)
Item 0	String	$(AppIdentifierPrefix)$(CFBundleIdentifier)

5. Add the following keys to the target's info.plist file:
   - "**network-authentication**" as one of the "**Required Background Modes**"

- Add "**App Transport Security Settings"** key to the list and add **"Allow Arbitrary Loads"** key to it and set it to **"YES"**
- Add the following description keys to the file which is a mandatory requirement for the services SmartConnect uses:
    1. **Privacy - Location Usage Description**
    2. **Privacy - Location When In Use Usage Description**
    3. **Privacy – Motion Usage Description**

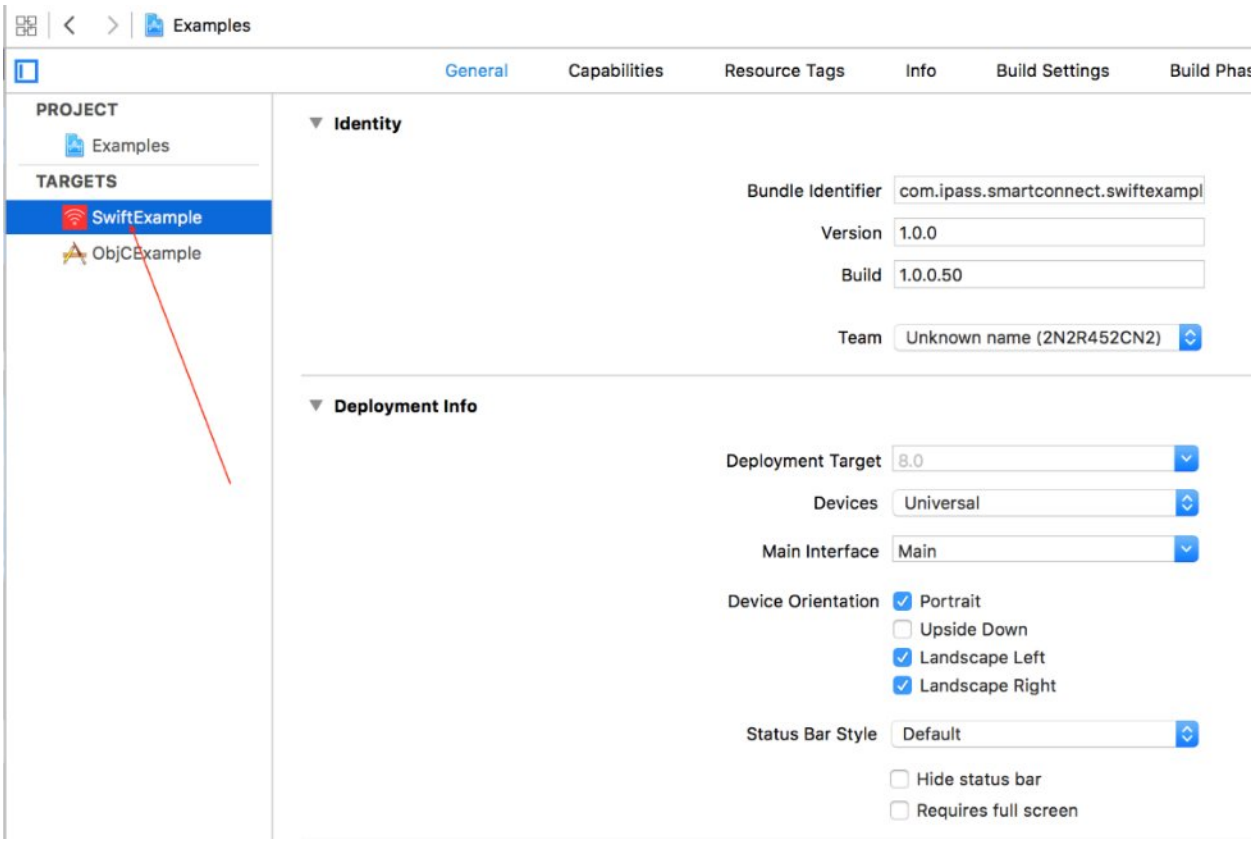
That's all that's needed to begin with. We'll now show you how to integrate the SDK framework in the application target and start coding.

Getting Started

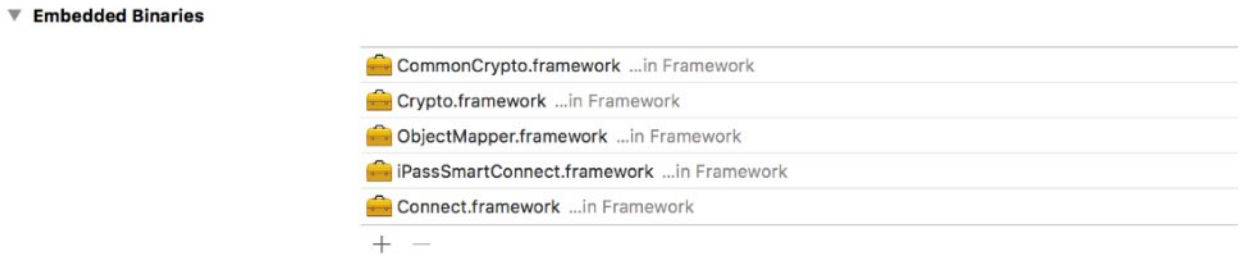
Note: Please note that the framework is a FAT framework, which means that it contains both device and simulator architecture. Apple does not currently allow uploading applications linked against FAT frameworks. The applications are suggested to remove simulator architecture slices before uploading the app to App Store. Follow **this link** for extra help on unwanted architectures in dynamic libraries.

