



CREDENCE ID

**Software Development Kit
5.6.1
Quick Start Guide**

Overview

The Credence ID SDK enables your Android applications to interact with various biometrics and secure document reading available on Credence ID devices such as:

- Fingerprint
- Face
- Iris
- SmartCard
- MRZ
- EPassport

This SDK not only allows developers to read data from various sources, but also perform the following operations.

- Template Creation (Finger, Face)
- Template Matching (Finger, Face)
- Biometric Verification(1:1)
- ICAO Document Reading

This guide assumes one is familiar with Android Studio IDE, Android SDK, ADB tool, creating a new Android Studio project, and basic application creation.

If you are new to any of the above topics, you may visit the following links to learn more.

- [Download Android Studio](#)
- [Building your first Android App.](#)
- [Using ADB \(Android Debug Bridge\)](#)

Installing Credence ID SDK

You may download the SDK at the following [link](#). You will see the following folders.

JARs	Contains all required JAR files in order to use SDK.
Docs	Contains JavaDocs for SDK

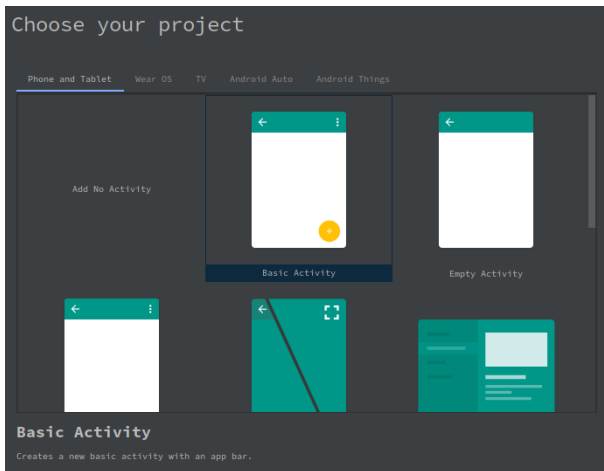
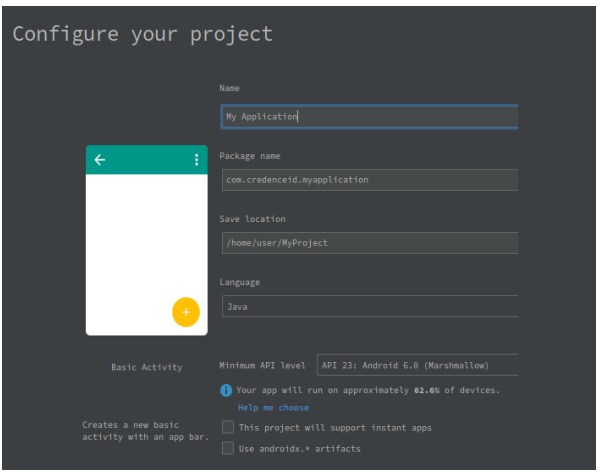
To use Credence Device SDK you must ensure **C-Service.apk** is installed on your Credence device.

Installing C-SDK

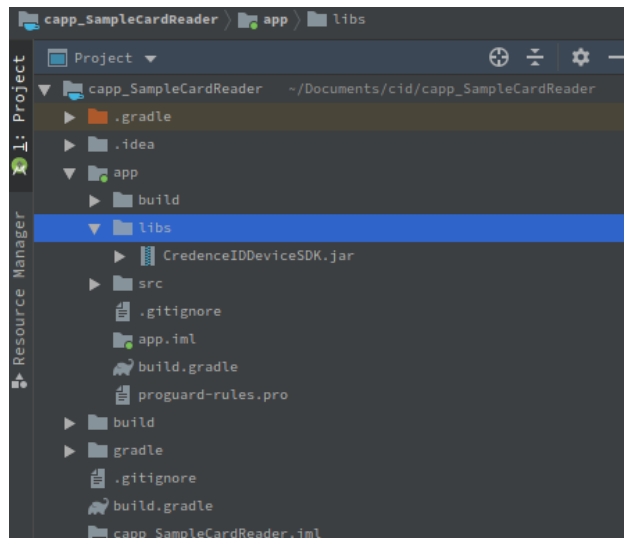
Before including the **CredenceIDDeviceSDK.jar** into your Android app you must first install the **C-SDK.apk** app to your Credence ID device. In order to do so use please update your operating system to the latest to obtain this release. Credence ID Device SDK (C-SDK.apk) application is now a part of Credence ID operating system.

