wexer

Android-Kotlin Content SDK Integration Guide

Date	14th April 2023
Guide Version	2.3

Wexer provides an SDK for Android that 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 WexerContent SDK.

There are two ways to integrate SDK into an Android app. If your environment supports an automated initialization using Gradle, please use Option 1. If you want to manually ingest the .arr file, please go to Option 2.

Install SDK

Prerequisites

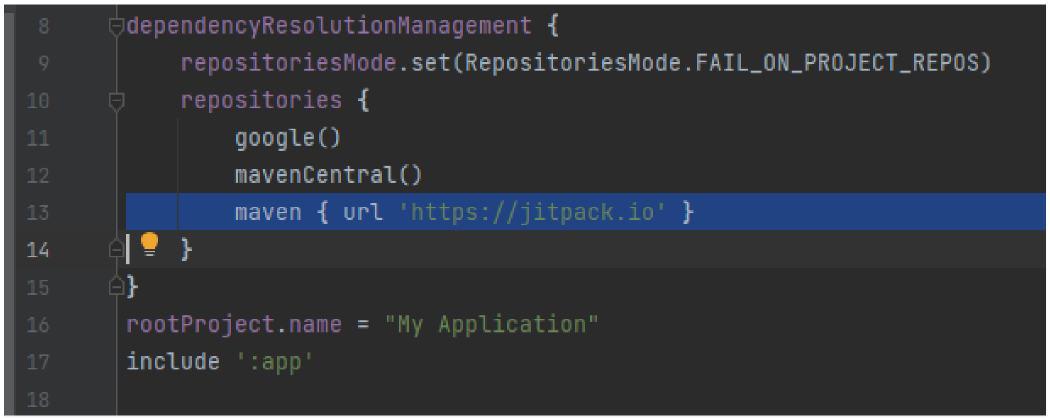
- Support on Android Studio IDE
- Android APIs level support from 21 and above
- Obtain access to the following repository in case you want to use the library as a gradle dependency. If you do not have this access, please contact Wexer:
  `https://github.com/wexervirtual/WexerContentSDK-Android/packages`
- Obtain the following details to configure the SDK. If you do not have these credentials, please contact Wexer.
  1. API BaseURL
  2. Client API Key
  3. Client Secret Key
  4. Tenant ID
  5. Localytics Key
  6. Subscription ID

Option 1: Automated as a Gradle dependency

1. Go to `app/build.gradle` and add the following lines as dependencies:

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

2. Go to the settings.gradle file and add the `maven { url 'https://jitpack.io' }` line below all the repositories as the below image.



```
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
        maven { url 'https://jitpack.io' }
    }
}
rootProject.name = "My Application"
include ':app'
```

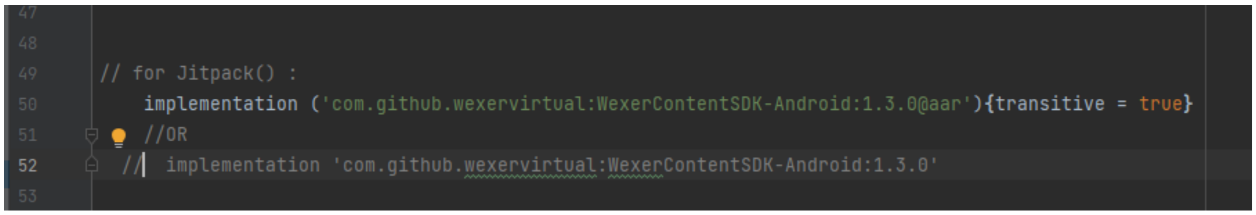
3. Add the following line in `gradle.properties` file:

```
android.enableJetifier=true
```

4. Add the following lines in the module `app/build.gradle:`

implementation('com.github.wexervirtual:WexerContentSDK-Android:1.3.0@aar'){transitive = true}

OR
implementation 'com.github.wexervirtual:WexerContentSDK-Android:1.3.0'