Adding SmartConnect to a Swift App

1. In XCode, select an available template to create a new Swift-based project.
2. Copy the iPass SmartConnect framework to a directory of your choice.
3. From the target navigator pane, select the Swift target that you want the iPass SmartConnect framework to link to.



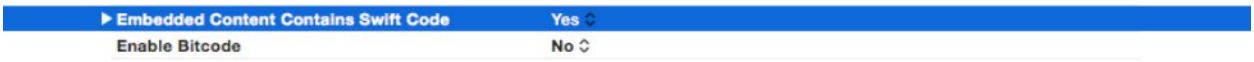
4. Go to the target **General** tab, then go to the Embedded Binaries section and select the "+" button to add the following frameworks to your project:



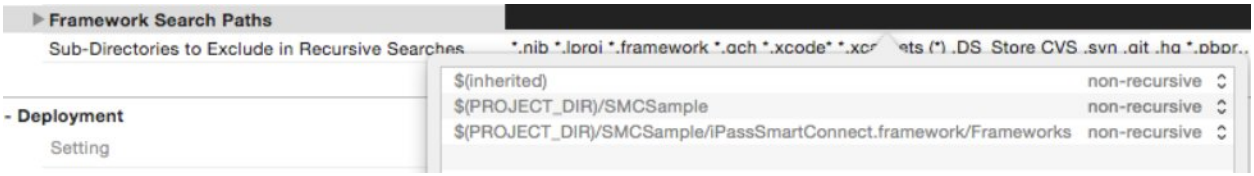
   Remove the frameworks that were automatically added in the Linked Frameworks and Libraries sections. Again, *not removing* these frameworks may cause problems while uploading your application in the App Store.
5. All frameworks--except **iPassSmartConnect**-can be found inside the ***Dependencies*** folder inside the iPassSmartConnect package.
6.

9.

10. Change these values in Build Settings:



11. Verify the framework search paths in Build Settings.



Once the project is set-up, import the iPassSmartConnect module to use iPassSmartConnect. We have attached an *Examples* project for greater details on how to get started with iPassSmartConnect, but to start with, add the following code sample to your application delegate:

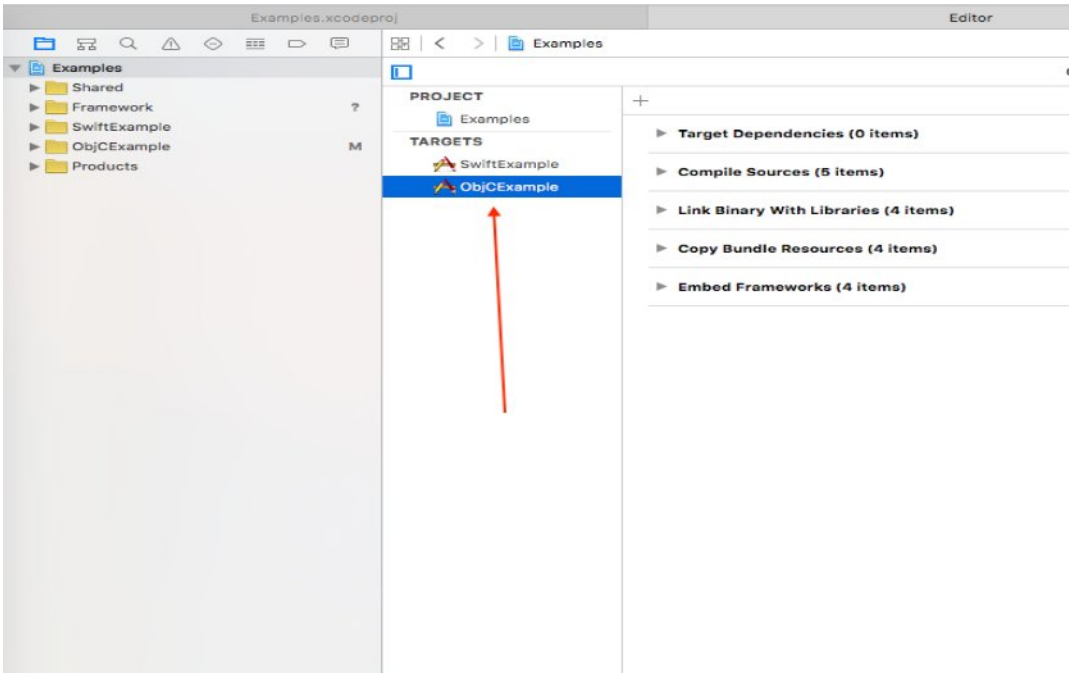
```
let connectionDelegate = MyConnectionDelegate()

try SMCCore.initialize(sdkKey, annotation: "Wifi Works !!", logLevel: .SMCDEBUG)
let characterizer = SMCCharacterizer()
let pluginHandler = SMCCaptivePluginHandler(characterizer: characterizer)
SMCCore.sharedInstance().connectionManager?.captivePlugin =
SMCCaptivePluginFactory.createPluginWith(pluginHandler)
SMCCore.sharedInstance().connectionManager?.delegate = connectionDelegate
```

