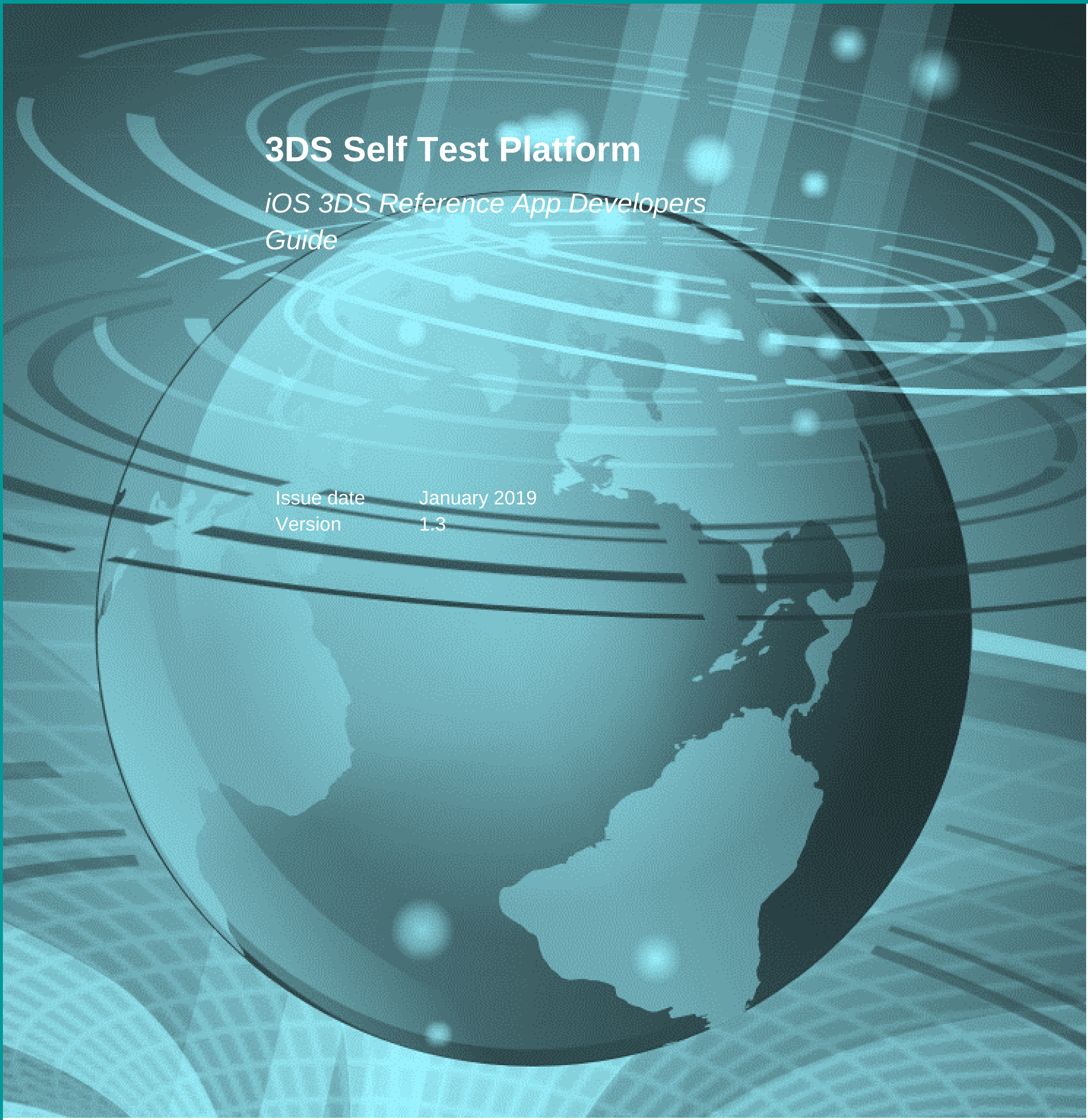


3DS Self Test Platform

iOS 3DS Reference App Developers Guide

Issue date	January 2019
Version	1.3

Document information

Author	UL
Date	January 2019
Status	Final

Table of Contents

1 Introduction

1.1 Purpose of this document

This document is the ‘iOS 3DS Reference App Developers Guide’. It serves as a guide for iOS SDK Developers and explains the steps needed to integrate their SDK with UL’s Reference App (available on UL 3DS Self Test Platform).

