

User Guide

TouchSense[®] SDK for Mobile Apps

Version 2.2 | July 2018

Table of Contents

| | | |
|-----|---|----|
| 1. | Scope..... | 1 |
| 1.1 | Package Structure..... | 2 |
| 1.2 | Minimum System Requirements | 3 |
| 2. | Getting Started | 4 |
| 2.1 | Setup TS SDK in a Java application..... | 4 |
| 2.2 | Setup TS SDK in a native application..... | 4 |
| 2.3 | Setup TS SDK in a Unity application | 6 |
| 2.4 | Setup TS SDK in a Cocos2d-x Android application | 6 |
| 2.5 | Running the Sample Applications | 7 |
| 3. | Using TS SDK..... | 11 |
| 3.1 | Initializing the Haptic Media Player | 11 |
| 3.2 | Adding Haptic Resources..... | 13 |
| 3.3 | Playing Haptic Effects..... | 14 |
| 3.4 | Additional playback options | 15 |
| 3.5 | Synchronization | 17 |
| 3.6 | Clean Up | 20 |
| 3.7 | ProGuard | 20 |
| 3.8 | Additional SDK Information..... | 20 |
| 4. | API Library | 22 |
| 5. | Troubleshooting | 23 |
| 5.1 | Error Codes..... | 23 |
| 5.2 | Return Codes..... | 24 |
| 5.3 | Pitfalls and Recovery Steps..... | 25 |
| 6. | FAQs..... | 26 |
| 7. | Additional Support Information | 28 |
| 7.1 | Support & Services | 28 |
| 7.2 | License Expiry | 28 |
| 7.3 | Supporting Documentation..... | 28 |
| 8. | About Immersion..... | 29 |

1. Scope

Tactile effects create more immersive and personal experiences for end users across a variety of mobile applications. The TouchSense® Software Development Kit for Mobile Apps (TS SDK) enables developers to bring these experiences to their Android applications. For example, TS SDK enables:

- Game developers to enhance their mobile gameplay with interactive tactile effects.
- Multimedia applications to synchronize tactile effects with media content (e.g. video)

This document explains how to integrate TS SDK in an application and how to start using it to render haptic effects.

Document Conventions

Screenshots in this document are for illustrative purposes only. They may not correspond exactly to what you see in your working environment.

[Table 1](#) lists typographical conventions adhered to in this user guide.

Table 1: Typographical Conventions

| Element | Description |
|--|--|
| Literals GUI elements GUI titles and commands | Boldface |
| Keystrokes and keyboard input | Boldface Keystrokes are shown ALL CAPS. Combinations of keys that must be pressed simultaneously are shown with a plus sign (+) linking the keys, for example, CTRL+J . |
| Code display Keyboard input display | Regular Courier font in gray block |
| Output display | Regular Courier font |
| User-supplied values Cross references Publication titles | <i>Italics</i> |
| GUI menu paths | For example: Run As > Android Application |

1.1 Package Structure

The SDK package contains everything you need to integrate TS SDK into your application.

[Table 2: SDK Package Content](#) lists the content of the SDK package.

Table 2: SDK Package Content

| Directory | Item | |
|-----------------------|---|--|
| Documentation | <i>TouchSenseSDK_Quick_Start_Guide.pdf</i> | Provides a brief description of the SDK, how to set it up, and how to use it in an application. |
| | <i>TouchSenseSDK_User_Guide.pdf</i> | Provides a detailed description of the SDK with all the information a developer needs to integrate the SDK in Java, native, Unity, and Cocos2d-x applications. |
| | <i>release_notes.txt</i> | Includes new features, known issues, defects, etc. |
| | <i>javadoc.zip</i> | Contains javadoc documentation for all public library classes. |
| | <i>NOTICE.txt</i> | Contains license and copyright information. |
| Haptic Effect Library | <i>HapticEffects.zip</i> | Pre-designed haptic media files for use in applications. |
| | <i>SDKPreviewer.apk</i> | Used to preview what the pre-designed effects feel like. |
| SDK/bin | The core libraries a developer would integrate to enhance their application with tactile effects. | |
| SDK/include | The C header files to be included when integrating the SDK in a native Android app. | |
| Plugins/Unity | <i>TouchSenseSDKPlugin_v2.2.x.unitypackage</i> | A Unity package used to integrate TS SDK into a Unity Android game. |

The Sample Apps package contains several Androids apps showcasing different uses of the SDK.

Table 3: Sample Apps Package Contents

| Directory | Description |
|------------------|---|
| StickerSampleApp | An Android application project used to demonstrate how to integrate and use TS SDK within an interactive context using chat stickers. |
| VideoSampleApp | An Android application project used to demonstrate how to integrate and use TS SDK to synchronize tactile effects with video media content. |
| NativeSampleApp | An Android application project used to demonstrate how to integrate and use TS SDK in native Android applications. |
| UnitySampleApp | A Unity project used to demonstrate how to integrate and use TS SDK in a Unity Android app. |

TS SDK provides a simple Java API for quick integration into Android applications. In the **bin** directory of the SDK package, there are two libraries to be placed into the Android application project:

- **TouchSenseSDK-v2.2.x.jar**

The Java application entry point, which allows the loading and playback of tactile effects.

- **libTouchSenseSDK.so**

A native engine for processing and rendering tactile effects.

Note: Clients using the Java API must include both of the above files.

1.2 Minimum System Requirements

Integration of the SDK requires:

- Android 4.0 Ice Cream Sandwich (API level 14) or newer, excluding wearables.
- Android SDK Tools Revision 23 or newer (to run the sample applications).

TS SDK is compatible with all Android devices equipped with vibration functionality. If a device is not equipped with vibration hardware and software, the SDK will not cause the device to malfunction in any way.

2. Getting Started

This section will discuss how to setup the SDK for use in Java, native, Unity, and Cocos2d-x applications. Before you begin, extract the SDK package you receive. The extracted package directory will be referred to as **SDK_PKG_PATH**, where applicable.

