wexer

Android-Kotlin Content SDK Integration Guide

Date	02 June 2020
Guide Version	1.2

Wexer Content SDK enables developers to show On Demand content in the app in the form of different types of collections, list of classes, filter/search content, and play the videos.

This guide will show you how to install the SDK and use of its exposed methods.

Prerequisites

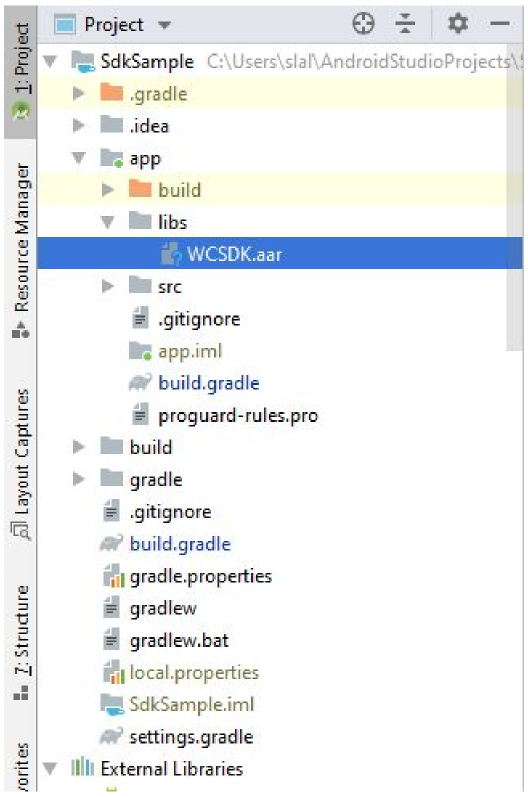
1. Support on Android Studio IDE.
2. Android APIs level support from 21 and above.

Step 1: SDK Setup

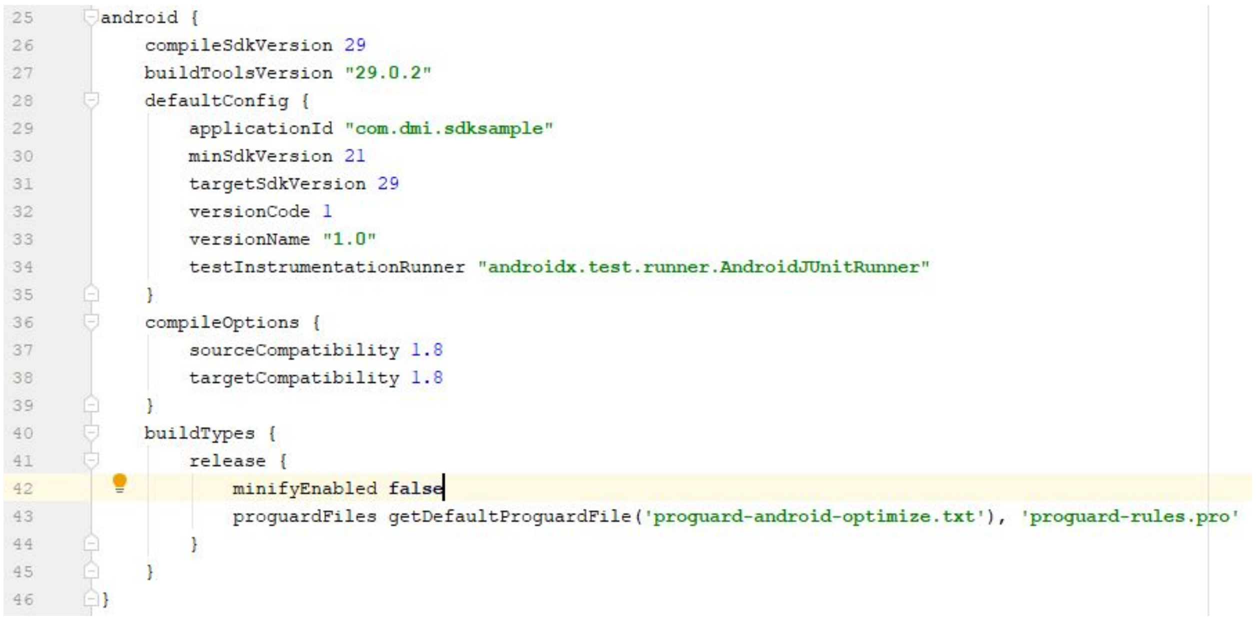
There are two ways to integrate SDK into an Android app.