1.2 Intended audience

This document is intended for iOS 3DS SDK App Developers. Assumptions for using this document are:

- The user of this guide has good knowledge of iOS App development and of the EMV® 3-D Secure specifications and technology.
- You have access to the UL 3DS Self Test Platform.

2 Prerequisites

App Developers are expected to have the following:

- **Device running macOS**
  To get started with the iOS reference app you need to have a device running on macOS (For example a Mac Book, iMac or a Mac Mini) with Xcode installed. You can use macOS Sierra (10.12.x) and High Sierra (10.13.x).
- **Xcode**
  To ensure you can actually compile and run the app, you need to verify your Xcode version. Xcode version 9.0.x and Swift version 3.2 are the supported and recommend versions. The SDK and the Reference App have been developed using the language Swift version 3.2. The latest versions of Xcode will no longer support that version of Swift.
  Make sure you download Xcode version 9.0.x (available through the link 'Other downloads' on the Apple developers download page).
  Note: To download any version of Xcode you need to have Apple developers credentials. If you don't have these credentials you can obtain them by registering as an Apple developer. While you can run the app in a simulator it is strongly recommended to have a physical iOS device attached to your macOS device.
- **Apple developers account** (https://developer.apple.com/)
- **Cocoapods** downloaded from https://cocoapods.org.
- EMV® 3-D Secure SDK Specification (available on www.emvco.com)

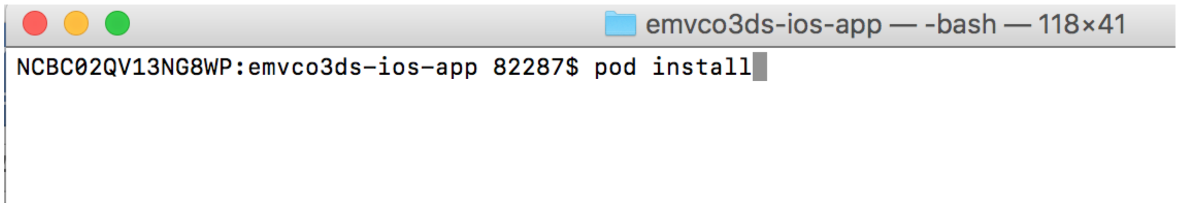
You have downloaded from UL 3DS Test Platform's Home page:

- UL Reference Application for SDK Developers – iOS (emvco3ds-ios-app.zip)
- UL Test Harness Specification for UL 3DS Test Platform

3 Open the example Project

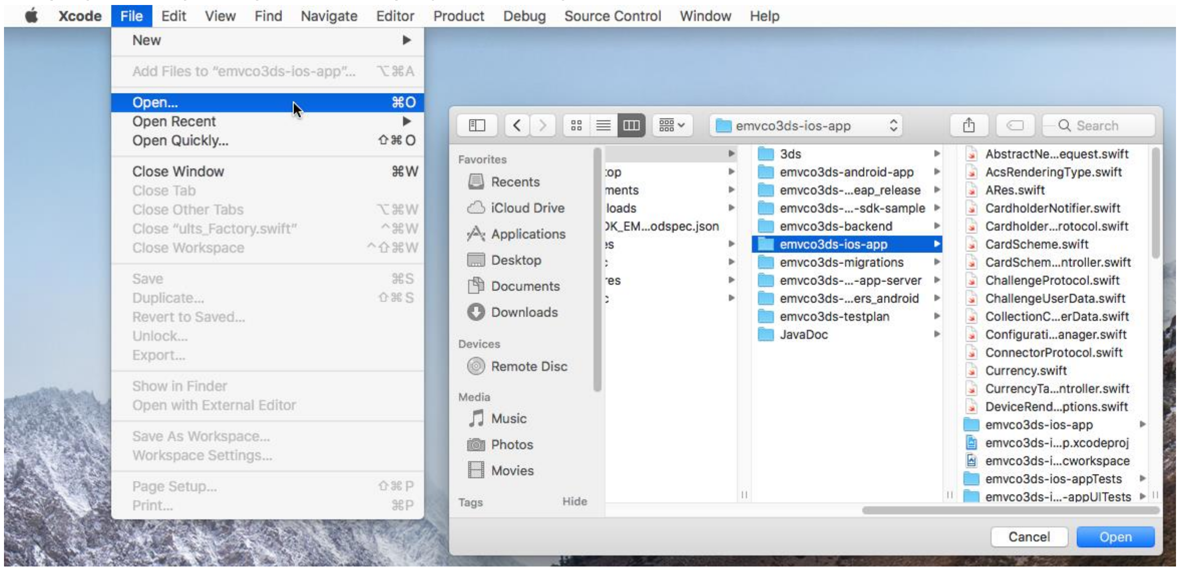
To open the example Project:

1. Unzip the file **emvco3ds-ios-app.zip** (downloaded from UL 3DS STP, version 6.25.2 or up) into a suitable location.
   The project is using **CocoaPods**. If CocoaPods has not been installed on your machine you need to download it first from https://cocoapods.org.
   Use the Terminal app to install it by typing: *sudo gem install cocoapods*.
   Note: A password for the administrator account of the macOS device will be required to execute this command.
2. The **emvco3ds-ios-pods** folder contains multiple ZIP files supporting multiple Swift versions (Version 3.2, 4.0, 4.1) just in case you need to use a Swift version other than version 3.2. Unzip the file that matches the version you need. Then copy the content of that file to the LocalPods (and/or LocalPods-Device) folder.
3. Open the **Terminal app** and type *pod install* to install all dependencies or *pod update* to update dependencies.



```
NCBC02QV13NG8WP:emvco3ds-ios-app 82287$ pod install
```

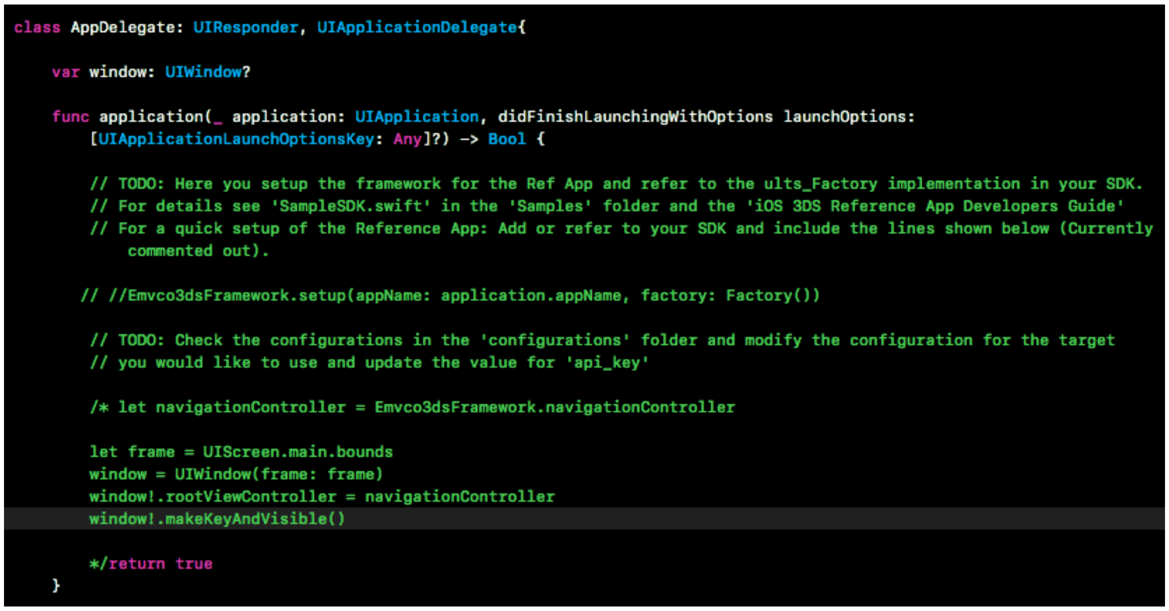
4. Within **Xcode**, choose **Open** from the **File** menu. Open the workspace file of *emvco3ds-ios-app*. Make sure you open the workspace file (having the *.xcworkspace* extension) rather than opening the project file (having the *.xcodeproj* extension).



5. If you have carefully followed these steps the project should be building successfully.
6. The workspace now contains three (3) projects:
   a. The app project (**emvco3ds-ios-app**),
   b. a framework (**emvco3ds-ios-framework**), and
   c. the **Pods**-project.

