Cisco Webex iOS SDK

The Cisco Webex iOS SDK makes it easy to integrate secure and convenient Cisco Webex messaging and calling features in your iOS apps.

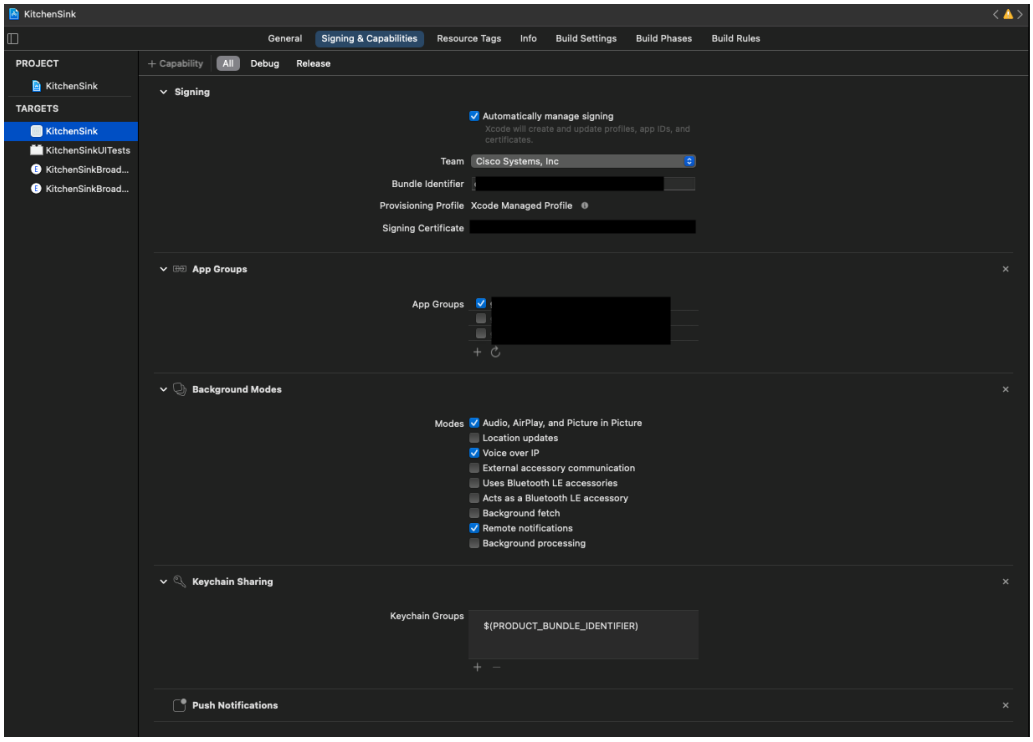
This SDK is written in Swift 5 and requires **iOS 13** or later.

Table of Contents

Install

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



Usage

To use the SDK, you will need Cisco Webex integration credentials. If you do not already have a Cisco Webex account, visit Webex for Developers to create your account and register your integration. Your app will need to authenticate users via an OAuth grant flow for existing Cisco Webex users or a JSON Web Token for guest users without a Cisco Webex account.

See the iOS SDK area of the Webex for Developers site for more information about this SDK.

Example

Here are some examples of how to use the iOS SDK in your app.

1. Create the Webex instance using Webex ID authentication (OAuth-based):

```
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. Create the Webex instance with Guest ID authentication (JWT-based):

```
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
                }
            })
        }
    }
```

3. Use Webex service:

```
webex.spaces.create(title: "Hello World") { result in
    switch result {
    case .success(let space):
        // ...
    case .failure(let error):
        // ...
    }
}

// ...

webex.memberships.create(spaceId: spaceId, personEmail: email) {
result in
    switch result {
        case .success(let membership):
            // ...
        case .failure(let error):
            // ...
        }
    }
}
```

4. Make an outgoing call:

```
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
    }
}
```

5. Make an outgoing CUCM call:

```
webex.phone.dial("+1180012345", option: MediaOption.audioVideo(local:
..., remote: ...)) { result in
    switch result {
    case .success(let call):
        call.onConnected = {
            // ...
        }
        call.onDisconnected = { reason in
            // ...
        }
    case .failure(let error):
        // failure
    }
}
```

6. Receive a call:

```
webex.phone.onIncoming = { call in
    call.answer(option: MediaOption.audioVideo(local: ..., remote:
...)) { error in
    if let error = error {
        // failure
    }
    else {
        // success
    }
}
```

7. Make an space call:

