Cisco Webex Android SDK

> The Cisco Webex™ Android SDK Version 3.0.0.2-beta

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

This SDK is built with **Android SDK Tools 29** and requires **Android API Level 24** or later.

Table of Contents

Install

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

1. Put AAR file in libs folder of your Android project
2. Open the project level Gradle file and add the following lines under the repositories tag, which is nested under allprojects.

```
allprojects {
  repositories {
      jcenter()
      google()
      flatDir { dirs 'aars'} //add this line
  }
}
```

3. Add the following dependency in module level Gradle file and press sync-now

```
implementation files('libs/Webex-SDK-v3.0.0.2-beta.aar')
```

Usage

To use the SDK, you will need Cisco Webex integration credentials. If you do not already have a Cisco Webex account, visit the Cisco Webex for Developers portal to create your account and register an 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 Android SDK area of the Cisco Webex for Developers site for more information about this SDK.

Examples

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

1. Create a new `Webex` instance using Webex ID authentication (OAuth-based):

```
val clientId: String = "YOUR_CLIENT_ID"
val clientSecret: String = "YOUR_CLIENT_SECRET"
val redirectUri: String = "https://webexdemoapp.com"

val authenticator: OAuthWebViewAuthenticator =
OAuthWebViewAuthenticator(clientId, clientSecret, redirectUri)
val webex = Webex(application, authenticator)
authenticator.authorize(loginWebview, CompletionHandler { result ->
      if (result.error != null) {
          //Handle the error
      }else{
          //Authorization successful
      }
  })
```

2. Create a new `Webex` instance using JWT authentication

```
val token: String = "jwt_token"

val authenticator: JWTAuthenticator = JWTAuthenticator()
val webex = Webex(application, authenticator)
authenticator.authorize(token, CompletionHandler { result ->
      if (result.error != null) {
          //Handle the error
      }else{
          //Authorization successful
      }
  })
```

3. Check if user is already authorised

```
webex.attemptToLoginWithCachedUser(CompletionHandler { result ->
    if (result.error != null) {
        //already authorised
    }else{
        //Need to authenticate again
    }
})
```

4. Create a new Cisco Webex space, add users to the space, and send a message:

```
// Create a Cisco Webex space:
webex.spaces.create("Hello World", null, CompletionHandler<Space?> {
result ->
        if (result.isSuccessful) {
            val space = result.data
        } else {
            val error= result.error
        }
    })

// Add a user to a space:
webex.memberships.create("spaceId", null, "person@example.com", true,
CompletionHandler<Membership?> { result ->
        if (result.isSuccessful) {
            val space = result.data
        } else {
            val error= result.error
        }
    })

// Send a message to a space:
webex.messages.postToSpace("spaceId", Message.Text.plain("Hello"),
null, null, CompletionHandler<Message> { result ->
        if(result != null && result.isSuccessful){
            val message = result.data
        }
    })
```

5. Make an outgoing call:

```
webex.phone.dial("person@example.com", MediaOption.audioVideo(local,
remote), CompletionHandler {
        val call = it.data
        call?.setObserver(object : CallObserver {
            override fun onConnected(call: Call?) {
                super.onConnected(call)
            }

            override fun onDisconnected(event: CallDisconnectedEvent?)
{
                super.onDisconnected(event)
            }

            override fun onFailed(call: Call?) {
                super.onFailed(call)
            }
        })
    })
```

6. Receive a call:

```
 webex.phone.setIncomingCallListener(object :
Phone.IncomingCallListener {
        override fun onIncomingCall(call: Call?) {
            call?.answer(MediaOption.audioOnly(), CompletionHandler {
                if (it.isSuccessful){
                   // ...
                }
            })
        }
    })
```

7. Screen sharing:

```
 webex.phone.dial("spaceId",
MediaOption.audioVideoSharing(Pair(localView, remoteView),
screenShareView), CompletionHandler {
        if(it.isSuccessful){
            val call = it.data
            call?.setObserver(object :CallObserver{
                override fun onConnected(call: Call?) {
                    super.onConnected(call)
                }

                override fun onDisconnected(event:
CallDisconnectedEvent?) {
                    super.onDisconnected(event)
                }
            })
        } else {
            val error = it.error
        }
    })
```

8. Start/stop sharing screen:

```
call.startShare("callid")
call.stopShare("callid")
```

9. Post a message

```
webex.messages.postToPerson(email, Message.Text.markdown("**Hello**",
null, null), null, CompletionHandler { result ->
    if (result.isSuccessful) {
        //posting a message is successful
    } else {
```

```
            val error = it.error
            //handle the error
        }
    })
```

10. Send Read Receipts

```
//Mark all exist messages in space as read
webex.messages.markAsRead(spaceId)

//Mark existing message before pointed message(include) in space as
read
webex.message.markAsRead(spaceId, messageId)
```

11. Get read status of a space

```
 webex.spaces.getWithReadStatus(spaceId, CompletionHandler { result ->
            if (result.isSuccessful) {
                //show the data
            } else {
                //handle error
            }
        })
```

License

See LICENSE for details.