12. MyConnectionDelegate class conforms to the ConnectionDelegate protocol and receives connection-related information.

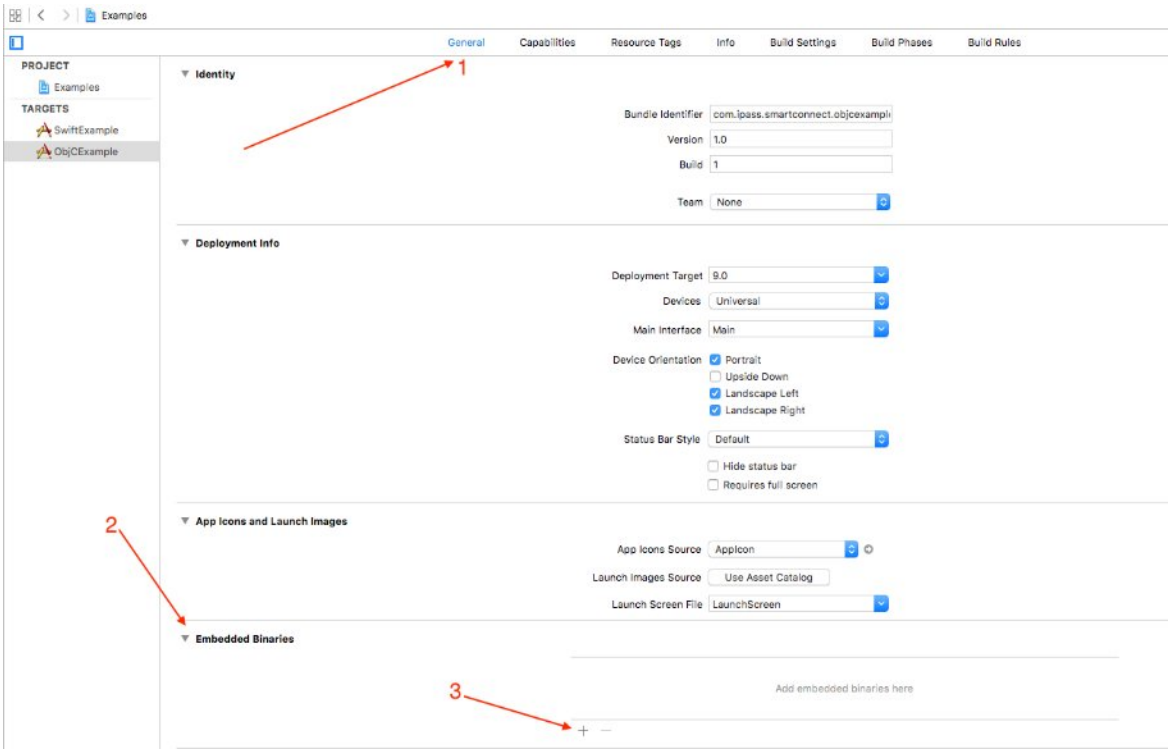
Adding iPass SmartConnect to an Objective C Application

1. In XCode, select an available template to create a new Objective-based project.
2. Copy the iPass SmartConnect framework to a directory of your choice.