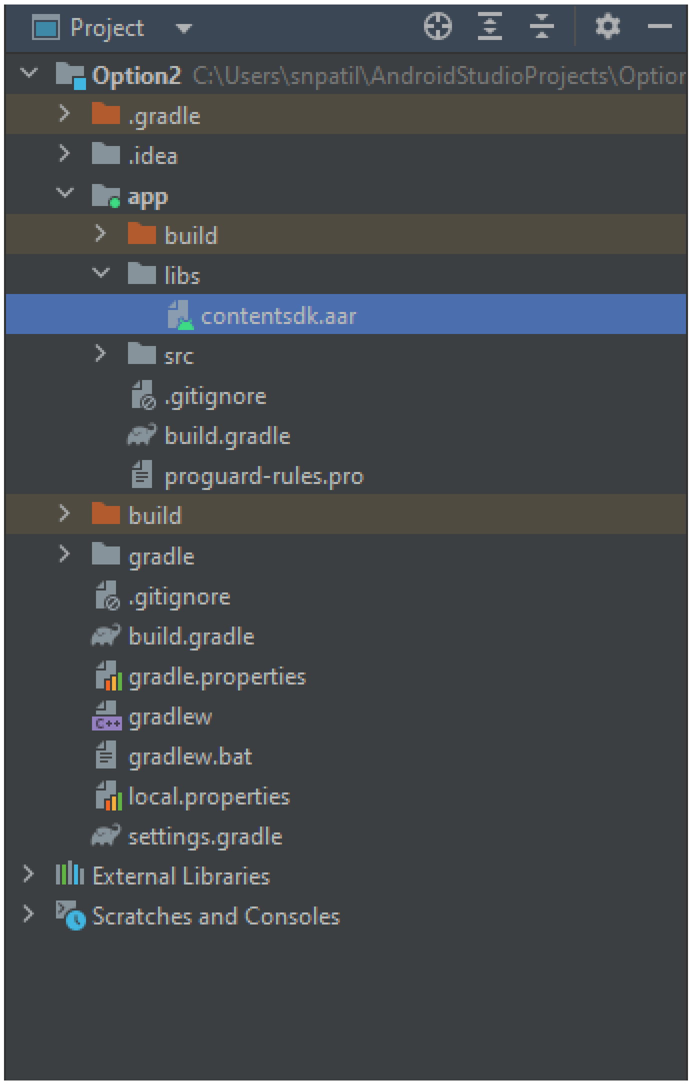
```


// for Jitpack() :
    implementation ('com.github.wexervirtual:WexerContentSDK-Android:1.3.0@aar'){transitive = true}
    //OR
  //  implementation 'com.github.wexervirtual:WexerContentSDK-Android:1.3.0'

```

Option 2: As .aar file

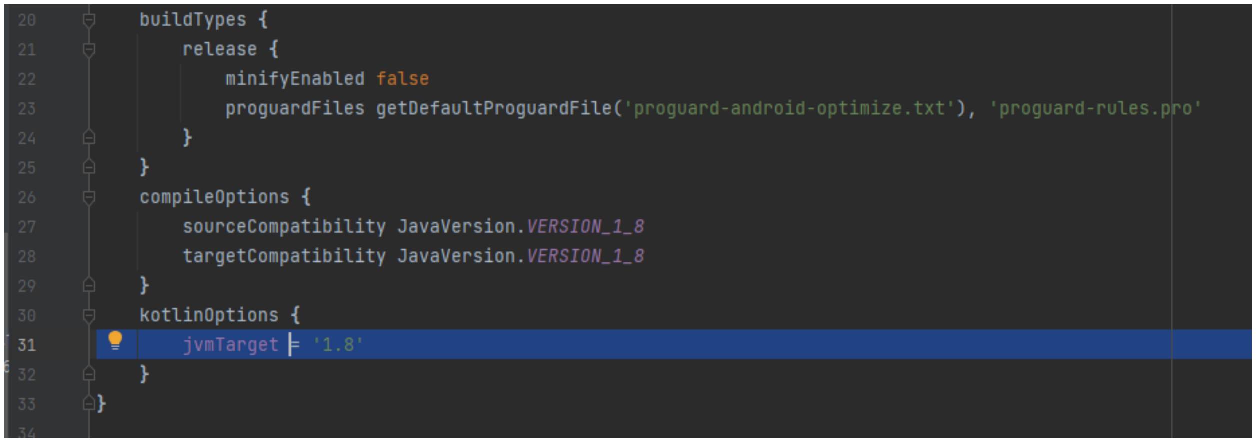
1. Download the .aar file from the GitHub repository

   https://github.com/wexervirtual/WexerContentSDK-Android/blob/master/contentsdk.aar

2. Put the .aar file (contentsdk.aar) into the `libs` folder of your app



3. Set the 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)



```
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
}
```