```
webex.phone.dial(spaceId, option: MediaOption.audioVideo(local: ...,
remote: ...)) { result in
    switch result {
    case .success(let call):
        call.onConnected = {
            // ...
        }
        call.onDisconnected = { reason in
            // ...
        }
        call.onCallMembershipChanged = { changed in
            switch changed {
            case .joined(let membership):
                //
            case .left(let membership):
```

```
                //
            default:
                //
            }
        }
    case .failure(let error):
        // failure
    }
}
```

8. Screen share (view only):

```
webex.phone.dial("coworker@acm.com", option:
MediaOption.audioVideoScreenShare(video: (local: ..., remote: ...))) {
ret in
    switch ret {
    case .success(let call):
        call.onConnected = {
            // ...
        }
        call.onDisconnected = { reason in
            // ...
        }
        call.onMediaChanged = { changed in
            switch changed {
                ...
            case .remoteSendingScreenShare(let sending):
                call.screenShareRenderView = sending ? view : nil
            }
        }
    case .failure(let error):
        // failure
    }
}
```

9. Post a message:

```
let plain = "foo"
let markdown = "**foo**"
let html = "<strong>foo</strong>"
```

```
let text = Message.Text.html(html: html)
webex.messages.post(text, toPersonEmail: emailAddress,
completionHandler: { response in
    switch response.result {
    case .success(let message):
        // ...
```

```
    case .failure(let error):
        // ...
    }
}
```

```
let text = Message.Text.markdown(markdown: markdown, html: html,
plain: text)
webex.messages.post(text, toPersonEmail: emailAddress) { result in
    switch result {
    case .success(let message):
        // ...
    case .failure(let error):
        // ...
    }
}
```

10. Receive a message event:

```
webex.messages.onEvent = { messageEvent in
    switch messageEvent{
    case .messageReceived(let message):
        // ...
    case .messageDeleted(let messageId):
        // ...
    }
}
```

11. send read receipt of a message

```
webex.messages.markAsRead(spaceId: spaceId, messageId: messageId,
completionHandler: { result in
     switch result {
     case .success(_):
         // ...
     case .failure(let error):
         // ...
     }
})
```

12. receive a membership event

```
webex.memberships.onEvent = { membershipEvent in
      switch membershipEvent {
      case .created(let membership):
          // ...
```

```
        case .deleted(let membership):
            // ...
        case .update(let membership):
            // ...
        case .messageSeen(let membership, let lastSeenId):
            // ...
        }
}
```

13. get read statuses of all memberships in a space

```
webex.memberships.listWithReadStatus(spaceId: spaceId,
completionHandler: { response in
        switch response.result {
        case .success(let readStatuses):
            // ...
        case .failure(let error):
            // ...
        }
})
```

14. receive a space event

```
webex.spaces.onEvent = { spaceEvent in
        switch spaceEvent {
        case .create(let space):
            // ...
        case .update(let space):
            // ...
        case .spaceCallStarted(let spaceId):
            // ...
        case .spaceCallEnded(let spaceId):
            // ...
        }
}
```

15. get read status of a space for login user

```
webex.spaces.getWithReadStatus(spaceId: spaceId, completionHandler: {
response in
        switch response.result {
        case .success(let spaceInfo):
            if let lastActivityDate = spaceInfo.lastActivityDate,
                let lastSeenDate = spaceInfo.lastSeenActivityDate,
                lastActivityDate > lastSeenDate {

                // space is unreadable
```

```
            } else {

                // space is readable
            }
        case .failure(let error):
            // ...
        }
})
```

16. get meeting detail of a space

```
webex.spaces.getMeetingInfo(spaceId: spaceId, completionHandler: {
result in
        switch result {
        case .success(let meetingInfo):
            // ...
        case .failure(let error):
            // ...
        }
})
```

17. get a list of spaces that have ongoing call

```
webex.spaces.listWithActiveCalls(completionHandler: { result in
    switch result {
    case .success(let spaceIds):
        // ...
    case .failure(_ ):
        // ...
    }
})
```

18. Change the layout for the active speaker and other attendee composed video

```
let option: MediaOption = MediaOption.audioVideo(local: ..., remote:
...)
option.layout = .grid

webex.phone.dial(spaceId, option: option) { ret in
    // ...
}
```

19. Background Noise Removal(BNR)

    19.1 Enable BNR

```
webex.phone.audioBNREnabled = true
```

19.2 Set BNR mode, the default is `.HP`. It only affects if setting `audioBNREnabled` to true.

```
webex.phone.audioBNRMode = .HP
```

License

See LICENSE for details.