2.1 Setup TS SDK in a Java application

In Android Studio, at the root of your application project, open the **app** directory. Inside the **app** directory, make sure you have a **libs** directory.

```
projectRootFolder/
├── app
│   └── libs
│       ├── TouchSenseSDK-vX.Y.Z.jar
│       └── armeabi
│           └── libTouchSenseSDK.so
```

1. Create or locate the **libs** directory at the root of your project.
2. Place the **SDK_PKG_PATH/bin/libs/TouchSenseSDK-v2.2.x.jar** file in the **libs** directory.
3. Create or locate the subdirectory **armeabi** inside the **libs** directory.
4. Place the **SDK_PKG_PATH/bin/libs/armeabi/libTouchSenseSDK.so** file in the **armeabi** directory.

Important: This step is critical. Make sure the directory is correctly named and the file correctly placed in the **armeabi** directory.

5. Add the following permissions to the **AndroidManifest.xml** file:

```
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.INTERNET" />
```

If you choose to store your haptic effect files on the Secure Digital (SD) card, add the following permissions to **AndroidManifest.xml** as well:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

6. Once the steps above are complete, you will need to modify your Gradle build file, **build.gradle**, to include the SDK libraries as dependencies.

Add the following code to your **build.gradle** file:

```
android.sourceSets.main.jniLibs.srcDirs = ['libs']
dependencies { compile fileTree(dir: 'libs', include: 'TouchSenseSDK*.jar') }
```

At this point, you should be able to start using the SDK's libraries from within your application.

2.2 Setup TS SDK in a native application

In Android Studio, at the root of your application project, open the **app** directory. Inside the **app** directory, make sure you have a **libs** directory.

```
projectRootFolder/
├── app
│   └── libs
│       └── armeabi
│           └── libTouchSenseSDK.so
```

1. Create or locate the **libs** directory at the root of your project.
2. Create or locate the subdirectory **armeabi** inside the **libs** directory.
3. Place the **SDK_PKG_PATH/bin/libs/armeabi/libTouchSenseSDK.so** file in the **armeabi** directory.

Important: This step is critical. Make sure the directory is correctly named and the file correctly placed in the **armeabi** directory.

4. Add the following permissions to the **AndroidManifest.xml** file:

```
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.INTERNET" />
```

If you choose to store your haptic effect files on the Secure Digital (SD) card, add the following permissions to **AndroidManifest.xml** as well:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

5. Once the steps above are complete, you will need to modify your top-level Gradle build file, **build.gradle**, to use **gradle-experimental** plugin which is required for NDK apps.

Change the following dependencies in your top-level **build.gradle** file:

```
dependencies { classpath 'com.android.tools.build:gradle-experimental:0.7.2' }
```

6. Now you will need to modify the individual module **build.gradle** file.
Add the SDK native binary as a prebuilt library in the module's repositories list.

```
repositories {
    libs(PrebuiltLibraries) {
        TSSDKprebuilt {
            headers.srcDir "src/main/jni"
            binaries.withType(SharedLibraryBinary) {
                sharedLibraryFile = file("libs/armeabi/libTouchSenseSDK.so")
            }
        }
    }
}
```

Add an **abiFilter** to the **android.ndk** instruction set.

```
android {
    ...
    ndk {
        ...
        abiFilters.add("armeabi")
    }
}
```

Finally, add the following instructions to the **sources.main** section.

```
sources {
    main {
        jni {
            dependencies { library 'TSSDKprebuilt' linkage 'shared' }
        }
        jniLibs {
            source { srcDir "libs" }
        }
    }
}
```

At this point, you should be able to start using the SDK's libraries from within your application.

For more information on using Android NDK with the gradle-experimental plugin, please see the links below:

- <http://tools.android.com/tech-docs/android-ndk-preview>
- <http://tools.android.com/tech-docs/new-build-system/gradle-experimental>

2.3 Setup TS SDK in a Unity application

1. Import the SDK Unity package into your project.
 - Select **Assets > Import Package > Custom Package...**
 - Navigate to **SDK_PKG_PATH/Plugins/Unity** folder
 - Select **TouchSenseSDKPlugin-v2.2.x.unitypackage**.
Unity 5.x.x: Manually select the Android platform for the plugin.
 - Navigate to **Assets/Plugins/Android/libs/armeabi/libTouchSenseSDK.so**
 - In the Inspector tab under **Select platform for plugin**, check **Android** and press **Apply**.
2. Locate and update your **AndroidManifest.xml** to include the **VIBRATE** and **INTERNET** permissions.

```
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.INTERNET" />
```

The default **AndroidManifest.xml** auto-generated by Unity can be found in your Unity project folder under **Temp/StagingArea/**. Once modified, place the **AndroidManifest.xml** file in the **Assets/Plugins/Android/** folder.

3. Place any haptic resource files (*.hapt) you wish to use in the **Assets/StreamingAssets** folder.
(Find pre-designed tactile effects in the **SDK_PKG_PATH/Haptic Effect Library/** folder.)

2.4 Setup TS SDK in a Cocos2d-x Android application

1. Create a directory under **proj.android/jni** named **TSSDK**.
2. Copy the TouchSense SDK library (**SDK_PKG_PATH/bin/libs/armeabi/libTouchSenseSDK.so**) and headers (**SDK_PKG_PATH/include/*.h**) to the directory created in step 1. Below is a generic Cocos2d-x project structure with the necessary TS SDK files.

```
.
├── CMakeLists.txt
├── Classes
├── Resources
├── proj.android
│   ├── AndroidManifest.xml
│   ├── ant.properties
│   ├── assets
│   ├── bin
│   ├── build-cfg.json
│   ├── build.xml
│   ├── jni
│   │   ├── Android.mk
│   │   ├── Application.mk
│   │   ├── TSSDK
│   │   │   ├── c_tssdk_error_codes.h
│   │   │   ├── c_tssdk_info_codes.h
│   │   │   ├── TouchSenseSDK.h
│   │   │   └── libTouchSenseSDK.so
│   ├── hellocpp
│   │   └── main.cpp
│   ├── libs
│   ├── local.properties
│   ├── obj
│   ├── proguard-project.txt
│   ├── project.properties
│   └── res
```



```
├── src
│   ├── proj.ios_mac
│   ├── proj.linux
│   ├── proj.win32
│   ├── proj.win8.1-universal
│   └── proj.win10
```