The app project combines the Reference App functionality (as defined in the framework project) with your new or with an existing SDK. Open the **AppDelegate.swift** file in the **emvco3ds-ios-app** project.

7. Follow the instructions found in the **didFinishLaunchingWithOptions** method in the **AppDelegate.swift** file. What you need to do here is to add your SDK implementation or to create a reference to your SDK and uncomment the code found after the TODO comment, as shown in the example below.

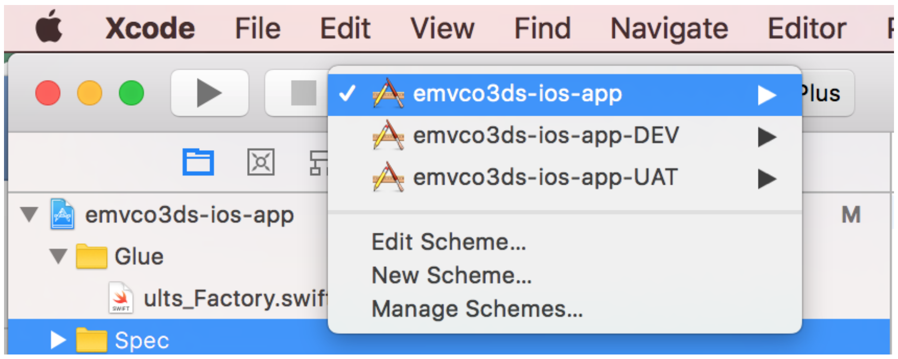


```
class AppDelegate: UIResponder, UIApplicationDelegate{

    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
        [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

        // TODO: Here you setup the framework for the Ref App and refer to the ults_Factory implementation in your SDK.
        // For details see 'SampleSDK.swift' in the 'Samples' folder and the 'iOS 3DS Reference App Developers Guide'
        // For a quick setup of the Reference App: Add or refer to your SDK and include the lines shown below (Currently
            commented out).

       // //Emvco3dsFramework.setup(appName: application.appName, factory: Factory())

        // TODO: Check the configurations in the 'configurations' folder and modify the configuration for the target
        // you would like to use and update the value for 'api_key'

        /* let navigationController = Emvco3dsFramework.navigationController

        let frame = UIScreen.main.bounds
        window = UIWindow(frame: frame)
        window!.rootViewController = navigationController
        window!.makeKeyAndVisible()

        */return true
    }
```