4. Go to `app/build.gradle` and add the following lines as dependencies:

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

5. Include the following line into your app module `app/build.gradle` file and initialize a sync.

```
implementation files('libs/contentsdk.aar')
```

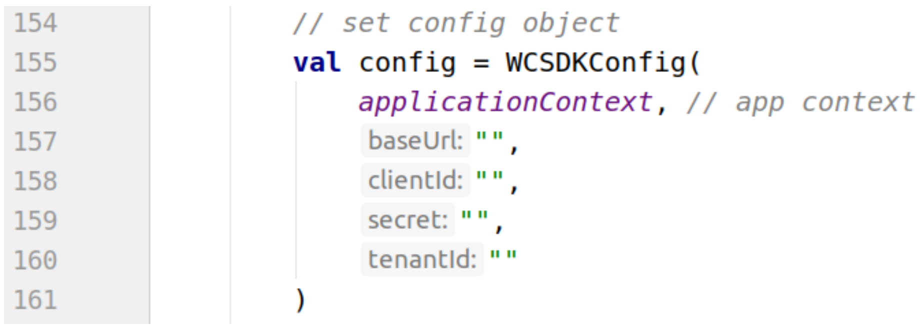
6. Add the following line in `gradle.properties` file:

```
android.enableJetifier=true
```

Initialize WexerContentSDK

1. Create a `WCSDKConfig` object with the specified values

```
            val config = WCSDKConfig (
                  applicationContext: Context
                  baseUrl: String,
                  clientId: String,
                  secret: String,
                  tenantId: String
            )
```



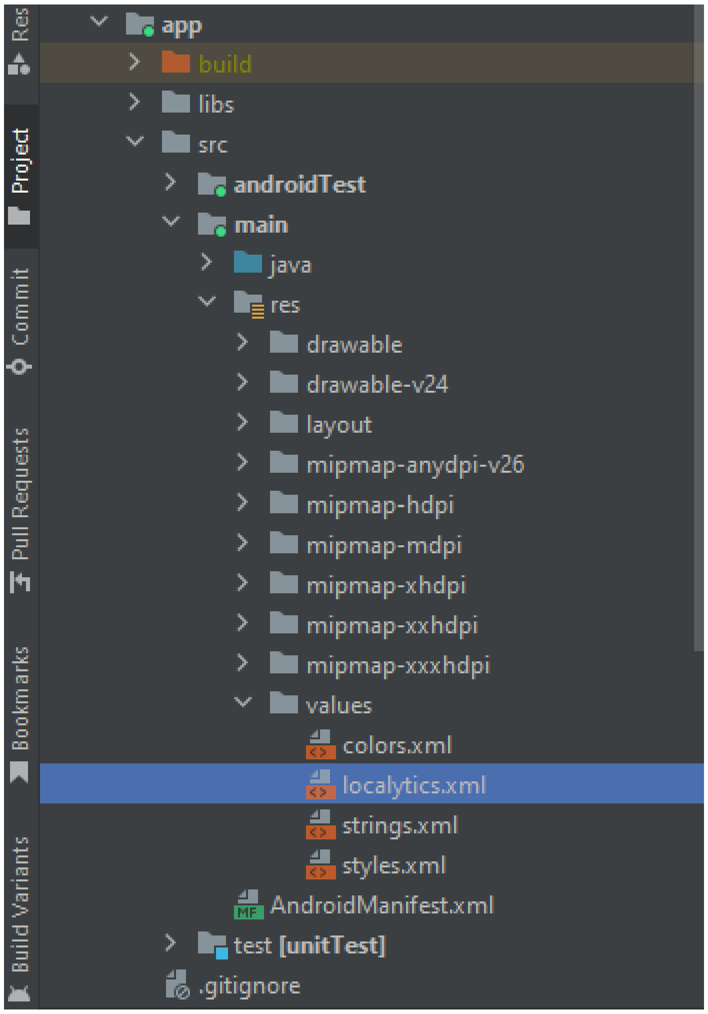
```
    // set config object
    val config = WCSDKConfig(
        applicationContext, // app context
        baseUrl: "",
        clientId: "",
        secret: "",
        tenantId: ""
    )
```

2. Initialize the SDK instance

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

Use this instance to call exposed methods of SDK in your app. For more details see step 3.
Note: The `wexerSdk` instance may be null due to the config object not being in a correct format, so make sure to pass it the same way as explained above.