3. Modify the `Android.mk` file to include the prebuilt shared library.

```
include $(CLEAR_VARS)
LOCAL_MODULE      := TSSDK
LOCAL_SRC_FILES   := TSSDK/libTouchSenseSDK.so
include $(PREBUILT_SHARED_LIBRARY)
```

4. Within the module that contains your Cocos2d-x source code include the TSSDK module you just created and the path to the TSSDK header files.

```
LOCAL_SHARED_LIBRARIES := TSSDK
LOCAL_C_INCLUDES += $(LOCAL_PATH)/TSSDK
```

5. Add the following permissions to the `AndroidManifest.xml` file found under the `proj.android` directory.

```
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.INTERNET" />
```

6. To initialize and play haptic effects from your Cocos2d-x application, please refer to the 'Native' code snippets outlined throughout Section [Using TS SDK3](#).

2.5 Running the Sample Applications

TS SDK includes sample applications that demonstrate the process of integrating the SDK into Android applications. The sample applications are contained in a compressed file separate (**SampleApps.zip**) from the SDK package. To use these, extract the source code from this file and import each into your IDE.

The **Video Sample App** contains a short sample video enhanced with tactile effects. When you launch the application, the short sample video plays along with a synchronized haptic effect.

The **Sticker Sample App** contains a dashboard of stickers, each enhanced with tactile effects. When you select a sticker, it animates and triggers the corresponding haptic effect.

The **Native Sample App** contains single button enhanced with a tactile effect. When you click the button, it triggers a short haptic effect.

The **Unity Sample App** contains four buttons, two of which are enhanced with tactile effects. After you click the "Add Resources" button, you can play each of the effects by clicking the corresponding button.

To import and run any of the *Java* sample applications:

1. Open Android Studio.
2. Select **File > New > Import project...**
3. Navigate to the **VideoSampleApp**, **StickerSampleApp** or **NativeSampleApp** folder and select the **build.gradle** file.
4. Click **OK** to import the project.

To open and run the *Unity* sample application:

1. Open Unity.
2. Select **File > Open Project**.
3. Navigate to the **UnitySampleApp** folder and select it.
4. Click **Open**.
Unity 5.x.x: Approve upgrading your project to Unity 5.x.x setup.

To successfully run the sample applications with correct TS SDK functionality, you *must* provide the username and password given to you by Immersion when invoking the `create` method.

```
create(Context context, String username, String password)
```

If you were provided with a specific DNS address, apply the DNS URL information using:

```
create(Context context, String username, String password, String dnsURL)
```

The username and password can be hardcoded in the sample application to make it run. However, in your own application, Immersion recommends storing the credentials on a server and fetching them at runtime.

Modify the following lines of code before you can run the sample apps successfully:

For **VideoSampleApp** and **StickerSampleApp**, inside **MediaActivity.java** and **HapticManager.java**, respectively, update:

```
private static final String USERNAME = null;
private static final String PASSWORD = null;
private static final String DNS = null;
```

For **NativeSampleApp** inside **native-sample-app.c** update:

```
const char* USERNAME = null;
const char* PASSWORD = null;
const char* DNS = null;
```

For **UnitySampleApp** inside **Controller.cs** update:

```
private string USERNAME = null;
private string PASSWORD = null;
private string DNS = null;
```

Substitute your credentials for `USERNAME` and `PASSWORD` to retrieve a valid instance. If you were provided with a specific DNS address, use that. Otherwise, keep `DNS` as `null`.

Troubleshooting

Depending on the version of Android Studio you use, you may receive an error message similar to the following when you import the application:

```
Error:Gradle version 1.10 is required. Current version is 2.2.1. If using the
gradle wrapper, try editing the distributionUrl in ...
```

```
The minimum supported version of the Android Gradle plugin is 1.0.0, but the
project is using 0.9.0.
```

This error occurs because the Gradle plugin used in the sample app is not compatible with your Gradle version. To achieve compatibility, change the version of the Gradle plugin in the sample application's Gradle build file. Specifically, change the gradle version in the following lines:

```
dependencies { classpath 'com.android.tools.build:gradle:2.1.2' }
```

Then, simply rebuild the project for a successful compilation. For more information on the compatibility list between the Gradle plugin in Android Studio and Gradle itself, please refer to this link: <http://tools.android.com/tech-docs/new-build-system/version-compatibility>.

During the installation of the **NativeSampleApp** on older Android versions you might encounter the following error:

```
java.lang.UnsatisfiedLinkError: Cannot load library: ... could not load library
"libTouchSenseSDK.so" needed by "libnative-sample-app.so";
```

This error occurs due to incorrect dynamic loading of dependent shared libraries. To solve this issue, simply uncomment the following line in the static constructor of **MainActivity.java** file:

```
System.loadLibrary("TouchSenseSDK");
```

Running a Sample Application with Android Studio

After you build your project successfully, the app shows as available under the run configuration list.

To run the app:

1. Click on the **Play/Run** button.
2. From the **Choose Device** window, select the device on which you want the app to run, as shown in [Figure](#).
3. Click **OK**.

