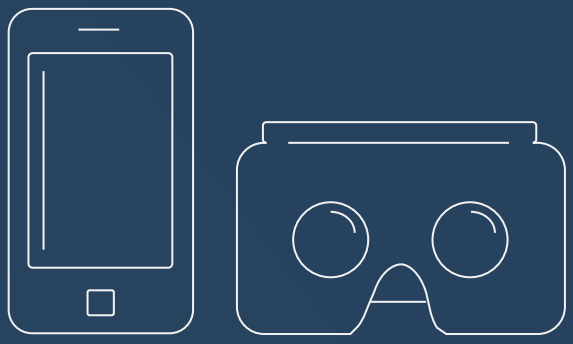
intel Sports



INTEL® IMMERSIVE EXPERIENCE SDK FOR MOBILE V1.8

User Manual Android

Table of Contents

Introduction

This document provides step-by-step directions for creating a simple Android Extension framework using the Intel® Immersive Experience SDK.

For platforms other than Windows, the steps outlined below should apply to Android Studio with minimal changes and should be broadly applicable to other IDEs.

Requirements

Minimum Android 4.4 Kit Kat (API Level 19), Android Studio

Step 1: Setup the Android Project

In this step you will setup the new project, SampleExtension (AAR file).

#	Step	Note
1.	Open **Android Studio**	
2.	Start a **new project**	
3.	Give project a name, say “SampleExtension”	
4.	Select **target** as **Phone and Tablet**	Set these parameters under **Target Android Devices** screen
5.	Set **Minimum SDK** to **Android 4.4**	
6.	Create the **project with No Activity**	

Now **convert the created project into a library**:

7.	Open the module-level **build.gradle** file	
8.	Delete the line for the **applicationId** in the build.gradle file	

9.	Change **apply plugin: 'com.android.application'** to: **apply plugin: 'com.android.library'**	
10.	Save the file	
11.	Press **File** > **Sync Project with Gradle Files**	This project now operates as an Android library and the build will now create an AAR file instead of an APK.

Step 2: Add IntelSportsExtension AAR File

#	Step

1.	Add the following lines in the **root build.gradle** file:
	
	`allprojects {`
	`  repositories {`
	`  jcenter()`
	`  maven{`
	`  url "https://maven.google.com"`
	`  }`
	`  flatDir {`
	`  dirs 'libs'`
	`  }`
	`}`
	`}`

2.	Create a **libs folder** under your app folder, if not present
3.	Browse and copy the provided **IntelSportsExtension.aar** file inside Libs folder to the **libs folder** of **SampleExtension**

Step 3: Update the Dependencies

#	Step	Note

1.	Add the following lines in the **app/build.gradle** file:
	
	`dependencies {`
	`implementation fileTree(dir: 'libs', include: ['*.jar'])`
	`implementation(name: 'IntelSportsExtension', ext: 'aar')`
	`implementation 'com.android.support:appcompat-v7:26.1.0'`
	`implementation 'com.airbnb.android:lottie:2.3.0'`
	`}`
		Update dependencies section.
		
		For older versions of gradle use compile instead of implementation.

2.	Review the below classes provided by IntelSportsExtension which can be extended and customized in your app:
	
	`AssetClient`
	`TokenizationClient`
	`ScoreboardClient`
	`LeaderboardClient`
		You can either make these custom classes in your app and use it or make a framework which contains these classes, so you can refer that library in your app.

Step 4: Create Class Extending from AssetClient

For customizing the below items we can create a custom class that extends from AssetClient:

- Background scene image in cardboard mode
- Background color and animation of choose your experience screen as well as cardboard transition screen
- The panoramic and experience logo

#	Step
1.	Create a **CustomAssetClient** class which extends AssetClient: `public class CustomAssetClient extends AssetClient {`

```
2. }
```

2. Add the following lines to **import the below packages**:

```
1. import com.intelsports.extension.AssetClient;
2. import com.intelsports.extension.LogoAsset;
3. import com.intelsports.extension.ConsumptionScene;
4. import com.intelsports.extension.ExperienceScene;
5. import android.os.Parcel;
6. import com.intelsports.extension.VideoType;
```

3. Add the following lines to **implement Parcelable** interface for this class:

```
1. @Override
2. public int describeContents() {
3. return 0;
4. }
5. @Override
6. public void writeToParcel(Parcel dest, int flags) {
7. }
8. public CustomAssetClient() {
9. }
10. protected CustomAssetClient(Parcel in) {
11. }
12. public
    static final Creator<CustomAssetClient> CREATOR = new Creator<CustomAss
    etClient>() {
13. @Override
14. public CustomAssetClient createFromParcel(Parcel source) {
15. return new CustomAssetClient(source);
16. }
17. @Override
18. public CustomAssetClient[] newArray(int size) {
19. return new CustomAssetClient[size];
20. }
21. };
```

In this class you can **override the below mentioned functions**:

4. **getConsumptionAssets()**

   This function is used to customize the background image of the cardboard mode.