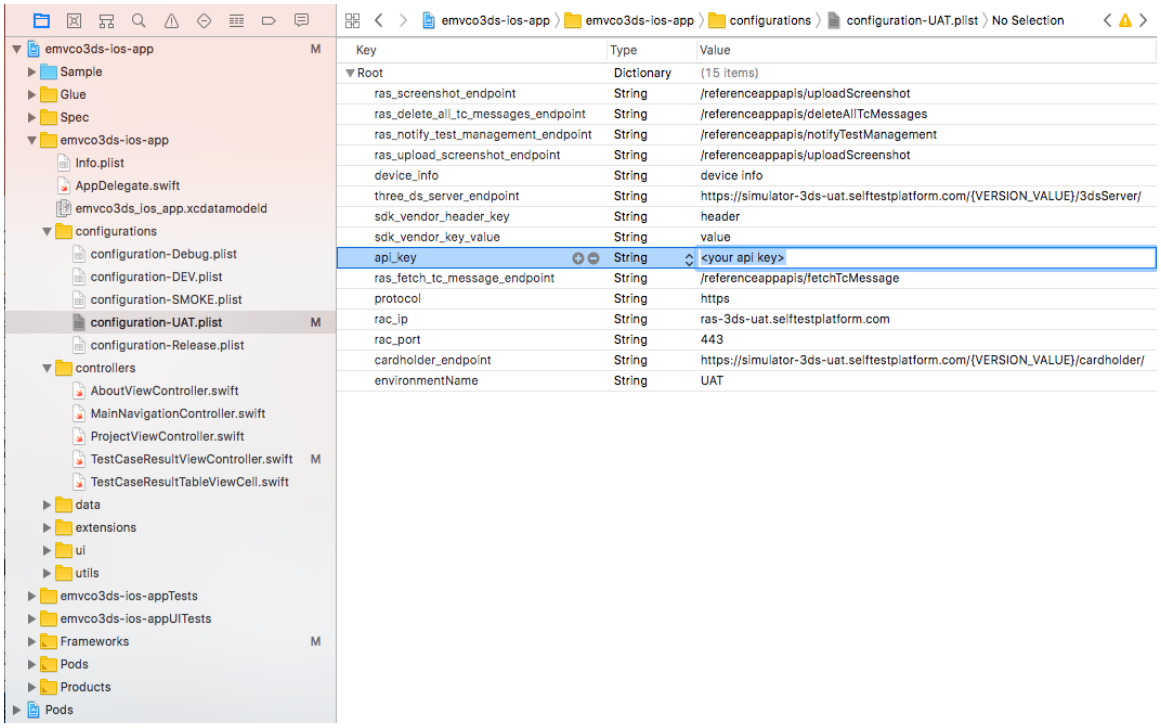
4 Configure your Project with a new SDK

To configure your Project with a new SDK:

1. Select the correct scheme for the desired environment against which you are testing.

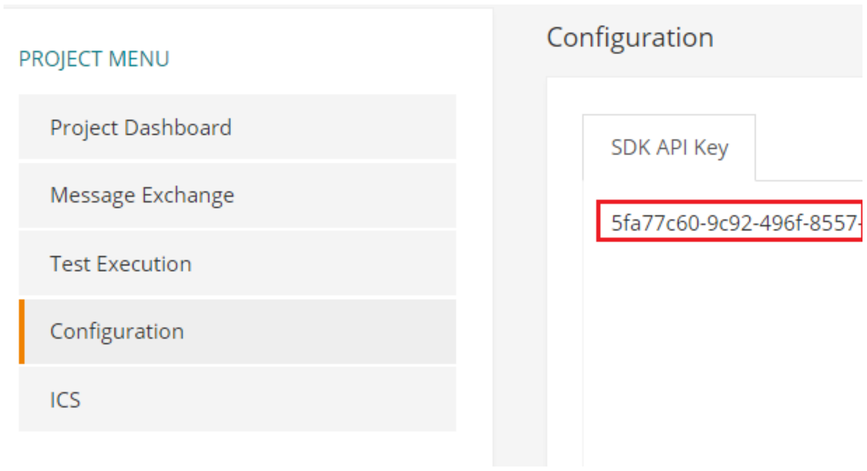


2. Find the *.plist* file for a given configuration and within the *.plist* file find the property *api_key*.


emvco3ds-ios-app 〉 emvco3ds-ios-app 〉 configurations 〉 configuration-UAT.plist 〉 No Selection
emvco3ds-ios-app
Sample
Glue
Spec
emvco3ds-ios-app
Info.plist
AppDelegate.swift
emvco3ds_ios_app.xcdatamodeld
configurations
configuration-Debug.plist
configuration-DEV.plist
configuration-SMOKE.plist
configuration-UAT.plist
configuration-Release.plist
controllers
AboutViewController.swift
MainNavigationController.swift
ProjectViewController.swift
TestCaseResultViewController.swift
TestCaseResultTableViewCell.swift
data
extensions
ui
utils
emvco3ds-ios-appTests
emvco3ds-ios-appUITests
Frameworks
Pods
Products
Pods

Key	Type	Value
▼Root	Dictionary	(15 items)
ras_screenshot_endpoint	String	/referenceappapis/uploadScreenshot
ras_delete_all_tc_messages_endpoint	String	/referenceappapis/deleteAllTcMessages
ras_notify_test_management_endpoint	String	/referenceappapis/notifyTestManagement
ras_upload_screenshot_endpoint	String	/referenceappapis/uploadScreenshot
device_info	String	device info
three_ds_server_endpoint	String	https://simulator-3ds-uat.selftestplatform.com/{VERSION_VALUE}/3dsServer/
sdk_vendor_header_key	String	header
sdk_vendor_key_value	String	value
api_key	String	<your api key>
ras_fetch_tc_message_endpoint	String	/referenceappapis/fetchTcMessage
protocol	String	https
rac_ip	String	ras-3ds-uat.selftestplatform.com
rac_port	String	443
cardholder_endpoint	String	https://simulator-3ds-uat.selftestplatform.com/{VERSION_VALUE}/cardholder/
environmentName	String	UAT

3. In the UL 3DS Self Test Platform at https://3ds.selftestplatform.com, open your project and choose the option **Configuration** from the **Project** menu. On the tab **SDK API Key** you can find the SDK API Key for your project.