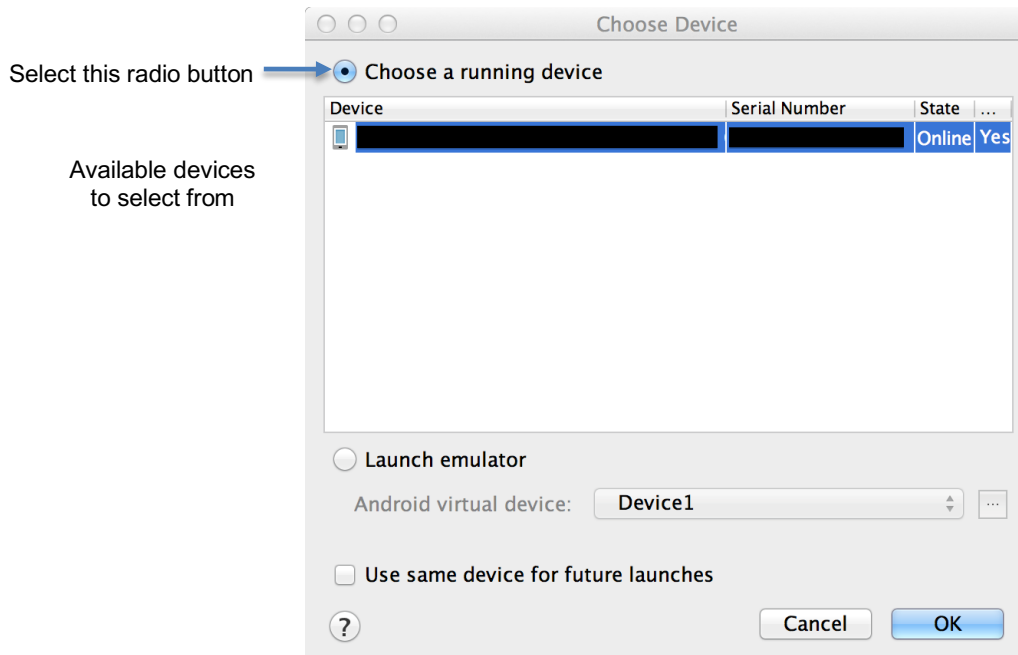


Figure 1: Using Android Studio to Select a Device

The app should now run on your Android device. You can view messages logged by the SDK in the Android Studio **Android Monitor** window, such as:

- **Errors in red**
- **Warnings in blue**
- **Information in green**

2.5.1 Running a Sample Application with Unity

After opening the Unity Sample App project update the build settings.

Set the application build to Android platform:

- Open **Build Settings**.
 - **File > Build Settings** or **CTRL+Shift+B**.
- Select **Android Platform**.
- Click **Switch Platform**.

- Click **Build And Run**
 - **File > Build & Run** or **CTRL+B**.

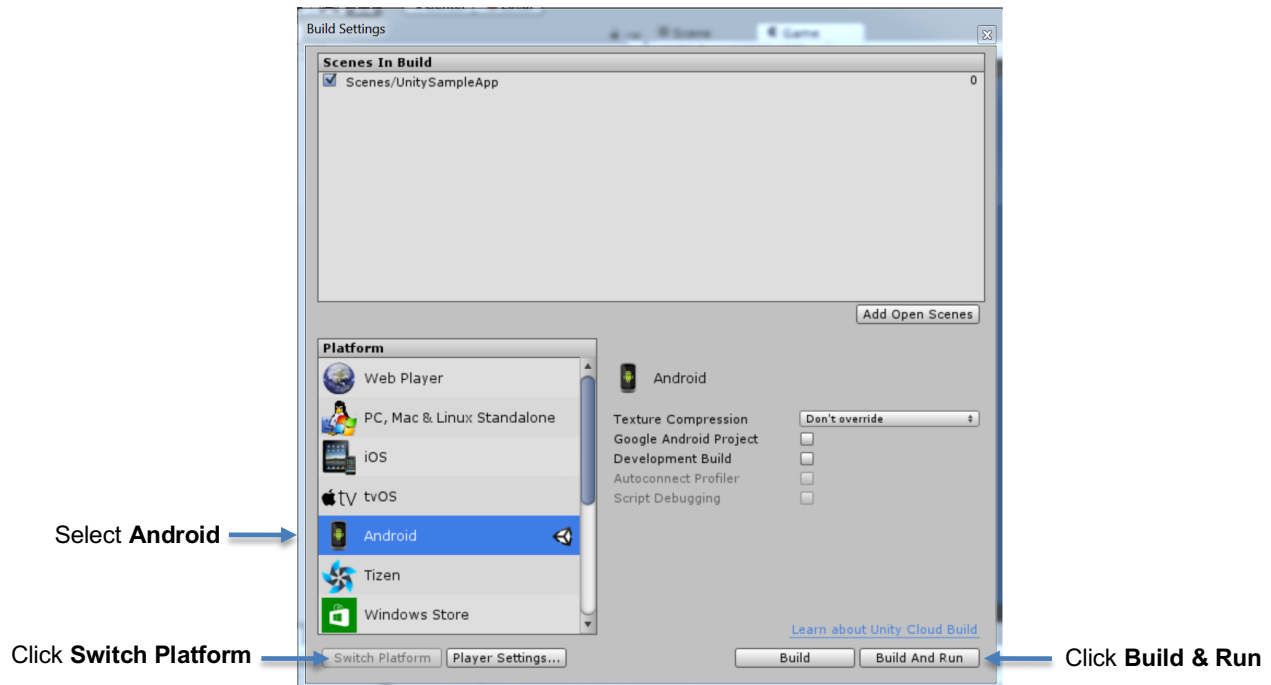


Figure 2: Using Unity to Switch Platform to Android and Run the application

The app should run on the attached Android device. To view SDK messages, you need to connect the device via Android Debug Bridge and monitor the **logcat** messages.

3. Using TS SDK

Now that you've integrated the SDK libraries into your project, you can start enriching your application's user experience with tactile effects. Before getting started, it is important to understand how the SDK fundamentally interacts with an application. This is described in the diagram below:

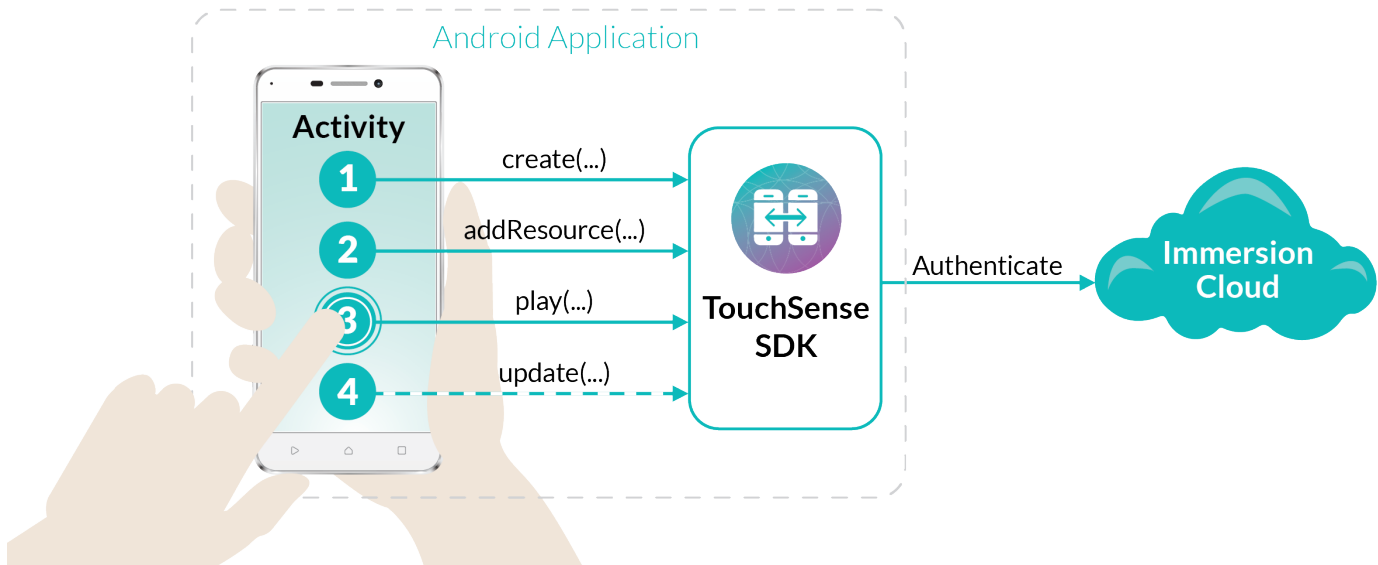


Figure 3: Integrating TS SDK into an Android Application

This section describes the full playback cycle shown in the diagram above, as well as most of the API methods that are made available. For a complete list of the API library's methods, see 4 [API Library](#). More detailed code examples can also be viewed in the sample applications provided as explained in section 2.5 [Running the Sample Applications](#).

3.1 Initializing the Haptic Media Player

The **HapticMediaPlayer** class allows an activity to play haptic effects. In order for it to work you must initialize it properly.

JAVA

To create an instance of **HapticMediaPlayer**, import the appropriate classes and create a class member.

```
// Add imports
import com.immersion.touchsensesdk.HapticMediaPlayer;
...
// Create a class member
private HapticMediaPlayer mHapticMediaPlayer;
```

Now create an instance using the `create` method. This method requires an Android Context object such as the current Activity, as well as valid username, password, and an optional DNS URL. The username, password, and DNS URL (if applicable) parameters are provided by Immersion.

```
// Pass in the username and password provided to you by Immersion
// Pass in the preferred DNS URL if it was provided to you by Immersion or
// use HapticMediaPlayer.create(context, username, password) instead
try {
    mHapticMediaPlayer = HapticMediaPlayer.create(context, username,
                                                    password, dns);
    int playerState = mHapticMediaPlayer.getPlayerInfo(
        HapticMediaPlayer.PlayerInfo.PLAYER_STATE);
}
```

```

        if (playerState != HapticMediaPlayer.PlayerState.INITIALIZED) {
            // Error creating player
        }
    } catch (IllegalStateException e) {
        // Thrown when the version of the .jar and .so do not match
    }

```

Note that the SDK validates the .jar and .so versions and throws an `IllegalStateException` if they do not match. Always update both .jar and .so when upgrading to a newer version of the SDK.

NATIVE

In order to create a native Haptic Media Player, include the appropriate header files, which can be found under **SDK/include** directory in the SDK package.

```

// Add includes
#include "TouchSenseSDK.h"
#include "c_tssdk_error_codes.h"
#include "c_tssdk_info_codes.h"

```

Now create an instance using the `touchsensesdk_create` function. This function requires a JavaVM object, an Android Context object as well as valid username, password, and an optional DNS URL. These parameters are provided by Immersion.

The JavaVM object can be obtained using the `JNI_OnLoad` callback function:

```

static JavaVM *g_javavm = NULL;
jint JNI_OnLoad(JavaVM* vm, void* reserved) {
    g_javavm = vm;
    return JNI_VERSION_1_6;
}

```

The Context object should be sent from the Java application to the native API and passed on to the `touchsensesdk_create` function.

```

void* player=touchsensesdk_create(g_javavm, &context, USERNAME, PASSWORD, DNS);
if (player != NULL) {
    int playerState = touchsensesdk_getPlayerInfo(player, TSSDK_PLAYER_STATE);
    if (playerState != TSSDK_INITIALIZED) {
        // Error creating player
    }
} else {
    //player memory could not be allocated
}

```

UNITY

The TS SDK Unity plugin contains a **HapticsManager** prefab under **Assets/Immersion/Prefabs**. This is a singleton class that manages haptic playback. The class encapsulates the **HapticMediaPlayer**, exposes some of its API and provides other important utility functions.

To obtain an instance of **HapticsManager** object simply drag it to your first Scene.

In a C# script, initialize the **HapticsManager** using its static `Init` function. This method requires valid username, password, and an optional DNS URL. These parameters are provided by Immersion.

```

// Add using
using com.immersion.touchsensesdk;
...
HapticsManager.Init(USERNAME, PASSWORD, DNS);

```

```
int playerState = HapticsManager.GetPlayerInfo(
    HapticMediaPlayer.PlayerInfo.PLAYER_STATE);
if (playerState != (int)HapticMediaPlayer.PlayerState.INITIALIZED) {
    // Error creating player
}
```

It is good practice to check the state of the player after creating it. The state must be `INITIALIZED` for the player to function. More information on the player's states is found in the [Additional SDK Information](#) section.

3.2 Adding Haptic Resources

As previously mentioned, the **HapticMediaPlayer** plays haptic effect files to produce tactile sensations. Like other media formats, haptic effect files have their own unique file format, ending in the **.hapt** extension. The haptic file contains the haptic effect data that only TS SDK can interpret to play the corresponding effect across various Android devices.