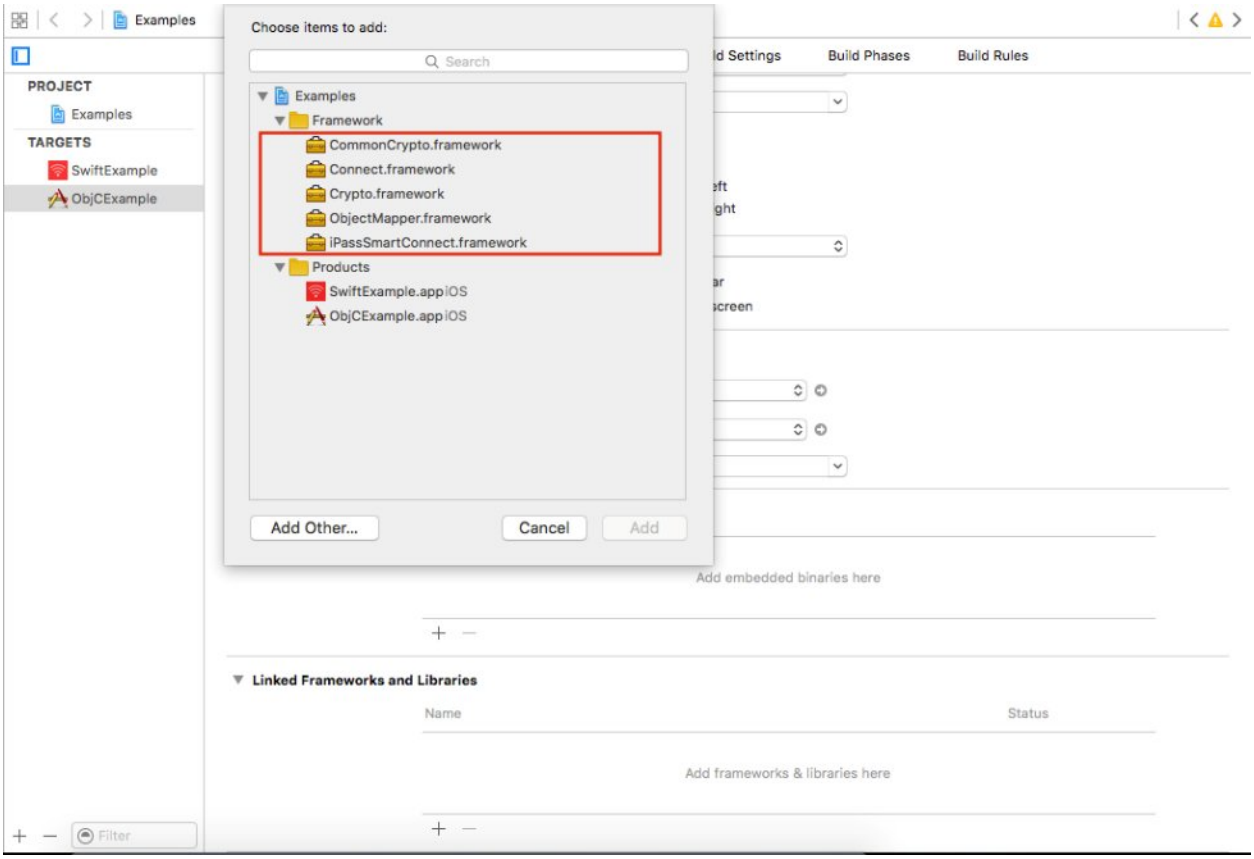
3. From the target navigator pane, select the Objective target that you want the iPass SmartConnect framework to link to:



4. Go to **General** tab of the target, then to the Embedded Binaries section. Select the '+ button to add the framework to your project.

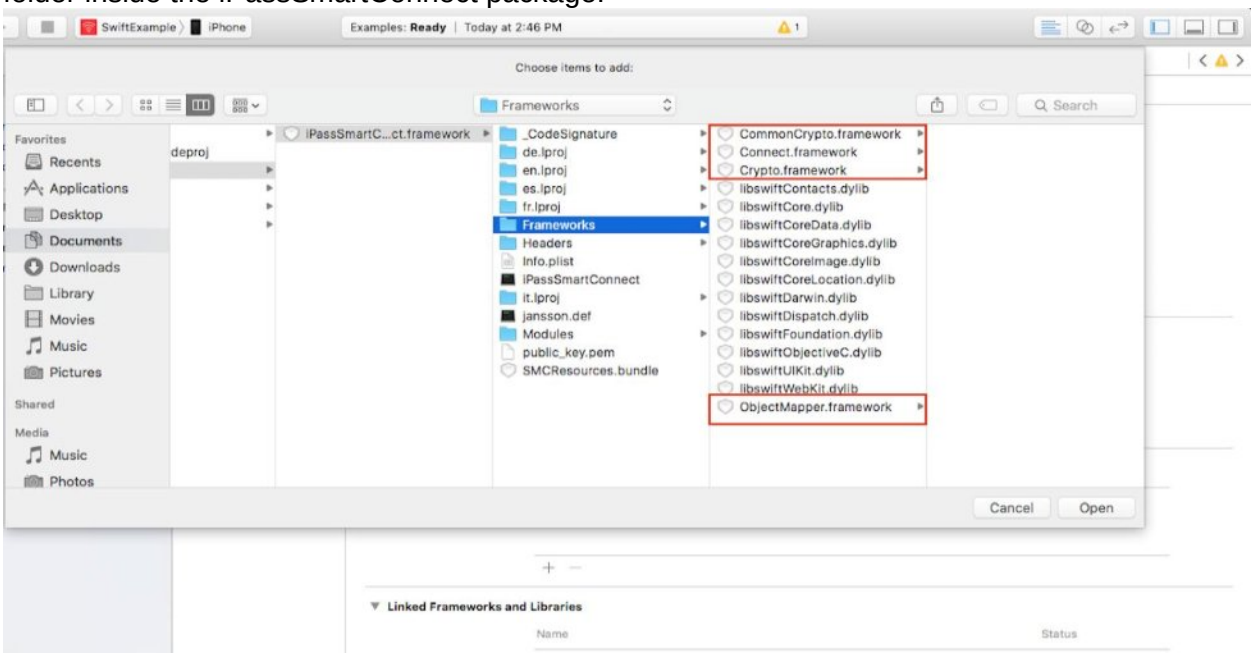


5. Add this framework to your project:



Remove the frameworks that were automatically added in the Linked Frameworks and Libraries section. Again, *not removing* these frameworks may cause problems while uploading your app to the App Store.

6. All frameworks--except **iPassSmartConnect**--can be found inside the ***Dependencies*** folder inside the iPassSmartConnect package.