5fa77c60-9c92-496f-8557-

4. Copy this value and paste it into the correct *.plist* file.

5 Build and Run

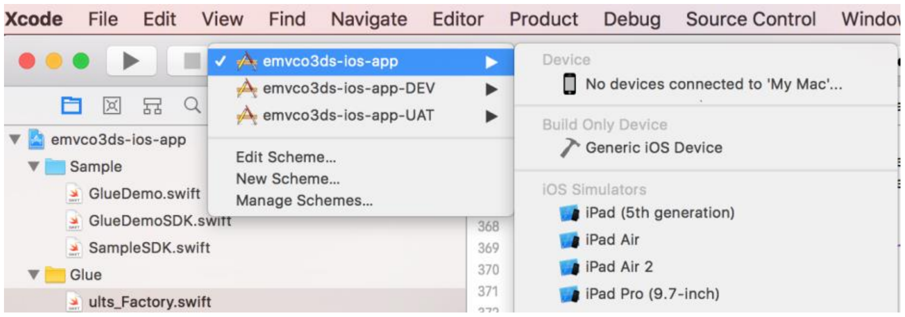
After you configured your Project, as explained in the previous chapter, the build process is executed as follows:

1. Attach your real device to your computer, such that it appears in the list of devices within Xcode.
2. In Xcode, select your device from the list appearing next to the selected scheme (e.g. emvco3ds-ios-app-UAT).
3. Select from the menu **Product > Build**.
   Make sure you have a valid provisioning profile for the iOS device that you attach to your computer. Please refer to the Apple documentation on the developers portal (https://developer.apple.com/).

Please note that to run the certification test cases you **MUST** use a real device.

To verify your implementation, you can use an iOS Simulator to test the Reference Application:

1. Within Xcode, select the appropriate iOS simulator from the list appearing next to the selected scheme (e.g. emvco3ds-ios-app-UAT).



2. Select from the menu **Product > Run**.

6 Reference App Integration Layer

The iOS/Swift implementation is following the EMV® 3-D Secure SDK Specification (Version 2.1.0, October 2017) as closely as possible. The implementation defines a set of Swift protocols (also known as interfaces), each corresponding to the Java/Android style definitions in the mentioned document.

The Reference App will use the SDK's functionality based on the abstract definitions mentioned above. You need to supply a full implementation for the set of protocols.

As a strategy you can choose between two possible implementation scenarios:

- implement a Reference App integration Layer in the SDK
- Create a 'glue layer' in the Reference App.

6.1 Reference App integration Layer in the SDK

UL provides an SDK sample with the classes needed to do the integration with the Reference App. You will need to implement those in your SDK to have it integrated with the UL 3DS Self Test Platform.

This is the preferred choice for most cases (the alternative is a 'glue layer' as explained in section 6.2). The actual implementation is a fairly straightforward process. If needed you can refer to the sample implementation. Comments indicate where you need to add your implementation specific code.

6.2 Using a Glue layer in the Reference App

UL provides a (non-functional) Demo SDK representing a typical existing SDK implementation (the one provided by UL is in Swift but it could have been in Objective-C as well).

UL also provided a 'glue' layer that would translate between the abstract protocol definitions and the actual objects in the Demo SDK. This 'glue' layer can also fix minor differences in the way the specification was interpreted.

The provided 'glue' layer source should be self-explanatory.

7 Implementation of UL Challenge Protocols

This section provides snippets of how the Challenge Protocols can be implemented in your SDK.

You can find the list of Challenge Protocols (also known as Listeners) and the definition in the document ‘UL Test Harness Specification for UL 3DS Test Platform’, section 5.1.2 *Challenge Listeners* and appendix 4 *SDK Challenge Listener example code* (You can download this document from the Home page of UL 3DS Self Test Platform).

Include the *Pods_emvco3ds_ios_app.framework* reference into your SDK project such that you can create implementations conform to the protocols, which are listed in the following sections. Some implementation examples can be found in the **SDKChallengeDataProtocol** class, appearing in the **Protocol** folder.