The **Haptic Effect Library** directory in your SDK package includes several pre-designed **.hapt** files for your use, each representing a unique tactile effect. It also contains the SDK Previewer APK which allows you to preview all of the effects before integrating them into your application. To get started immediately, this section will discuss the background and steps required to integrate these pre-designed assets into your application's user experience.

Each such file is referred to as a haptic *resource*. Haptic resources may reside as part of your application's assets, locally on the device, or be downloaded at runtime by the SDK from a remote URL. Haptic resources must be added to the SDK using the `addResource` method before you can play them. This method takes the URI of the **.hapt** file, and an additional parameter that specifies whether the resource playback should be synchronized with a time source (`SYNC_HAPTIC_EFFECT`) or not (`ASYNC_HAPTIC_EFFECT`). The difference between the synchronous and asynchronous effects is outlined in section 3.5. Each haptic resource successfully added to the player using the `addResource` API receives a unique positive resource ID (`resourceId`) assignment, used for playback requests. It is recommended that you save each ID once a resource has been added and throughout your application's lifecycle to be able to make future playback requests.

JAVA

To add a regular (asynchronous) haptic effect:

```
int resourceId = mHapticMediaPlayer.addResource(resourceUri,
    HapticMediaPlayer.HapticEffectType.ASYNC_HAPTIC_EFFECT);
if (resourceId <= 0) {
    // Failed to add resource, lookup resourceId error code value in
    // TouchSenseSDKError static class
}
```

Alternatively, to add a synchronized haptic effect:

```
int resourceId = mHapticMediaPlayer.addResource(resourceUri,
    HapticMediaPlayer.HapticEffectType.SYNC_HAPTIC_EFFECT);
```

NATIVE

To add an asynchronous haptic effect:

```
int resourceId = touchsensesdk_addResource(player, uri,
    TSSDK_ASYNC_HAPTIC_EFFECT);
if (resourceId <= 0) {
    // Failed to add resource, lookup resourceId error code value in
    // TSSDK_Error enum (found in c_tssdk_error_codes.h)
}
```

Alternatively, to add a synchronized haptic effect:

```
int resourceId = touchsensesdk_addResource(player, uri,
                                           TSSDK_SYNC_HAPTIC_EFFECT);
```

UNITY

First, find the URI of your local resources (.hapt files you placed in the **StreamingAssets** folder) by extracting them from the compressed **StreamingAssets** file to the **PersistentDataPath**:

```
string[] resourceNames = new string[] { "Resource.hapt" };
HapticsManager.CopyStreamingAssetsToPersistentData(resourceNames);
string resourceUri = HapticsManager.GetLocalResourceURI(resourceNames[0]);
```

Once this completes, add your resources to the player optionally providing the effect type:

```
int resourceId = HapticsManager.AddResource(resourceUri,
                                           (int)HapticMediaPlayer.HapticEffectType.ASYNC_HAPTIC_EFFECT);
if (resourceId <= 0) {
    // Failed to add resource, lookup resourceId error code value in
    // TouchSenseSDKError enum (found in HapticMediaPlayer.cs) or by using
    // HapticsManager.GetErrorString(resourceId)
}
```

Alternatively, to add a synchronized haptic effect:

```
int resourceId = HapticsManager.AddResource(resourceUri,
                                           (int)HapticMediaPlayer.HapticEffectType.SYNC_HAPTIC_EFFECT);
```

3.3 Playing Haptic Effects

While a haptic resource (discussed in section 3.2) represents the actual haptic effect file, we define a haptic effect to be the rendered haptic resource. Haptic effects are created by invoking the `play` method with the associated haptic resource ID (returned from `addResource`) and optionally, a priority level. Up to 50 effects can be played concurrently.

Additionally, the `play` method returns a unique positive effect ID (`effectId`) for each haptic effect even if the same resource is played multiple times. The haptic effect's lifecycle begins with the `play` method and ends either when the effect reaches the end, or when the `stop` method is called. Once the haptic effect is finished its `effectId` is no longer valid and will result in an `INVALID_PARAMETER` return code for subsequent API calls.

JAVA

```
int effectId = mHapticMediaPlayer.play(resourceId);
if (effectId <= 0) {
    // Failed to play resource, lookup effectId error code value in
    // TouchSenseSDKError class
}
```

NATIVE

```
int effectId = touchsensesdk_play(player, resourceId, TSSDK_NORMAL);
if (effectId <= 0) {
    // Failed to play resource, lookup effectId error code value in
    // TSSDK_Error enum (found in c_tssdk_error_codes.h)
}
```

UNITY

```
int effectId = HapticsManager.PlayResource(resourceId);
if (effectId <= 0) {
    // Failed to play resource, lookup effectId error code value in
    // TouchSenseSDKError enum (found in HapticMediaPlayer.cs) or by using
    // HapticsManager.GetErrorString(effectId)
}
```

3.4 Additional playback options

TS SDK supplies multiple APIs that allow a haptic effect to be controlled much like audio or video as it being played, including stop, pause, resume, seek, and mute functionality.

JAVA

```
// Stop a haptic effect. effectId will no longer be valid.
mHapticMediaPlayer.stop(effectId);

// Pause a playing haptic effect. Playback position will be remembered.
mHapticMediaPlayer.pause(effectId);

// Resume a paused haptic effect
mHapticMediaPlayer.resume(effectId);

// Seek playback a haptic effect to a specific timestamp in milliseconds
mHapticMediaPlayer.seek(effectId, timestampMS);

// Mute a haptic effect playback
mHapticMediaPlayer.mute(effectId);

// Unmute a muted haptic effect playback
mHapticMediaPlayer.unmute(effectId);

// Get information a haptic effect state
mHapticMediaPlayer.getEffectInfo(effectId,
                                HapticMediaPlayer.EffectInfo.EFFECT_STATE);
```