Option 1 > As **.aar** file

1. First, put .aar file (***WCSDK.aar***) into ***libs*** folder of your app

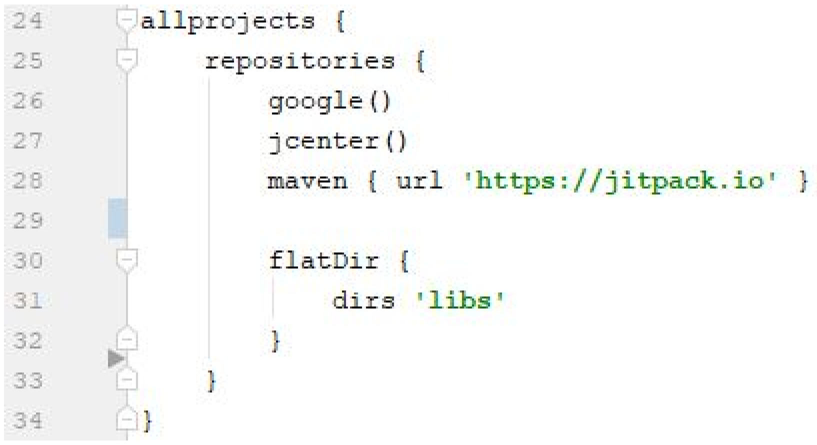


2. Set compile option to Java 8 as following in **app** module gradle file under android section (if your app compile version is below Java version 8 else no need to do this step)



```
android {
    compileSdkVersion 29
    buildToolsVersion "29.0.2"
    defaultConfig {
        applicationId "com.dmi.sdksample"
        minSdkVersion 21
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    compileOptions {
        sourceCompatibility 1.8
        targetCompatibility 1.8
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}
```

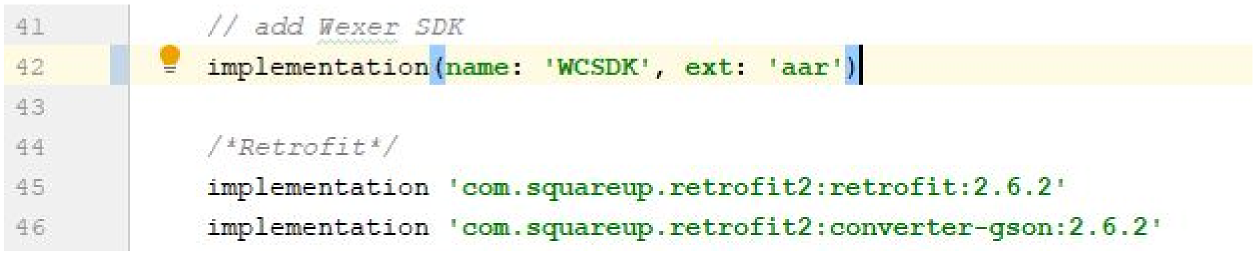
3. Add ***flatDir*** into your **root(project)** level gradle file as given below.



```
allprojects {
    repositories {
        google()
        jcenter()
        maven { url 'https://jitpack.io' }

        flatDir {
            dirs 'libs'
        }
    }
}
```

4. Include the following line into your **app** module gradle file and sync.



```
// add Wexer SDK
implementation(name: 'WCSDK', ext: 'aar')

/*Retrofit*/
implementation 'com.squareup.retrofit2:retrofit:2.6.2'
implementation 'com.squareup.retrofit2:converter-gson:2.6.2'
```