7.1 Generic Challenge Protocol

```
@objc public protocol GenericChallengeProtocol {
    func clickVerifyButton()
    func getChallengeType() -> String
    func clickCancelButton()
    func setChallengeProtocol(sdkChallengeProtocol: SDKChallengeProtocol)
    func expandTextsBeforeScreenshot()
}
```

7.2 Challenge Type

There are four (4) known challenger types:

- `TextChallenge`
- `SingleSelector`
- `MultiSelector`
- `OOB`
- `WebChallenge`

7.3 Text Challenge Protocol

```
@objc public protocol TextChallengeProtocol : GenericChallengeProtocol {
    public func typeTextChallengeValue(_ value: String)
}

//objective-c sample
@interface TextChallengeViewController : UIViewController<TextChallengeProtocol>
@end

@interface TextChallengeViewController() {
}
@property (nonatomic, retain) id<SDKChallengeProtocol> sdkChallengeProtocol;
@end
```

```
@implementation TextChallengeViewController
@synthesize sdkChallengeProtocol;

- (void) clickVerifyButton {
    // Implement a way to click the submit button by this callback
}

- (void) typeTextChallengeValue:(NSString * _Nonnull)value {
    // Implement a way to set the textview value with the String parameter of this
callback
}

- (NSString * _Nonnull)getChallengeType SWIFT_WARN_UNUSED_RESULT {
  NSString *challengeType = @"01";
 return challengeType;
}

- (void)clickCancelButton {
  // Implement a way to click the cancel button by this callback
}

- (void)expandTextsBeforeScreenshot {
  // Implement a way to expand the whyInfo/expandInfo section by this callback
}

- (void)setChallengeProtocolWithSdkChallengeProtocol:(id <SDKChallengeProtocol>
_Nonnull)challengeProtocol {
 sdkChallengeProtocol = challengeProtocol;
}

- (void) handleChallenge {
// when the Challenge UI is ready and displayed in the screen
// based on the received CRes from ACS, the SDK must call
// the handleChallenge() callback so the Reference App can
// input the challenge data and continue with the challenge
   [sdkChallengeProtocol handleChallenge];
}
@end
```

7.4 Single Select Challenge Protocol

```
@objc public protocol SingleSelectorChallengeProtocol : GenericChallengeProtocol {

      public func selectObject(_ index: Int)
}
```

```
//objective-c sample
@interface SingleSelectChallengeViewController :
UIViewController<SingleSelectorChallengeProtocol>
@end
```

```
@interface SingleSelectChallengeViewController() {
}
@property (nonatomic, retain) id<SDKChallengeProtocol> sdkChallengeProtocol;
@end

@implementation SingleSelectChallengeViewController
@synthesize sdkChallengeProtocol;

- (void) clickVerifyButton {
    // Implement a way to click the submit button by this callback
}

- (void) selectObject:(NSInteger)index {
    // Implement a way to select the object indicated by the index parameter of
this callback
}

- (NSString * _Nonnull)getChallengeType SWIFT_WARN_UNUSED_RESULT {
  NSString *challengeType = @"02";
 return challengeType;
}

- (void)clickCancelButton {
  // Implement a way to click the cancel button by this callback
}

- (void)expandTextsBeforeScreenshot {
  // Implement a way to expand the whyInfo/expandInfo section by this callback
}

- (void)setChallengeProtocolWithSdkChallengeProtocol:(id <SDKChallengeProtocol>
_Nonnull)challengeProtocol {
 sdkChallengeProtocol = challengeProtocol;
}

- (void) handleChallenge {
// when the Challenge UI is ready and displayed in the screen
// based on the received CRes from ACS, the SDK must call
// the handleChallenge() callback so the Reference App can
// input the challenge data and continue with the challenge
   [sdkChallengeProtocol handleChallenge];
}
@end
```