You can create an instance of **ConsumptionScene** class that is available in IntelSportsExtension.
For this consumption scene object we can specify:

- The background image that need to be set
- The scene name in which the background has to be loaded
- The type of background (2D or Top Bottom)

Note: The background image should keep in `drawable-nodpi` as it is needed only in VR.

Below is an example which shows how these properties are to be set:

```
1. @Override
2. public ConsumptionScene getConsumptionAssets(Context context) {
3. ConsumptionScene consumptionScene = new ConsumptionScene();
4. sceneImage = BitmapFactory.decodeResource(context.getResources(),R.drawabl
   e.consumption_scene);
5. destinationScene = "Scene_Sphere";
6. videoType = VideoType.VIDEO_TB;
7. return consumptionScene;
8. }
```

5. **getLogoAsset()**

This function is used to **customize the panoramic and experience logo**.

Inside this function you can create an instance of LogoAssets class that is available in IntelSportsExtension. For this Logo Assets you can specify the panoramic and cardboard logo.

Note: The logo images should be in your **drawable** folder.

Below is an example which shows how these properties are to be set:

```
1. @Override
2. public LogoAsset getLogoAsset() {
3. LogoAsset logoAsset = new LogoAsset();
4. panoramicLogo = R.drawable.panoramic_logo;
5. experienceLogo = R.drawable.experience_logo;
6. return logoAsset;
7. }
```

6. **getExperienceSceneAssets()**

This function is used to **customize the Choose your Experience screen animation and background color**.

	Make an instance of ExperienceScene class that is available in IntelSportsExtension. For this experience scene you can assign any animation/view to **cardboardView**, **panoramicView** and **centerAnimation**. Below is an example which sets an image view for Choose Your Experience screen:

```
1. @Override
2. public ExperienceScene getExperienceSceneAssets(Context context) {
3. ExperienceScene experienceScene = new ExperienceScene();
4. ImageView cardboardView  = new ImageView(context);
5. setImageResource(R.drawable.cardboardView);
6. ImageView centerAnimation  = new ImageView(context);
7. setImageResource(R.drawable.centerAnimation);
8. ImageView panoramicView  = new ImageView(context);
9. setImageResource(R.drawable.panoramicView);
10.   cardboardView = cardboardView;
11.   centerAnimation = centerAnimation;
12.   panoramicView = panoramicView;
13.   return experienceScene;
14.   }
```

7.	Assign the **background color** inside the overridden getExperienceSceneAssets function (in the class that is extended from AssetClient). The color should be defined in your library.

```
1. @Override
2. public ExperienceScene getExperienceSceneAssets(Context context) {
3. ExperienceScene experienceScene = new ExperienceScene();
4. bgSceneColor = R.color.experience_scene_color;
5. return experienceScene;
6. }
```

Step 5: Create Class Extending from ScoreboardClient

For customizing Scoreboard in panoramic and cardboard video modes in the SDK, create a class which extends ScoreboardClient of IntelSportsExtension class.

#	Step
1.	Create a '**CustomScoreboardClient**' class which extends ScoreboardClient:

```
1. public class CustomScoreboardClient extends ScoreboardClient {
2. }
```

2. Add the following lines to **import the below packages**:

```
1. import com.intelsports.extension.ScoreboardClient;
2. import com.intelsports.extension.ScoreboardClientInterface;
```

3. Add the following lines to **implement Parcelable** interface for this class:

```
1. @Override
2. public int describeContents() {
3. return 0;
4. }
5. @Override
6. public void writeToParcel(Parcel dest, int flags) {
7. }
8. public CustomScoreboardClient() {
9. }
10.    protected CustomScoreboardClient(Parcel in) {
11.    }
12.    public
   static final Creator<CustomScoreboardClient> CREATOR = new Creator<Custo
   mScoreboardClient>() {
13.    @Override
14.    public CustomScoreboardClient createFromParcel(Parcel source) {
15.    return new CustomScoreboardClient(source);
16.    }
17.    @Override
18.    public CustomScoreboardClient[] newArray(int size) {
19.    return new CustomScoreboardClient[size];
20.    }
21.    };
```

In this class you can **override the below mentioned functions** to create a custom scoreboard in 2D and VR.

