cmbSDK Toolkit (v2.7.x)

Overview

Description

Implement cmbSDK Toolkit in your application

To implement cmbSDK Toolkit in your application, please follow these instructions. Note that **cmbSDK Toolkit depends on cmbSDK** and you should always import cmbSDK in your project to use the cmbSDK Toolkit.

- Android
- iOS

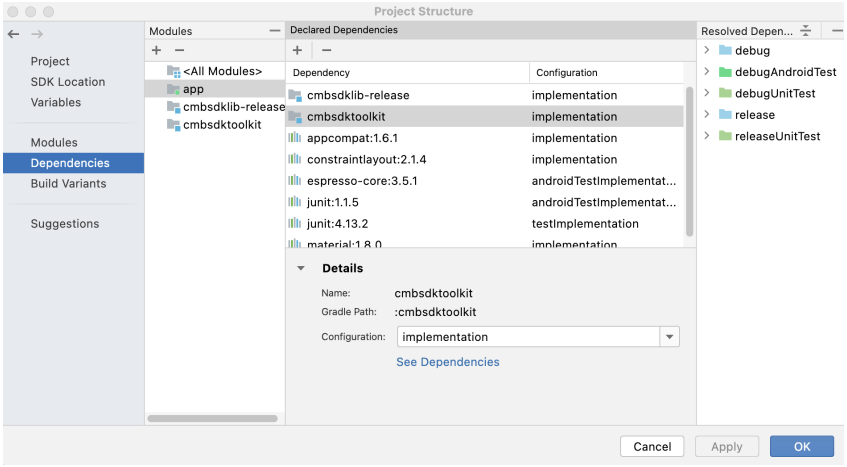
1. First of all make sure that your application **supports Kotlin**. cmbSDK Toolkit **can't be used in applications that don't support Kotlin**.
2. Download the **Cognex Mobile Barcode SDK** for Android and **cmbSDK Toolkit** from the Cognex Mobile Barcode Scanner Solutions page.
3. Start Android Studio and add the AAR files as a modules to your project. Starting from Android Studio **v4.2.1** and later, .AAR files can't be imported directly as modules. You need to import the Gradle or Eclipse project:

    1. Right-click your app module, select *New* > *Module* > *Import*.

    2. In the Source directory field browse the **cmbsdktoolkit** directory that contains the build.gradle file and the cmbsdktoolkit.aar file inside. In the Module name field you should see **:cmbsdktoolkit,** and click **Finish**. In the same way import cmbSDK.

4. After the new modules are available, right-click your app module, select the **Open Module Settings**, and choose the **Dependencies** tab.

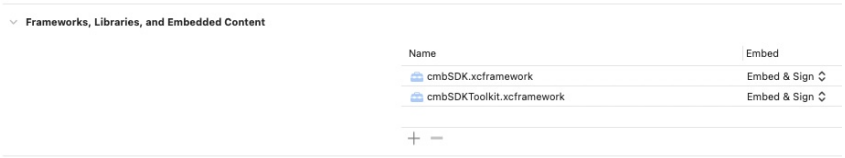
5. Click the **+** sign at the top of the Declared Dependencies dialog box and select the **3 Module dependency**.
6. Select **cmbsdktoolkit** and **cmbsdklib-release** from the popup window and click **OK**, making them available under the **Dependencies** tab. Also com.squareup.moshi:moshi-kotlin:1.13.0 dependency is needed to be linked inside your project because of the cmbsdktoolkit module.



7. Sync your project with the latest changes in the gradle files. Your application build.gradle should contain these dependencies:

1. Download the **Cognex Mobile Barcode SDK** for iOS and **cmbSDK Toolkit** from the Cognex Mobile Barcode Scanner Solutions page.
2. Add cmbSDK and cmbSDKToolkit xcframeworks to your Xcode project.
3. Use import statements in your source files to use the API.

```
import cmbSDK
import cmbSDKToolkit
```



Dynamic symbology settings

Description

cmbSDK Toolkit contains a *Dynamic Symbology Settings* module that can be used to configure Cognex readers and the camera API through a symbology settings screen in your application. If you want to see how it works in live, I would suggest to try one of the Cognex mobile applications (Quick Setup, Cognex Browser, Cognex Keyboard).