3. Set the Localytics key. The Localytics key can be set as per the below
   a. Set by client app: Put the `localytics.xml` config file in the `res/values` folder under app module. This file can be downloaded from here. Update the `ll_app_key` string value, in the `localytics.xml` file with your Localytics key. If you do not have this key, please contact Wexer.



Call SDK methods

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

On Demand Collections

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

On Demand Classes

```
	wexerSdk.getOnDemandClasses(
		take: Int,
		skip: Int,
		sort: String,
		dir: String,
		mClassDataFetchListener: ClassDataFetchListener?
	)
```

On Demand class detail

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

Show On Demand classes filter parameters

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

Search and filter On Demand classes

```
        var mOnDemandSearch = WCSDKOnDemandFilterRequest()
        mOnDemandSearch.level = ExerciseLevel.Advanced
        //"Beginner", "Intermediate", "Advanced"

        mOnDemandSearch.query = "yoga"

        mOnDemandSearch.classLanguage = "en"

        mOnDemandSearch.provider = "cycling"

        mOnDemandSearch.take = 20
        //It is count of items to be fetched in single hit

        mOnDemandSearch.skip = 0
        //It is start of items to be skipped for which data is fetched

        mOnDemandSearch.dir = "desc"
        //"desc", "asc"

        mOnDemandSearch.sort = "scheduledate"
        //"scheduledate", "createddate" ,"name"

        mOnDemandSearch.categoryId = "1001"

        wexerSdk.getOnDemandClassesForCriteria(
                mWCSDKOnDemandFilterRequest: WCSDKOnDemandFilterRequest,
                mOnDemandMetadataListener: OnDemandMetadataSearchListener?
        )
```

Get Consent

```
        wexerSdk.getConsent(
                mGetConsentListener: GetConsentListener?
        )
```

Accept Consent

```
        wexerSdk.acceptConsent(
                mConsentTag: String,
                mAcceptConsentListener: AcceptConsentLister?
        )
```

Initialize user session and play a class

Before a class can be requested to play, the user must be logged in with a valid subscription. This section explains how to initialize an individual user session.

1. Create a user session, providing a username that can be any alphanumeric unique value. This session will remain active while the parent app is active. Upon closing the application, the session will terminate and the app will have to call the `startSession` method again to proceed.

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

2. Fetch all the classes;

```
wexerSdk.getOnDemandClasses(
            take: Int,
            skip: Int,
            sort: String,
            dir: String,
            mClassDataFetchListener: ClassDataFetchListener?
      )
```

3. Fetch class details; call the tag for the class that is to be fetched.

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

4. Play a class; call the tag for the class that is to be played.

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

Playing content

This section will explain which options are available while playing content using the SDK.

1. Create an instance of `videoViewBuilder` in your activity class.
```
var videoViewBuilder: ISdkInstance.VideoViewBuilder? = null
```

2. Create a method and initialize the `videoViewBuilder` instance.
```
videoViewBuilder = wexerSdk.getVideoViewBuilder()
```

3. Set `onDemandClass` in videoViewBuilder.
```
videoViewBuilder?.setOnDemandClass(
      onDemandClass, object: VideoPlayerStateListener{
            override fun playerStatus(status: Int) { }
            override fun playerExit(duration: Long?) { }
            override fun onViewTouchEvent() { }
            override fun
            currentVideoPosition(currentVideoPosition: Long){ }
            override fun totalVideoDuration(totalVideoDuration:
            Long) { }
      }
)
```

   Here, `VideoPlayerStateListener` will provide you following callbacks when the video is played:
   - `playerStatus` returns the current status of the player when video state is changed. Can be `PLAYER_STATE_PLAYING` or `PLAYER_STATE_PAUSED`.
   - `playerExit` returns the duration of watched video when the video is stopped.
   - `onViewTouchEvent` triggers on touching the video screen. Can be used to show/hide overlay.
   - `currentVideoPosition`: returns the current video position when the video is playing. Used to update the seekbar if `customOverlayView` is set to `true` (see step 4).
   - `totalVideoDuration`: returns the total video duration when the video is played. Used to update the seekbar if `customOverlayView` is set to `true` (see step 4).

4. **Overlay Views**. The SDK provides a default overlay view for small screens and a fullscreen player which can be customized. Furthermore, it’s possible to create custom overlay views for small screen and fullscreen and use them instead of the default overlay view. Both steps are explained below.

