Cisco Webex iOS SDK Transition Guide

> The Cisco Webex™ iOS SDK Version 3.0.0.2-beta

The Cisco Webex iOS SDK Version 3.0.0.2-beta is a major rewrite of the version 2.x. It introduces a completely new architecture based on the same technology that powers the native Webex apps. This includes a lot of enhancements with the performance of the SDK most notably with the introduction of a local datawarehouse to cache all the spaces, messages, people, calls and much more. It also introduces a revolutionary new Catch up API that avoids redundant fetching of information from servers. Future updates and feature parity with the native clients will also be quicker than what it is at present. The v3 SDK is built in **Swift 5** and requires **iOS 11** or later. If you have developed an app with Cisco Webex SDK version 2.X, this guide will help you in transitioning from Webex SDK version 2 to version 3.

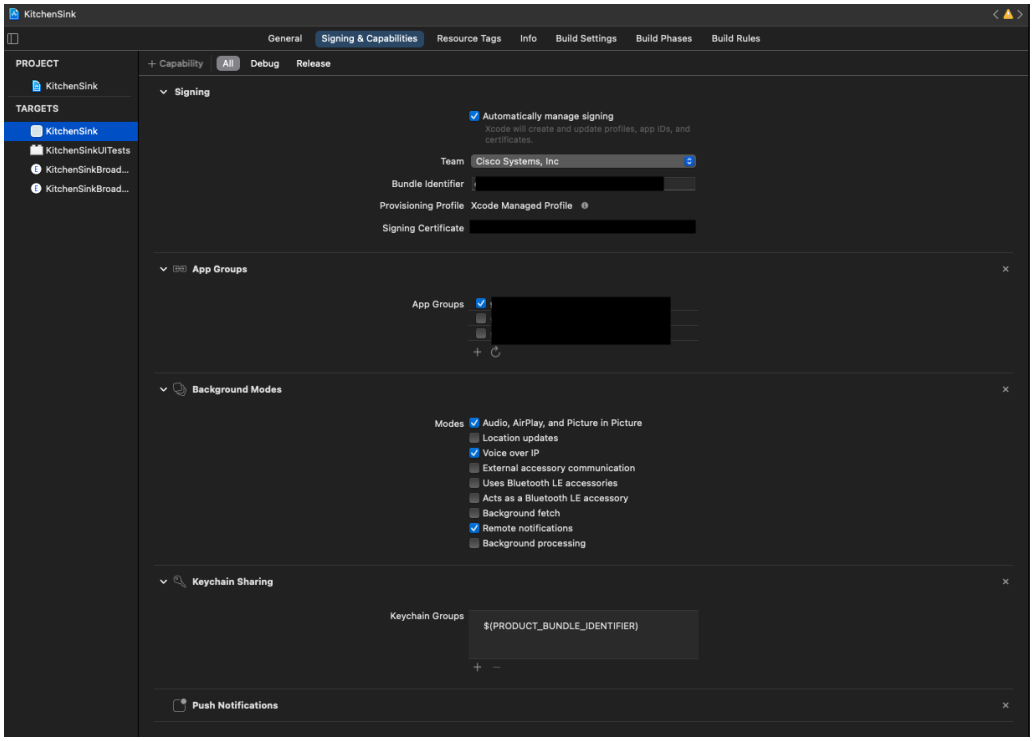
Table of Contents

Install

Assuming you already have an iOS project, e.g. *KitchenSinkApp*, for your iOS app, here are the steps to integrate the Cisco Webex iOS SDK into your project:

1. Drag and drop the `webexsdk.framework`, `util_ios.framework`, `wbxaecodec.framework`, `wbxaudioengine.framework` file into your app project
2. Set these frameworks as `Embed & Sign` in your app target
3. set `ENABLE_BITCODE=No`