The module displays all known barcode type by connected reader. If user turns on the symbology, a new gear icon is displayed where further sub-symbology types (if applicable) and symbology dependent settings are available.

There are checks implemented to prevent that all sub-symbology types are turned off.

Certain symbology types have *Standard* as sub-symbology type, that means the main symbol type and cannot be turned off (disabled).

Here is how to implement in your application

- Android
- iOS

Inside the cmbsdktoolkit module, dynamic symbologies settings are implemented as fragment that can be attached from any fragment manager.
This fragment expects the reader device object as input parameter on which symbology settings will be applied.

```
parentFragmentManager.beginTransaction()
  .replace(
      R.id.fragment_holder,
      SymbologySettingsFragment.newInstance(readerDevice),
      SymbologySettingsFragment.TAG
  )
  .addToBackStack(SymbologySettingsFragment.TAG)
  .commit()
```

If the reader device is not connected or disconnected while this fragment is shown, symbology settings fragment will be closed and removed from the back stack with warning message.

The symbology settings module is available as a UIViewController instance that has to be instantiated as the following code snippet demonstrates.

```
let symbologySettingsVC = SymbologySettingsTVC(dataManSystem: self.datamanSystem, style: UITableView.Style.grouped)
```

Please note that you have to pass a `CDMDataManSystem` object to let the module communicate with the reader. If you are using the high level `CMBReaderDevice` class you can get the underlying `CDMDataManSystem` object by calling the `dataManSystem` instance method on your `CMBReaderDevice` object.

Bluetooth pairing

Description

cmbSDK Toolkit contains a Pairing module that makes the implementation easy to connect to Bluetooth Direct Connect enabled Cognex readers. If you want to see how it works in live, I would suggest to try one of the Cognex mobile applications (Quick Setup, Cognex Browser, Cognex Keyboard).

There are several possibilities within the Pairing module to establish a connection with a BT enabled Cognex device:

- One-Step Pairing: This is the easiest approach, as the user needs to read only a DataMatrix code from the phone's screen and after accepting the OS Pairing request information, the reader and the mobile device are paired and connected to each other.

- Advanced Mode: If the above method does not work for some reason, Pairing modul provides an alternative way to establish the connection. In this case user needs to put the reader into discoverable mode (either with a data matrix code or following the steps shown on the mobile device and executing them on the reader's OLED screen) and select the reader to connect to.

Here is how to implement in your application

- Android
- iOS

First we need to add required bluetooth permissions in the application manifest file:

```
    <!-- Request legacy Bluetooth permissions on older devices. -->
    <uses-permission
        android:name="android.permission.BLUETOOTH"
        android:maxSdkVersion="30" />
    <uses-permission
        android:name="android.permission.BLUETOOTH_ADMIN"
        android:maxSdkVersion="30" />
    <uses-permission
        android:name="android.permission.ACCESS_FINE_LOCATION"
        android:maxSdkVersion="30"
        tools:ignore="CoarseFineLocation" />
    <!---->

    <uses-permission
        android:name="android.permission.BLUETOOTH_SCAN"
        android:usesPermissionFlags="neverForLocation"
        tools:targetApi="s" />
    <uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
```

Inside the cmbsdktoolkit module, bluetooth pairing functionality is implemented as fragment that can be attached from any fragment manager.