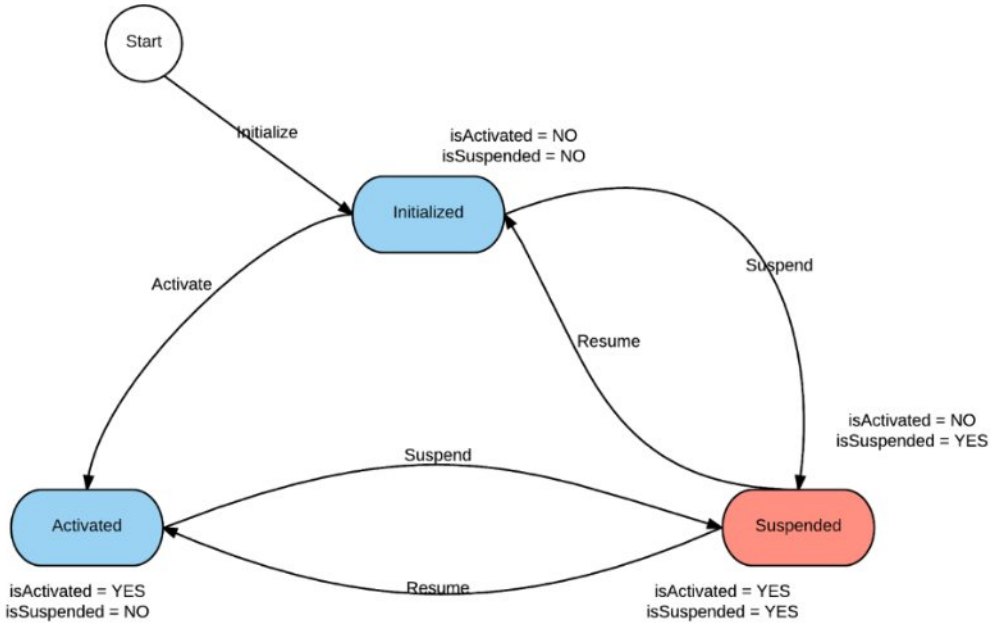


10. After this project is set-up, you must import the umbrella header file for iPassSmartConnect or use @import iPassSmartConnect to import the module.
11. We have attached an *Examples* project for greater detail on how to get started with iPassSmartConnect. To start, add the following code sample in your application delegate:

```
self.connectionDelegate = [MyConnectionDelegate new];

NSString* sdkKey = @"<SDK Key>"
NSError* error;
[SMCCore initialize:sdkKey annotation:@"Test ObjC Annotation" logLevel:SMCDEBUG error:&error];
id<Characterizer> characterizer = [[SMCCharacterizer alloc] init];
id<CaptivePluginHandler> pluginHandler = [[SMCCaptivePluginHandler alloc]
initWithCharacterizer:characterizer];
[SMCCore sharedInstance].connectionManager.captivePlugin = [SMCCaptivePluginFactory
createPluginWith:pluginHandler];
[SMCCore sharedInstance].connectionManager.delegate = self.connectionDelegate;
```

MyConnectionDelegate class conforms to the ConnectionDelegate protocol and receives connection-related information.

SDK State Architecture



Start

In this state, the SDK has been loaded but has not been initialized yet. No information--such as connection status or auto connect, for example--will be passed to the host app.

Initialized

1. The first thing the host app should do is initialize the SDK by making a call to SMCCore's initialize function.
2. The initialize function takes the SDK key, annotation string that the host app wants to show in the native WiFi settings, and the log level.
3. The initialize call is **synchronous call** and returns once the SDK is initialized. The SDK key should not be nil, as this will create an error.

Activated

Once the SDK is initialized, it's still not usable. Even though the connection status and network reachability changes would be notified to the host app, the connection still needs to be activated. The activation can be done using the following method and generally should be done only once through the lifecycle of the app.

Token Activation

The first method uses a token. The host makes an API call to its server to get the token, The host app's server makes a call to the iPass server to get the token. The iPass server returns the token to the host app's server, which then returns it back to host app. The host app then uses this token to activate the SDK.

Suspended

At any given point in time, if the host app wants to stop the iPass SDK from attempting connection to a network, stop using location and motions sensors so that the app can call the *suspend function.* Once in *suspended* state, the SDK will not perform any connection, although it will still notify the caller of the network changes using one of the connection delegates.

Resume

To come out of *suspended* state the host app can call the *resume* function on the *SMCCore.* Once resumed, the SDK goes back to activated state and starts to work normally.

UI Recommendations

Auto Login Enabled

Our SDK will automatically connect in the background: no UI is required.

Disabled

Although the host app should display some type of "login" or "connect" button and ask the user to connect, it's recommended to always have the *auto-login* setting turned on in the profile

Browser Login

The SDK sends a callback to the UI. The host app then shows the message, informs the SDK, and launches the browser.