4. Modify the `Signing & Capabilities` section in your xcode project as follows



Note

1. From 3.0.0.2-beta onwards, we have a new entry point for the SDK. i.e: `webex.initialize(:completionHandler)`. This needs to be invoked before any other API(even before Auth). This method internally bootstraps the SDK and all it's components.
2. We have transitioned from using a `Response` class wrapped around a `Result` object to just using a `Result` class in our completion handlers

Previous syntax:

```
webex.phone.dial("coworker@acm.com", option: MediaOption.audioVideo(local:
..., remote: ...)) { response in
        switch response.result {
        case .success(let call):
            call.onConnected = {
                // ...
            }
            call.onDisconnected = { reason in
                // ...
            }
        case .failure(let error):
            // failure
        }
    }
```

New syntax:

```
webex.phone.dial("coworker@acm.com", option: MediaOption.audioVideo(local:
..., remote: ...)) { result in
        switch result {
        case .success(let call):
            call.onConnected = {
                // ...
            }
            call.onDisconnected = { reason in
                // ...
            }
        case .failure(let error):
            // failure
        }
    }
```

Authentication

1. Instantiating `OAuthAuthenticator`. We were using client scope as a constructor argument which is now removed because scope is added internally.

Previous syntax for OAuth:

```
    let clientId = "$YOUR_CLIENT_ID"
    let clientSecret = "$YOUR_CLIENT_SECRET"
    let scope = "spark:all"
    let redirectUri = "https://webexdemoapp.com"

    let authenticator = OAuthAuthenticator(clientId: clientId,
clientSecret: clientSecret, scope: scope, redirectUri: redirectUri)
    let webex = Webex(authenticator: authenticator)

    if !authenticator.authorized {
        authenticator.authorize(parentViewController: self) { success in
            if !success {
                print("User not authorized")
            }
        }
    }
```

New Syntax for OAuth

```
    let clientId = "$YOUR_CLIENT_ID"
    let clientSecret = "$YOUR_CLIENT_SECRET"
    let redirectUri = "https://webexdemoapp.com/redirect"

    let authenticator = OAuthAuthenticator(clientId: clientId,
clientSecret: clientSecret, redirectUri: redirectUri, emailId: "")
    let webex = Webex(authenticator: authenticator)
```

```
    webex.initialize { isLoggedIn in
            if isLoggedIn {
                print("User is authorized")
            } else {
                authenticator.authorize(parentViewController: self) {
success in
                    if success {
                        print("Login successful")
                    } else {
                        print("Login failed")
                    }
                }
            }
        }
```

2. If you're Guest ID authentication (JWT-based) using JWTAuthenticator authorization. In this version, we have added completion handler which gets invoked when authorization process is done.

Previous syntax

```
    let authenticator = JWTAuthenticator()
    let webex = Webex(authenticator: authenticator)

    if !authenticator.authorized {
        authenticator.authorizedWith(jwt: myJwt)
    }
```

New syntax

```
    let authenticator = JWTAuthenticator()
    let webex = Webex(authenticator: authenticator)

    webex.initialize { [weak self] isLoggedIn in
            guard let self = self else { return }
            if isLoggedIn {
                print("User is authorized")
            } else {
                authenticator.authorizedWith(jwt: myJwt) { success in
                    if success {
                        print("Login successful")
                    } else {
                        print("Login failed")
                        return
                    }
                })
            }
        }
```

Examples of transition

Here are some examples to transition from Webex SDK v2 to Webex SDK v3:

1. `Phone.register()` API is no more required because this happens at the time of login internally. You have to comment/remove this.
2. `Phone.deregister()` API is no more required. You have to comment/remove this.
3. `JWTAuthenticator.isAuthorized()` is a work in progress

In Progress APIs

Below APIs are a work in progress which will be available in subsequent releases

1. letIn APIs.
2. AuxStream APIs.
3. MediaOption Moderator APIs
4. MediaOption Pin APIs
5. Authenticator `refreshToken` API
6. Multistream API
7. Read receipt API
8. MediaOption.layout
9. `oniOSBroadcastingChanged()` API