This fragment has two optional input parameters:

1. **connectAfterPairing** - true means bluetooth reader device will be created and connection to it is performed automatically after successful pairing. If you use false, you can start the process manually to create and connect to bluetooth reader device. Default is **true**
2. **awaitUserApprovalOnConnect** - true means that screen where user need to allow connection to the bluetooth reader device to be completely finished will be shown. If false is set no interaction from user is needed. Default is **true**

```
parentFragmentManager.beginTransaction()
  .replace(
      R.id.fragment_holder,
      BluetoothPairingFragment.newInstance(),
      BluetoothPairingFragment.TAG
  )
  .addToBackStack(BluetoothPairingFragment.TAG)
  .commit()
```

The parent activity **need to implement BluetoothPairingFragmentListener**:

```
class MainActivity : AppCompatActivity(), BluetoothPairingFragmentListener
```

- **onBluetoothDevicePaired(device: BluetoothDevice)** - If **connectAfterPairing** input param is set to true, connection to the bluetooth device will be performed automatically after successful pairing. If it is set to false, then you can start the connection process.

```
override fun onBluetoothDevicePaired(device: BluetoothDevice) {
        // In BluetoothPairingFragment as input param we've set connectAfterPairing to true, which means
        // that connection to the bluetooth device will be performed automatically after successful pairing.
        // If we set that to false, from here we can start the connection process
}
```

- **onBluetoothDeviceConnectionFailed(error: Throwable)** - The throwable error object contains message that explains the cause why the connection has failed

```
override fun onBluetoothDeviceConnectionFailed(error: Throwable) {
        Snackbar.make(
            findViewById(R.id.fragment_holder),
            getString(R.string.bluetooth_pairing_error, error.localizedMessage),
            Snackbar.LENGTH_LONG
        ).show()
}
```

- **onBluetoothReaderDeviceConnected(reader: BluetoothReaderDevice)** - There is a valid connection to the bluetooth reader device that is received as input parameter. You can start to configure the reader device or to scan. Do not forget to close and remove the BluetoothPairingFragment from the back stack if needed.

```
override fun onBluetoothReaderDeviceConnected(reader: BluetoothReaderDevice) {
        reader.setReaderDeviceListener(this)

        readerDevice = reader

        clearAllFragments()
        configureReaderDevice()

}
```

- Add Bluetooth usage description entries to your Info.plist file in your Xcode project. You might be required to add values for both the NSBluetoothAlwaysUsageDescription and NSBluetoothPeripheralUsageDescription keys.

Privacy - Bluetooth Always Usage Description	String	The app discovers Bluetooth capable Cognex devices.
Privacy - Bluetooth Peripheral Usage Description	String	The app discovers Bluetooth capable Cognex devices.

- You have to add an observer to be notified about a successful pairing event

```
NotificationCenter.default.addObserver(self, selector: #selector(handlePeripheralDidPairNotification(_ :)), name: .cognexBT
```

- In your notification handler you have to save the paired device's UUID to load it later for reconnection.

```
@objc func handlePeripheralDidPairNotification(_ notification: NSNotification) {
    let peripheralUUID = notification.userInfo![pairedCognexBTDeviceUUIDKey] as! UUID
    // Save peripheralUUID...
    // Pop the presented pairing viewcontroller
}
```

- To present the pairing UI use the following code snippet

```
let settingsStoryBoard = UIStoryboard(name: "cmbSDKToolkit", bundle: Bundle(identifier: "com.cognex.cmbSDKToolkit"))
let btPairingVC = settingsStoryBoard.instantiateViewController(withIdentifier: "BT_AUTO_CONNECT_VC") as! BTAutoConnect
```

```
btPairingVC.peripheralUUID = // load previously paired device's UUID or use nil
let navController = UINavigationController(rootViewController: btPairingVC as! UIViewController)
```

- Please note that setting nil value for the peripheralUUID property makes the pairing module present the one-step-pairing screen to setup a new pairing. If you set a previously paired device's UUID the pairing module will try to connect to that device immediately.
- Please note that if you paired a certain reader with a certain iOS device and you removed the pairing info from the reader afterwards (for example by activating discoverable mode or applying factory reset) the subsequent connection attempt will fail and you have to manually remove the reader from the iOS Bluetooth settings to let the iOS initiate a new pairing process.