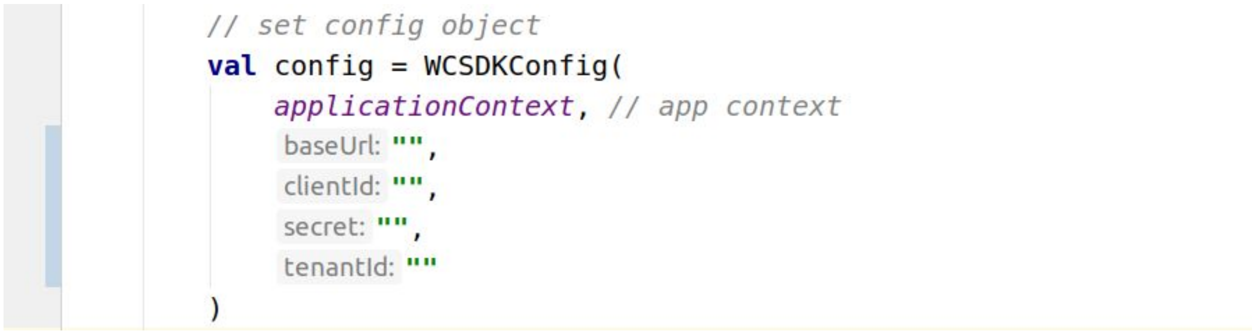
Option 2 > As **Gradle** Dependency

Details will be available once we will publish the SDK on gradle repository servers.

Step 2: SDK Initialization

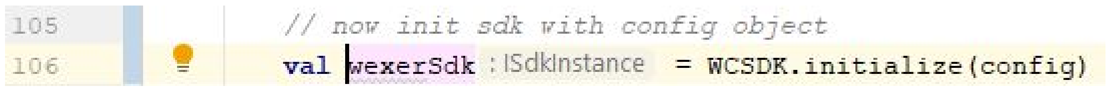
Below are the steps to initialise SDK. Please contact the Wexer sales team to obtain test keys.

1. Create WCSDKConfig object with specified values



```
// set config object
val config = WCSDKConfig(
    applicationContext, // app context
    baseUrl: "",
    clientId: "",
    secret: "",
    tenantId: ""
)
```

2. Now initialize SDK instance



```
// now init sdk with config object
val wexerSdk : ISdkInstance  = WCSDK.initialize(config)
```

Use this instance to call exposed methods of SDK in your app. For more details see step 3.

Note. wexerSdk instance may be null due to the config object not being in proper format, so try to pass it the same way as explained above.

3. Set localytics key

Localytics key can be set in two ways:

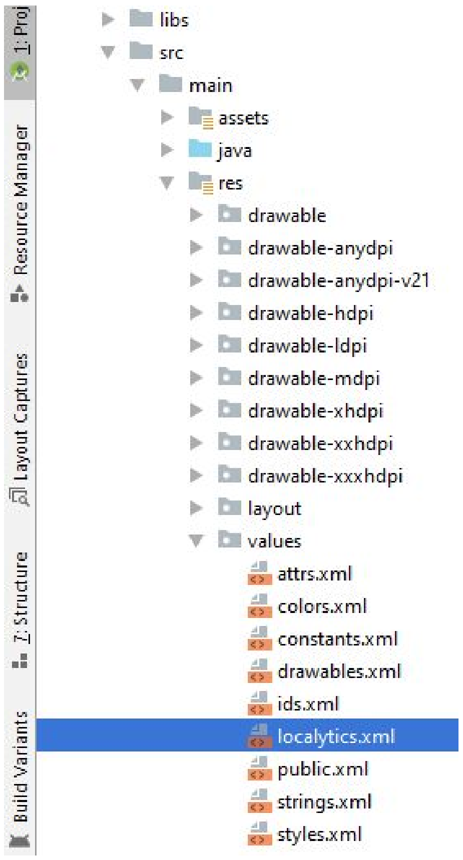
1. **Pre-set while SDK generation**

   Clients are required to provide localytics keys to us and we will use it while SDK generation. In this way, there will be specific SDK for different clients.