CAPTCHA-based Networks

iPass SmartConnect SDK also supports authentication to CAPTCHA (in-flight) -based networks, such as Gogo inflight. Unlike authentication to non-CAPTCHA-based networks, which can happen in the background without user intervention, CAPTCHA-based networks require the

host app to be brought to the foreground so that the user can enter CAPTCHA. When a user selects an in-flight network, the SDK notifies the host app that it requires a UI in order to complete the authentication. The invoked callback is:

-(void)connectionRequiresUI:(SMCUIRequiredReason)reason completionHandler:(nullable UIRequiredCompletionHandler)completionHandler;

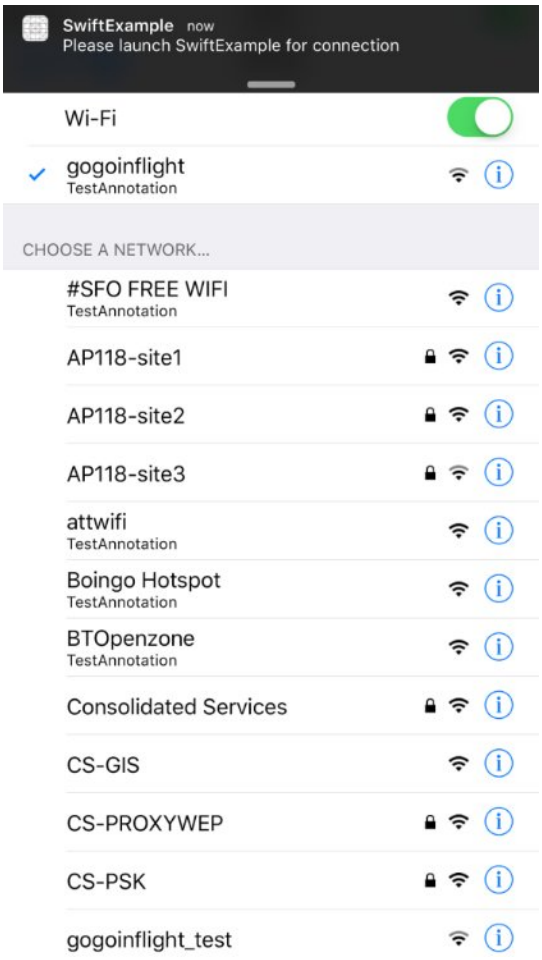
The required reason for this particular case is:
SMCUIRequiredReasonRequiresAppInForeground

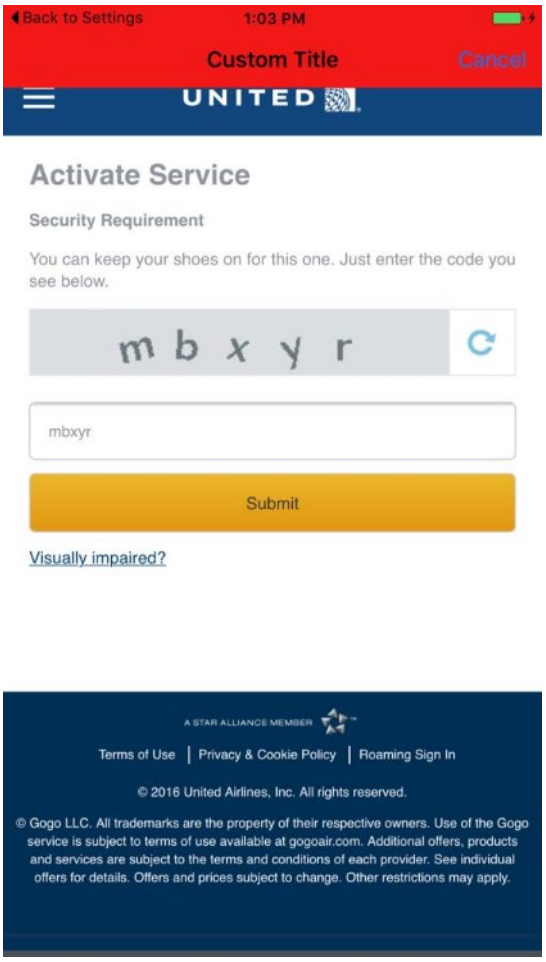
When the host app receives this callback, it should display a notification using UILocalNotification, asking you to bring the application to the foreground. Once the user brings the app to foreground, the SDK will automatically detect it and resume the authentication, eventually launching the CAPTCHA page. The iPass SDK also notifies the host app before launching the CAPTCHA, using the following callback:

-(void)willDisplayCaptcha:(SMCCaptchaViewController*)viewController;

```
@objc func willDisplayCaptcha(viewController: SMCCaptchaViewController) {
    viewController.title = "Custom Title"
    UINavigationBar.appearance().barTintColor = UIColor.redColor()
}
```

This gives the host app the ability to customize the UI (such as navigation barColor or tintColor.). Once the CAPTCHA is displayed and you enter the CAPTCHA, authentication is completed and the callback is notified of success or failure.