Building Your First Credence SDK App

1. Create a new Android App project.
 - a. Select the **BasicActivity** option.
 - b. Select target API version **23**.

	
Choose BasicActivity .	Select API level 23 .

2. Include the **CredenceIDDeviceSDK.jar** file to your project.
 - a. Open up **Project Explorer**.
 - b. Copy over the JAR file to **app/libs** directory.



Copy JAR file to **app/libs** directory.

Getting started

There are two main ways to call the various APIs contained within the SDK. Either through creating a **BiometricsManager** object instance or extending your Activity with the **BiometricsActivity** class.

BiometricsManager

An instance of **BiometricsManager** must be instantiated. Once initialized this object may be used to invoke SDK APIs.

- SDK must be initialized with a call to *initializeBiometrics(OnBiometricsInitializedListener)*
- SDK must be de-initialized on application exit with a call to *finalizeBiometrics(boolean)*

BiometricActivity

You must extend your main Activity with this type. Once extended, you will be able to invoke any SDK API. Using this approach you do not need to worry about SDK initialization or uninitialization.

Sample Apps

Please take a look at any of the sample Android apps included in the SDK package to better understand how to use SDK APIs.

[Sample EPassport Project](#)

Demonstrates how to use all MRZ, ePassport, and ICAO APIs.

[Sample App Project](#)

A combination of all the above sample apps together using multiple activities and a shared common BiometricsManager object instance.

Signing Your Application

Starting with this release of our SDK all apps that you wish to install to a device **must** be signed with Credence ID keys. This is done for security reasons so that non-trusted apps are not installed to a device. There are two ways to sign an app, either through an app's source in AndroidStudio or using **apksigner** command line tool for a prebuilt APK.

In order to do this you will need to download Credence ID public keys which can be downloaded at the following [link](#).

Signing with AndroidStudio

If you wish to sign an app from source you may do so through the app's **build.gradle** file. If you goto the following [link](#) it will take you to the build.gradle file for our capp_SampleApp project. Inside the block **signingConfigs** you can see that the release APK is being signed with Credence ID keys. Make sure you set the key path to point to where-ever you have placed the **PublicKey.jks** file.

Now inside the **buildTypes.release** configuration block you will see a line which says **signingConfigs signingConfigs.release**. What this does is tell AndroidStudio that when it builds a release version of your APK to sign it with the key configuration specified by the **signingConfigs.release** block.

If you wish to sign a debug version of your APK while you are building/testing your app or to use it with the debugger you may simply add the line **signingConfigs.release** inside the **buildTypes.debug** config block.

Signing with ZipAlign/APKSigner

This method allows you to sign an already build APK. This is a two step process. First is to zipalign your APK, meaning to properly format it's contents to allow for a key to be placed inside it. Secondly to actually sign the APK. You may visit this [link](#) for zipalign and this [link](#) for apksigner. At a high level the two commands you will need are as follows:

```
zipalign -f -v 4 <input.apk> <output.apk>
```

```
apksigner sign --ks /path/to/CredenceID/PublicKey.jks <output_apk_from_zipalign.apk>
```

Contact Credence ID

California +1.888.CID.5452 (GMT-7) support@CredenceID.com 2335 Broadway, Suite 100 Oakland California 94612	Europe Marseilles, France
The Americas Mexico City, Mexico	Middle East Dubai, Abu Dhabi