   Note. For the time being, a demo app (**WexerSDK-Android-Test**) localytics key is used while generating the .aar file.

2. **Set by Client app**

   Put **localytics.xml** config file at **res/values** folder under **app** module. This file can be downloaded from here. Now update **ll_app_key** string value with your app key. App key can be found from the user localytics dashboard -> setting page -> apps page.



```
libs
src
  main
    assets
    java
    res
      drawable
      drawable-anydpi
      drawable-anydpi-v21
      drawable-hdpi
      drawable-ldpi
      drawable-mdpi
      drawable-xhdpi
      drawable-xxhdpi
      drawable-xxxhdpi
      layout
      values
        attrs.xml
        colors.xml
        constants.xml
        drawables.xml
        ids.xml
        localytics.xml
        public.xml
        strings.xml
        styles.xml
```

Step 3: Call SDK Methods

After initialization, use SDK instance (created in step 2)

1. On Demand Collections -

```
wexerSdk.getOnDemandCollections(
    collectionId: String?,
    maxResult: Int?,
    mOnDemandCollectionListener: OnDemandCollectionListener?
)
```

2. On Demand Classes - It allows to fetch response with pagination which can be sorted on date/alphabet with “asc”/”dsc”

```
 wexerSdk.getOnDemandClasses(
   pageSize: Int,
   pageNumber: Int,
   filterBy: String,
   orderBy: String,
   mClassDataFetchListener: ClassDataFetchListener?
)
```

3. On Demand class detail - for single class having tag as 43794

```
wexerSdk.getOndemandClassDetails(
   classTag: String,
   mOnDemandClassDetailsListener: OnDemandClassDetailsListener?
)
```

4. Show On Demand classes filter parameters

```
 wexerSdk.getOnDemandMetadata(
       mOnDemandMetadataListener: OnDemandMetadataListener?
 )
```

5. Search & filter On Demand Classes - sorting can be applied on date/alpha.

```
var mOnDemandSearch = WCSDKOnDemandFilterRequest()
mOnDemandSearch.level  =ExerciseLevel.Advanced
// beginner intermediate advanced
mOnDemandSearch.query  ="yoga"  // provide search text in query parameters
mOnDemandSearch.provider="cycling"
wexerSdk.getOnDemandClassesForCriteria(
   mWCSDKOnDemandFilterRequest: WCSDKOnDemandFilterRequest,
   mOnDemandMetadataListener: OnDemandMetadataSearchListener?
)
```

6. Perform On Demand content :

   On Demand class can be performed only when the user has initiated a session, has a valid subscription to playback on demand content.

   a. Create user session with providing username (a unique value in the client application that identifies an app user)

```
wexerSdk.startSession(userName: String,
        mStartSessionListener: StartSessionListener?
)
```

b. Activate subscription

```
val userSubscription =
WCSDKUserSubscriptionRequest("2019-01-02T06:20:49.000",
"monthly")

 wexerSdk.activateSubscription(
    userSubscription,
    mSubscriptionActivatedListener:
SubscriptionActivatedListener?
)
```

c. Play On Demand - pass viewcontroller reference on which player shall be presented.

```
wexerSdk.performOnDemandContent(
    classTag: String,
    mOnDemandPerformListener: OnDemandPerformListener?
)
```

It will work if a user has logged in with a valid subscription. It will throw an error having code & userInfo with details .
Error - 403 - Not Subscribed
Step 1: Call startSession method for login
Step 2: call activateSubscription method

d. Define custom behaviours on play/pause/exit events.

   Client app will get video player events into the above method callback listener. So it is mandatory to pass OnDemandPerformListener object in it. Below is the signature of the callback listener.

```
interface OnDemandPerformListener {
      ....

      // This method will be called when video player start 
paying or
      // paused. Status value 1 for playing and 0 for paused

      fun playerStatus(status: Int)


      // This method will get called when video player closed 
and
      // it will return total played duration

      fun playerExit(duration: Long? = 0)

      ...
}
```