NATIVE

```
// Stop a haptic effect. effectId will no longer be valid.
touchsensesdk_stop(player, effectId);

// Pause a playing haptic effect. Playback position will be remembered.
touchsensesdk_pause(player, effectId);

// Resume a paused haptic effect
touchsensesdk_resume(player, effectId);

// Seek playback a haptic effect to a specific timestamp in milliseconds
touchsensesdk_seek(player, effectId, timestampMS);

// Mute a haptic effect playback
touchsensesdk_mute(player, effectId);
```

```
// Unmute a muted haptic effect playback
touchsensesdk_unmute(player, effectId);

// Get information a haptic effect state
touchsensesdk_getEffectInfo(player, effectId, TSSDK_EFFECT_STATE);
```

UNITY

```
// Stop a playing haptic effect. effectId will no longer be valid.
HapticsManager.StopEffect(effectId);

// Pause a playing haptic effect. Playback position will be remembered.
HapticsManager.PauseEffect(effectId);

// Resume a paused haptic effect
HapticsManager.ResumeEffect(effectId);

// Seek playback a haptic effect to a specific timestamp in milliseconds
HapticsManager.SeekEffect(effectId, timestampMS);

// Mute a haptic effect playback
HapticsManager.MuteEffect(effectId);

// Unmute a muted haptic effect playback
HapticsManager.UnmuteEffect(effectId);

// Get information a haptic effect state
HapticsManager.GetEffectInfo(effectId,
                             HapticMediaPlayer.EffectInfo.EFFECT_STATE);
```

As seen in the examples above, the `pause` method pauses haptic playback, saving its playback position. The `resume` method continues from the playback position that was saved from invoking the `pause` method. Note that you cannot use `play` to resume an effect because it takes a *resource* ID as opposed to an *effect* ID. Using `play` this way will return `INVALID_PARAMETER`.

Haptic effects can also be seeked by using the `seek` method. The `seek` method will advance playback to a specified position (in milliseconds) when it is playing or paused. However, calling `seek` while the effect is paused does not automatically resume playback.

The `mute` method silences effects. While muted, the effect's timeline still advances, but it doesn't produce vibrations. Revert this state by calling `unmute`.

To retrieve an effect's state and other information (see [Additional SDK Information](#)) use the `getEffectInfo` method at any time during playback.

Finally, the `stop` method will stop the effect playback completely, terminating the effect and ends its lifecycle. At this point, it can no longer be manipulated as its effect ID is no longer valid. This state is also reached when the effect reaches its end via regular playback.

Haptic Effect Priorities

In certain use cases, multiple effects may be played simultaneously, which could result in an altered user experience. To give you more playback control, the SDK allows you to associate a priority to each haptic effect when it is created by `play`. The priority levels are defined in `HapticMediaPlayer.EffectPriority`. There are 5 priority levels: `LOWEST`, `LOW`, `NORMAL`, `HIGH` and `HIGHEST`.

The priority can be altered by simply supplying the optional effect priority parameter to the `play` method. At any given time, only the highest priority effects currently running actually produce vibrations; all others continue to play, but in a muted state. If there are multiple effects with the same highest priority, all such effects will play in parallel.

```
int longEffectId = mHapticMediaPlayer.play(longResourceId,
                                           HapticMediaPlayer.EffectPriority.NORMAL);
int alertEffectId = mHapticMediaPlayer.play(alertResourceId,
                                           HapticMediaPlayer.EffectPriority.HIGHEST);
// While alert effect is playing other lower priority effects are muted
```

Note that the `mute` method cannot supersede or change priority: Calling `unmute` on a low-priority effect does not cause it to produce vibrations. Similarly, muting a higher priority effect does not cause the lower priority ones to produce vibrations.

An effect's priority only has an influence on the effects existing within the same SDK instance. That is, a higher priority effect in one SDK instance will not mute a lower priority effect in a separate SDK instance.

Haptic Resource and Haptic Effect States

The current state of a haptic resource or effect can be retrieved using `getEffectInfo` with either a resource ID or an effect ID.

Possible haptic resource states as defined in `HapticMediaPlayer.ResourceState`:

- **NOT_READY**: The initial state of a remote haptic resource. The resource remains in this state until enough data has been downloaded to validate it, at which point it transits to **READY** and can be played.
- **INVALID**: The resource is either corrupted, does not point to a .hapt file, or is of an unsupported format.
- **DOWNLOAD_ERROR**: The remote resource failed to download. Call `addResource` to try and download it again.
- **READY**: The resource is ready to play.

Possible haptic effects states, as defined in `HapticMediaPlayer.EffectState`:

- **IDLE**: The initial transient state of an effect, immediately after a call to `play` and just before the SDK actually starts rendering the effect.
- **PLAYING**: The effect is being played.
- **PAUSED**: Indicates `pause` has been called on the effect.
- **BUFFERING**: The waiting state when, at any point during playback, the player has consumed all haptic data that has been downloaded so far and must wait for more data to arrive before playback can continue.
- **TIMEOUT**: Similar to **PAUSED**, but occurs when playing a synchronous effect and no call to `update` has been made in the last second. As opposed to the **PAUSED** state however, calling `update` in the **TIMEOUT** state resumes playback at the given position.

Note that there is no **STOPPED** state, since the effect no longer exists once it stops.

It is important to understand that the APIs affecting the `EffectState` (i.e. `play`, `pause`, `resume`, `stop` and `update`) are asynchronous, therefore you might not observe the state change immediately after calling the API.

3.5 Synchronization

Using the `update` method allows the **HapticMediaPlayer** to stay in sync with a time source, whether that is media time or another source. Great user experiences come from events correctly aligned in the audio, video, and haptic domains, so that all three are perceived as a single entity, producing a multisensory experience.

To that end, the client application calls the `update` method to provide timing for effects generated from `SYNC_HAPTIC_EFFECT` resources. This method *must* be called at least once every second to maintain accurate synchronization.