7.5 Multi Select Challenge Protocol

```
@objc public protocol MultiSelectChallengeProtocol : GenericChallengeProtocol {
    public func selectIndex(_ index: Int)
}
```

```
//objective-c sample
@interface MultiSelectChallengeViewController :
UIViewController<MultiSelectChallengeProtocol>
@end

@interface MultiSelectChallengeViewController() {
}
@property (nonatomic, retain) id<SDKChallengeProtocol> sdkChallengeProtocol;
@end

@implementation MultiSelectChallengeViewController
@synthesize sdkChallengeProtocol;

- (void) clickVerifyButton {
    // Implement a way to click the submit button by this callback
}

- (void)selectIndex:(NSInteger)index {
    // Implement a way to select the object indicated by the index parameter of
this callback
}

- (NSString * _Nonnull)getChallengeType SWIFT_WARN_UNUSED_RESULT {
  NSString *challengeType = @"03";
 return challengeType;
}

- (void)clickCancelButton {
  // Implement a way to click the cancel button by this callback
}

- (void)expandTextsBeforeScreenshot {
  // Implement a way to expand the whyInfo/expandInfo section by this callback
}

- (void)setChallengeProtocolWithSdkChallengeProtocol:(id <SDKChallengeProtocol>
_Nonnull)challengeProtocol {
 sdkChallengeProtocol = challengeProtocol;
}

- (void) handleChallenge {
// when the Challenge UI is ready and displayed in the screen
// based on the received CRes from ACS, the SDK must call
// the handleChallenge() callback so the Reference App can
// input the challenge data and continue with the challenge
   [sdkChallengeProtocol handleChallenge];
}
@end
```

7.6 OOB Challenge Protocol

```
@objc public protocol OutOfBandChallengeProtocol : GenericChallengeProtocol {}

//objective-c sample
@interface OutOfBandChallengeViewController :
UIViewController<OutOfBandChallengeProtocol>
@end

@interface OutOfBandChallengeViewController() {
}
@property (nonatomic, retain) id<SDKChallengeProtocol> sdkChallengeProtocol;
@end

@implementation OutOfBandChallengeViewController
@synthesize sdkChallengeProtocol;

- (void) clickVerifyButton {
    // Implement a way to click the continue button by this callback
}

- (NSString * _Nonnull)getChallengeType SWIFT_WARN_UNUSED_RESULT {
  NSString *challengeType = @"04";
 return challengeType;
}

- (void)clickCancelButton {
  // Implement a way to click the cancel button by this callback
}

- (void)expandTextsBeforeScreenshot {
  // Implement a way to expand the whyInfo/expandInfo section by this callback
}

- (void)setChallengeProtocolWithSdkChallengeProtocol:(id <SDKChallengeProtocol>
_Nonnull)challengeProtocol {
 sdkChallengeProtocol = challengeProtocol;
}

- (void) handleChallenge {
// when the Challenge UI is ready and displayed in the screen
// based on the received CRes from ACS, the SDK must call
// the handleChallenge() callback so the Reference App can
// input the challenge data and continue with the challenge
   [sdkChallengeProtocol handleChallenge];
}
@end
```

7.7 Web Challenge Protocol

```
@objc public protocol WebChallengeProtocol : GenericChallengeProtocol {
    public func getWebView() -> UIWebView
}


//objective-c sample
@interface WebChallengeViewController : UIViewController<WebChallengeProtocol>
@end

@interface WebChallengeViewController() {
}
@property (nonatomic, retain) id<SDKChallengeProtocol> sdkChallengeProtocol;
@end

@implementation WebChallengeViewController
@synthesize sdkChallengeProtocol;

- (UIWebView * _Nonnull)getWebView SWIFT_WARN_UNUSED_RESULT {
    // Implement a way to return the UIWebView used for this challenge
}

- (void) clickVerifyButton {
    // do nothing
}

- (NSString * _Nonnull)getChallengeType SWIFT_WARN_UNUSED_RESULT {
  NSString *challengeType = @"05";
 return challengeType;
}

- (void)clickCancelButton {
  // Implement a way to click the cancel button by this callback
}

- (void)expandTextsBeforeScreenshot {
  // do nothing
}

- (void)setChallengeProtocolWithSdkChallengeProtocol:(id <SDKChallengeProtocol>
_Nonnull)challengeProtocol {
 sdkChallengeProtocol = challengeProtocol;
}

- (void) handleChallenge {
// when the Challenge UI is ready and displayed in the screen
// based on the received CRes from ACS, the SDK must call
// the handleChallenge() callback so the Reference App can
// input the challenge data and continue with the challenge
   [sdkChallengeProtocol handleChallenge];
}
@end
```