   Default overlay view

   1. Set the default overlay view.
      ```
      videoViewBuilder?.useCustomOverlayView(false)
      ```

   2. Modifying the default overlay view (optional): It will apply to both small and full screen view. The parameters below are usable individually or combined.
      ```
      videoViewBuilder?.setDefaultPlayButton(drawableResId =
      R.drawable.ic_default_play, colorTint = R.color.red)

      videoViewBuilder?.setDefaultPauseButton(drawableResId =
      R.drawable.ic_default_pause, colorTint = R.color.green)

      videoViewBuilder?.setDefaultFullScreenOpenButton(drawableRe
      sId = R.drawable.ic_default_open_fullscreen, colorTint =
      R.color.green)

      videoViewBuilder?.setDefaultFullScreenCloseButton(drawableR
      esId = R.drawable.ic_default_close_fullscreen, colorTint =
      R.color.red)

      videoViewBuilder?.setDefaultStopButton(drawableResId =
      R.drawable.ic_default_stop, colorTint =
      R.color.light_green)

      videoViewBuilder?.setDefaultSeekBarProgressColorTint(colorT
      int = R.color.red)

      videoViewBuilder?.setDefaultSeekBarThumbColorTint(colorTint
      = R.color.red)
      ```

Custom overlay view (optional)
In order to use the custom overlay view, you need to set the below properties (if you decide to go with the default overlay view provided by SDK, you can skip this step):

```
videoViewBuilder?.useCustomOverlayView(true)
```

(Only set this property if you have a small screen view. Refer to step 5 below)

```
videoViewBuilder?.setCustomOverlayView(R.layout.client_playback_o
verlay_view, object: OverlayViewProvider{
override fun getOverlayView(view: View)
      { //set click listeners on small screen overlay view here
      }
}, object: OverlayViewProvider{override fun getOverlayView(view:
View)
      {//set click listeners on on full screen overlay view here
      }
})
```

Here, both small and fullscreen views can be set. In case of individual fullscreen mode, you can skip setting `setCustomSmallScreenOverlayView()` above.

5. **Creating a small screen view (optional):** If you want to show the video in a small screen view, see the steps below. If just a full screen view is desired, skip this step and step 6.1).

    1. Add `WCSDKView` in your xml file:

    ```
          <com.dmi.mykotlinlib.vplayer.WCSDKView
          android:id="@+id/wcsdkView"
          android:layout_width="match_parent"
          android:layout_height="@dimen/videoViewHeight"/>
    ```

    2. Initialize the view in Activity/Fragment class:

    ```
     wcsdkView = findViewById(R.id.wcsdkView)
    ```

6. **Starting the video:** Start the video by calling the following methods:
    1. Small screen mode: If you have followed step 5 above, the video can be started in the defined player by executing the following line:

        ```
        videoViewBuilder?.startVideoIn(wcsdkView)
        ```

    2. Individual full screen mode: If only full screen video playback is desired, execute the following line:

        ```
        videoViewBuilder?.openFullScreen()
        ```

7. **Player controls:** The video playback can be controlled with the following methods: (Note: by default, the video will be in playing state)

    1. **Play video:** `videoViewBuilder?.playVideo()`

    2. **Pause video:** `videoViewBuilder?.pauseVideo()`

    3. **Stop video:** `videoViewBuilder?.stopVideo()`

        If this method is called, the video will be stopped and the play and pause methods will not work. Also, `VideoPlayerStateListener.onPlayerExit` will be invoked. Furthermore, if “small screen view” (step 5) is chosen, it is the responsibility of the developer to call this method once the video purpose has been delivered to release the player. If “small screen view” is skipped, this method will be called directly from `closeFullScreen()`. See step 7.4

    4. **Close fullscreen:** `videoViewBuilder?.closeFullScreen()`

        This method closes the full screen view. If “small screen view” (step 5) is chosen, the video will resume in small screen view, otherwise if the video is individually started in fullscreen, the video will be stopped by itself and the full screen will end.

The following methods works for both default and custom overlay views.

5. **Hide overlay view:** `videoViewBuilder?.hideOverlayView()`
6. **Show overlay view:** `videoViewBuilder?.showOverlayView()`
7. **Show overlay view for a specific duration:**

   ```
   videoViewBuilder?.showOverlayViewForTime(duration)
   ```

8. **Seek-to:** `videoViewBuilder?.seekTo(position)`