4. **setID3Tag()**

   This function will get called when there will be a metadata coming from the video when plays. The function contains an interface as argument called `ScoreboardClientInterface`. Use `onScoreboardUpdated` function (part of this interface), which contains a bitmap and a string as argument for the VR scoreboard.

   You can:
   - Pass **bitmap image** of scoreboard and the scene name in which the bitmap has to be updated

	○ Parse id3metadata **string** for making scoreboard You can **create your own scoreboard** inside this function.

	`1. void onScoreboardUpdated(Bitmap var1, String var2);`
	`2. @Override`
	`3. public void setID3Tag(Context context, String id3metaData,`
	`4. ScoreboardClientInterface scoreboardClientInterface) {`
	`5. }`

5.	**get2DScoreboardView()** This function is used to **create a scoreboard for 2D**. A scoreboard display view can be returned and added on top of the video view in full screen.

	`1. @Override`
	`2. public RelativeLayout get2DScoreboardView(final Context context) {`
	`3. }`

Apart from the previously mentioned functions there are some methods available for **overriding in the Custom class** from **ScoreboardClient** which can be used for further customization:

	enableScoreboard() This function is used to **determine if scoreboard has to be enabled or not.** Passing **false value** to this function can **disable** the scoreboard. As SDK support videoPlaylist we can pass a list of flags which enable/disable the scoreboard for the list.

	`1. @Override`
	`2. public void enableScoreboard(List<Boolean> isEnable) {`
	`3. }`

	activateClientTopLayer() This function enables to **activate the scoreboardboard** view that has been deactivated.

	`1. @Override`
	`2. public void activateClientTopLayer() {`
	`3. }`

	deactivateClientTopLayer()

	This function enables to **deactivate the scoreboardboard** view.
	1. @Override 2. public void deactivateClientTopLayer() { 3. }

Step 6: Create Class Extending from LeaderboardClient

For customizing Leaderboard in SDK video, create an extension class from LeaderboardClient.

#	Step
1.	Create a '**CustomLeaderboardClient**' class which extends LeaderboardClient: 1. public class CustomLeaderboardClient extends LeaderboardClient { 2. }
2.	Add the following lines to **import the below packages**: 1. import com.intelsports.extension.LeaderboardClient; 2. import com.intelsports.extension.LeaderboardClientInterface;

3.	Add the following lines to **implement Parcelable** interface for this class:
	3. @Override
	4. public int describeContents() {
	5. return 0;
	6. }
	7. @Override
	8. public void writeToParcel(Parcel dest, int flags) {
	9. }
	10. public CustomLeaderboardClient() {
	11. }
	12. protected CustomLeaderboardClient(Parcel in) {
	13. }
	14. public static final Creator<CustomLeaderboardClient> CREATOR = new Creator<CustomLeaderboardClient>() {
	15. @Override
	16. public CustomLeaderboardClient createFromParcel(Parcel source) {
	17. return new CustomLeaderboardClient(source);
	18. }
	19. @Override

```
20.    public CustomLeaderboardClient[] newArray(int size) {
21.    return new CustomLeaderboardClient[size];
22.    }
23.    };
```

In this class you can **override the below mentioned functions** to create a custom leaderboard in 2D and VR.

4.	**setID3Tag()** This function will get called when there will be a metadata coming from the video when plays. It contains a `LeaderboardClientInterface` as argument. Use `onLeaderboardUpdated` function of this interface, which contains a bitmap and a string as argument for the VR leaderboard. You can: o Pass **bitmap image** of leaderboard and the scene name in which the bitmap has to be updated o Parse id3metadata **string** for making leaderboard You can **create your own leaderboard** inside this function.

```
1. void onLeaderboardUpdated(Bitmap var1, String var2);

2. @Override
3. public void setID3Tag(String id3metaData, LeaderboardClientInterface
                         leaderboardClientInterface) {


4. }
```

5.	**get2DLeaderboardView()** This function is used to **create a leaderboard for 2D**. A scoreboard display view can be returned and added on top of the video view in full screen.

```
1. @Override
2. public
   RelativeLayout get2DLeaderboardView(final Context context) {
3. }
```

Apart from the previously mentioned functions there are some methods available for **overriding in the Custom class** from **LeaderboardClient** which can be used for further customization:

6.	**enableLeaderboard ()**

	This function is used to **determine if leaderboard has to be enabled or not**. Passing false value to this function can disable the leaderboard. As SDK supports a video playlist we can pass a list of flags which enable/disable the leaderboard for the list.
	1. @Override 2. **public void enableLeaderboard**(List<Boolean> isEnable) { 3. }
7.	**startLeaderboardFetch()** It can be used to set a **timer which can be started in this function for determining the time interval for leaderboard to be shown**.
	1. Override 2. **public void startLeaderboardFetch**(Context context, **int** videoInfoInde x) { 3. }
8.	**stopLeaderboardFetch()** This function is used to **stop a timer started during startLeaderboardFetch.**
	1. @Override 2. **public void stopLeaderboardFetch**() { 3. }
9.	**activateClientTopLayer()** It is possible to **activate the leaderboard view that has been deactivated**.
	1. @Override 2. **public void activateClientTopLayer**() { 3. }
10.	**deactivateClientTopLayer()** It is possible to **deactivate the leaderboard view** returned.
	1. @Override 2. **public void deactivateClientTopLayer**() {

```
3. }
```

Step 7: Create Class Extending from TokenisationClient

You can tokenize the URL in SDK. The SDK supports AKAMAI tokenization. To set this tokenization create an extension class from **TokenisationClient** of IntelSportsExtension class.

#	Step

1.	Create a '**CustomTokenizationClient**' class which extends TokenizationClient:
	
	`1. public class CustomTokenizationClient extends TokenizationClient {`
	`2. }`

2.	Add the following line to **import the below package**: `1. import com.intelsports.extension.TokenizationClient;`

3.	Add the following lines to **implement Parcelable** interface for this class:
	
	`1. @Override`
	`2. public int describeContents() {`
	`3. return 0;`
	`4. }`
	`5. @Override`
	`6. public void writeToParcel(Parcel dest, int flags) {`
	`7. }`
	`8. public CustomTokenizationClient() {`
	`9. }`
	`10. protected CustomTokenizationClient(Parcel in) {`
	`11. }`
	`12. public static final Creator<CustomTokenizationClient> CREATOR = new Parcelable.Creator< CustomTokenizationClient >() {`
	`13. @Override`
	`14. public CustomTokenizationClient createFromParcel(Parcel source) {`
	`15. return new CustomTokenizationClient(source);`
	`16. }`
	`17. @Override`
	`18. public CustomTokenizationClient[] newArray(int size) {`
	`19. return new CustomTokenizationClient[size];`
	`20. }`
	`21. };`

4.	Override the function **getTokenizedUrl()** inside this class:
	
	`1. @Override`
	`2. public String getTokenizedUrl(UrlTokenType tokenType, String url) {`
	`3. }`

	Inside this function we can generate a tokenized URL and return it.

Step 8: Add ProGuard Exclusions

At this stage you can **add the following script lines** to the **proguard-rules.pro** file of SampleExtension.

#	Step	Note

1.	Set **minifyEnabled** to **true** in **app/build.gradle:**
	
	`1. buildTypes {`
	`2. release {`
	`3. minifyEnabled true`
	`4. proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'`
	`5. }`
	`6. }`
		This is done to enable the ProGuard.

2.	Set the **ProGuard rules**, is an example:
	
	`1. -keepclassmembers class* implements os.Parcelable {`
	`2. static ** CREATOR;`
	`3. }`
	`4. -keep class intelsports.sampleextension.CustomAssetClient {`
	`5. public *;`
	`6. }`
	`7. -keep class intelsports.sampleextension.CustomLeaderboardClient {`
	
	`8. public *;`
	`9. }`
	`10. -keep class intelsports.sampleextension.CustomScoreboardClient {`
	
	`11. public *;`
	`12. }`	

	13. -keep class intelsports.sampleextension.CustomTokenizationClient { 14. public *; 15. }	

Step 9: Build SampleExtension Library to AAR

At this stage you will build this library to aar.

#	Step	Note
1.	Select the **assembleRelease** from Gradle window	
2.	Then the output will generate the following path: **app/build/outputs/aar**	

Step 10: Use SampleExtension in Your App

For this copy the app-release.aar file is generated in app/build/outputs/aar folder of SampleExtension to the libs folder of your application.

#	Step	Note
1.	Add the **SampleExtension.aar** in your project: `1. implementation (name: 'app-release', ext: 'aar')`	Use `compile` instead of `implementation` for the older versions of gradle.