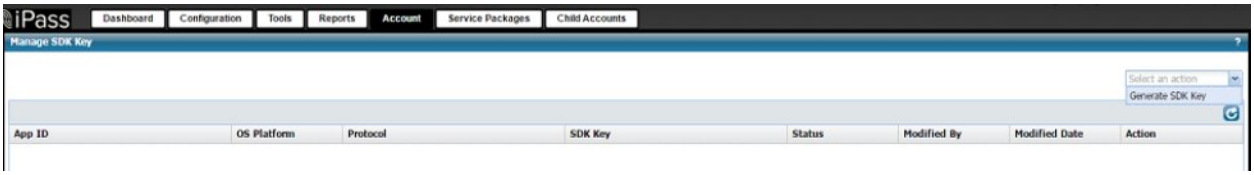




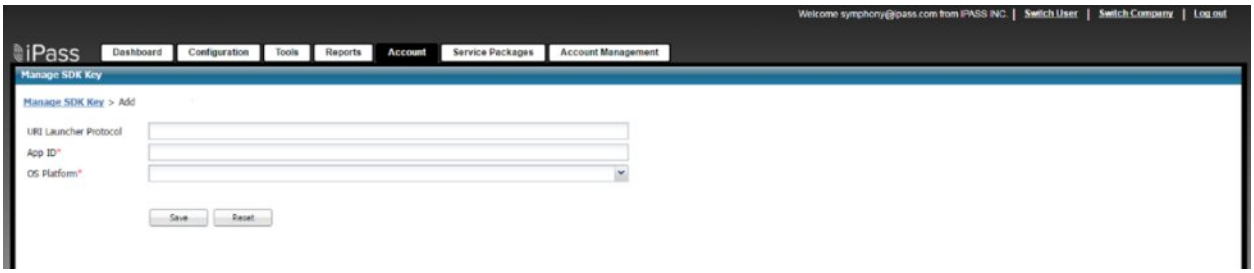
How to register Your App and Generate an SDK Key

To register your app and generate an SDK key:

1. Sign in to the iPass Portal.
2. From the **Accounts** tab, select **Manage SDK Key**.
3. Select the **Generate SDK Key** to register your app and create a new SDK Key.



Form to register a new app

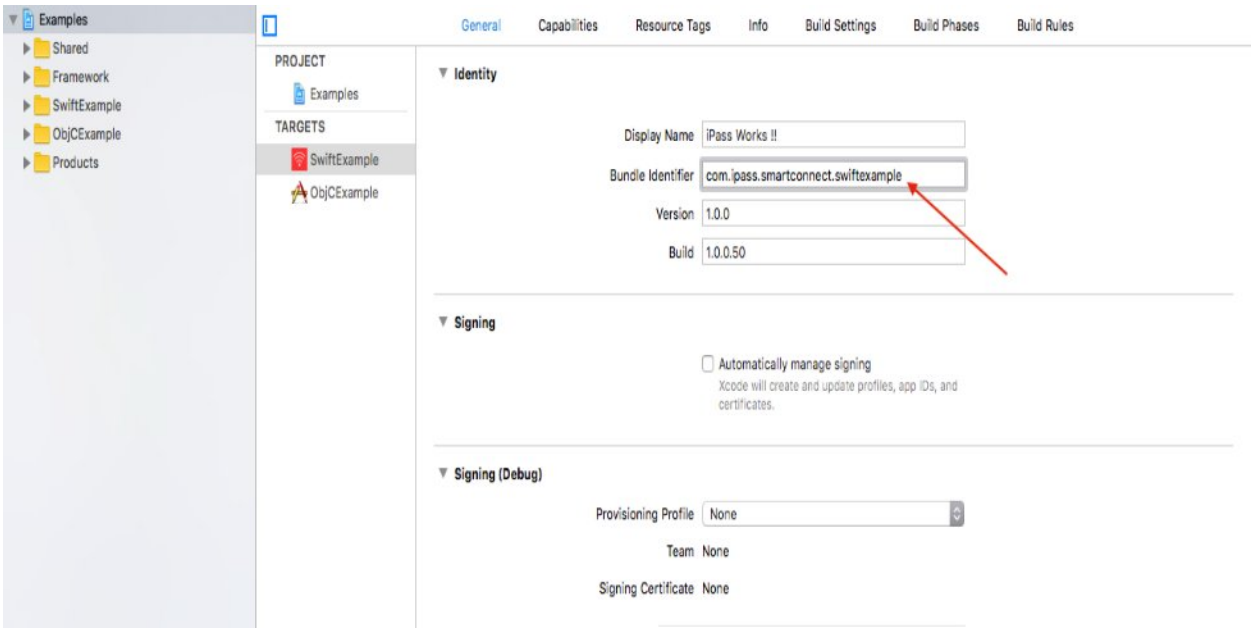


The **Generate SDK Key** page has the following fields:

- URI Launcher Protocol
- App ID
- OS Platform

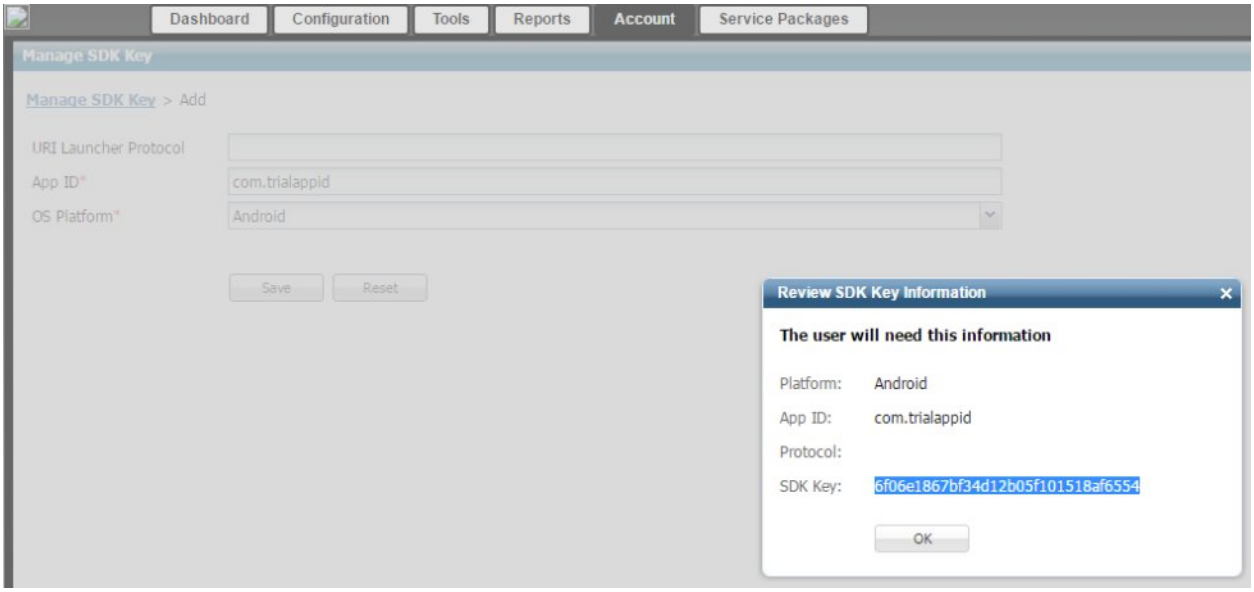
App ID and OS Platform are mandatory fields. The URI Launcher Protocol is not mandatory and can remain blank.

App ID should be the *'bundle identifier'* of the application as mentioned in the "Getting Started" section of this SDK, under application target . For example, the reference app's application id is *com.ipass.smartconnect.swiftexample*.

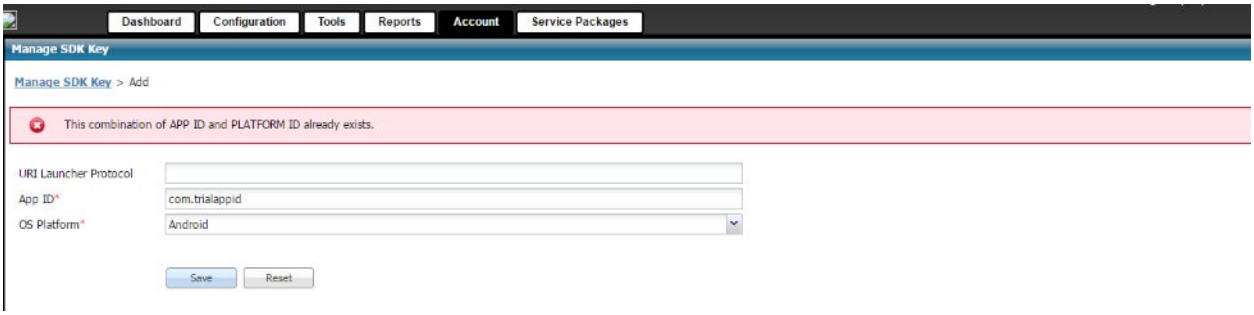


The SDK key is generated based on the combination of "App ID" and "Platform." This combination has to be unique and never used before..

Once the App is registered, Review SDK Key Information shows the generated SDK Key, along with app information.



If your “App ID” and “Platform” combination has been used previously, you’ll receive an error message:
“This combination of App ID and Platform ID already exists.”



After the SDK Key is generated, the App ID details can be viewed on the Manage SDK Key page. Copy the SDK key to use in the client app.

CONSTANTS

Note: All constants are Integers.

ACTIVATION STATUS CODE

VALUE	DESCRIPTION
100	**InvalidJobResource**: We received an invalid job resources. Possible causes are invalid token or credentials.
101	Token is not valid.
102	The profile that we got is not meant for this platform. Make sure that there's a default (favorite) profile for your company.
103	The profile id is not valid.
104	Failed to load profile.
105	The activation data for this request is not valid.
106	Invalid URL request.
107	Invalid Activation Response.

CONNECTION ERROR CODES

The SMCConnectionStatus object in the connection delegate callbacks contains an NSError object which is valid whenever an error occurs, for example, connectionFailed. The error code in the NSError object provides information about the kind of error that occurred. They error code will be one of the following values:

VALUE	DESCRIPTION
100	Login failed due to invalid credentials. This ideally should never happen. Contact your administrator.
101	Internal Gateway Error. There's some problem with the gateway. Please try again later.
102	Authentication request timed out on server.
103	Server has disabled authentication for this account. Please contact your administrator.
104	Unexpected response to the authentication request.
105	Logoff from the connection failed.
106	Authentication was aborted. This could be due to lost signal, user selecting different network, etc.
107	Http timed out.
108	Insufficient time for connection. Please try again.

109	Unknown Error. Cannot be classified.

APIs

The SDK bundle has a documentation folder, which contains detailed information about the APIs that the SDK provides to the host application for initialization, activation, and connection status.