System Time Source Example

This java snippet shows how to use the system time as a time source:

```
// SystemTimeSource.java

package com.immersion.sample.timesource;

import com.immersion.touchsensesdk.HapticMediaPlayer;

public class SystemTimeSource implements Runnable {
    private static final String TAG = "SystemTimeSource";

    private static final long HAPTIC_SYNC_PLAYER_UPDATE_RATE_MS = 1000;
    private Handler mHandler;
    private HapticMediaPlayer mHapticPlayer;

    private long mCurrentTimeMs;
    private long mLastSystemTimeMs;

    public SystemTimeSource(Handler handler, HapticMediaPlayer hapticPlayer) {
        mHandler = handler;
        mHapticPlayer = hapticPlayer;
    }

    public void start() {
        reset();
        mHandler.post(this);
    }

    @Override
    public void run() {
        long currentSystemTimeMs = System.currentTimeMillis();
        long diff = currentSystemTimeMs - mLastSystemTimeMs;

        mCurrentTimeMs += diff;
        mLastSystemTimeMs = currentSystemTimeMs;

        if (mHapticPlayer.update(mCurrentTimeMs) != HapticMediaPlayer.
            TouchSenseSDKError.SUCCESS)
            Log.e(TAG, "Error updating HapticMediaPlayer");

        mHandler.postDelayed(this, HAPTIC_SYNC_PLAYER_UPDATE_RATE_MS);
    }

    public void reset() {
        // Set our time to 0
        mCurrentTimeMs = 0;

        // Save the current system time as last system time
        mLastSystemTimeMs = System.currentTimeMillis();
    }

    public void stop() {
        mHandler.removeCallbacks(this);
    }
}
```

In this snippet the call `System.currentTimeMillis()` methods to keep track of time. When playback starts, `SystemTimeSource` sets `mCurrentTimeMs` to 0 and starts a 1-second update rate. Time is tracked by determining the difference between the current time and the last time update was called. This difference is added to `mCurrentTimeMs` and the result is passed into `update`. Immersion recommends that the `Handler` object, which is passed into the

SystemTimeSource's constructor, be associated with a high-priority HandlerThread or Looper to ensure correct timing and to produce a quality user experience.

Media Time Source Example

The Java snippet below demonstrates how to use Android's MediaPlayer as a time source, maintaining synchronization between the **HapticMediaPlayer** and the MediaPlayer.

```
// MediaTimeSource.java
package com.immersion.sample.MediaTimeSource;

import android.media.MediaPlayer;
import android.os.Handler;
import com.immersion.touchsensesdk.HapticMediaPlayer;

public class MediaTimeSource implements Runnable {
    private static final String TAG = "MediaTimeSource";
    private static final long HAPTIC_SYNC_PLAYER_UPDATE_RATE_MS = 1000;

    private Handler mHandler;
    private HapticMediaPlayer mHapticPlayer;
    private MediaPlayer mMediaPlayer;

    public MediaTimeSource(Handler handler, HapticMediaPlayer hapticPlayer,
                           MediaPlayer mediaPlayer) {
        mHandler = handler;
        mHapticPlayer = hapticPlayer;
        mMediaPlayer = mediaPlayer;
    }

    public void start() {
        mHandler.post(this);
    }

    @Override
    public void run() {
        int currentPositionMs = mMediaPlayer.getCurrentPosition();
        if (mHapticPlayer.update(currentPositionMs) != HapticMediaPlayer.
            TouchSenseSDKError.SUCCESS)
            Log.e(TAG, "Error updating HapticSyncPlayer");

        mHandler.postDelayed(this, HAPTIC_SYNC_PLAYER_UPDATE_RATE_MS);
    }

    public void stop() {
        mHandler.removeCallbacks(this);
    }
}
```

AutoSyncPlayer

The SDK's **VideoSampleApp** sample application bundles a helper class that handles the synchronization complexity. If you are using Android's MediaPlayer as your main player, use the **AutoSyncPlayer** to play haptics with ease.

AutoSyncPlayer allows users to play media and haptics through a few simple API calls by synchronizing them in the **AutoSyncPlayer** code. When the media is paused, the haptics player pauses. When the media is seeking, the haptics player seeks. All interactions required to provide a great experience are handled automatically. To create an instance of

AutoSyncPlayer, simply invoke its static `create` method and supply the same arguments specified for `HapticMediaPlayer.create`:

To open a media file, call `openMedia`:

```
player.openMedia(getApplicationContext(), someUri);
```

To associate a haptic resource to the player, call `addHapticResource`:

```
player.addHapticResource(someUrl);
```

When you are ready for playback, simply call `start`. Call `stop`, `pause`, and `seek` for other multimedia actions to match Android's `MediaPlayer`.

See the **VideoSampleApp** source code for more details.

3.6 Clean Up

Call `dispose` to free system resources used by the **HapticMediaPlayer**. Once `dispose` is called, the **HapticMediaPlayer** instance is no longer usable. You must make another one by calling `create` again. It is recommended to call `dispose` at the end of the application's life-cycle as opposed to the activity's lifecycle.

Important: You must call `dispose` even if `create` did not successfully initialize the player instance.

JAVA

```
mHapticMediaPlayer.dispose();
```

NATIVE

```
touchsensesdk_dispose(player);
```

UNITY

You do not need to manually dispose of the **HapticMediaPlayer** since the **HapticsManager** automatically disposes of it when the application is terminated.

3.7 ProGuard

If your release build uses a ProGuard script, find your ProGuard rules file (default: **proguard-rules.pro**) and add the following line:

```
-keep class com.immersion.touchsensesdk.** { *; }
```

3.8 Additional SDK Information

This section discusses additional options for retrieving miscellaneous SDK and player information.

Retrieving HapticMediaPlayer Information

Use the `getPlayerInfo` method to obtain information about the player. The list of available information is listed in `HapticMediaPlayer.PlayerInfo`:

- **PLAYER_STATE**: State of the `HapticMediaPlayer`.
- **PLAYER_VERSION_MAJOR**: Major version of the `HapticMediaPlayer`.
- **PLAYER_VERSION_MINOR**: Minor version of the `HapticMediaPlayer`.
- **PLAYER_VERSION_BUILD**: Build version of the `HapticMediaPlayer`.

- **PLAYER_VERSION_MAINTENANCE**: Maintenance version of the HapticMediaPlayer.
- **PLAYER_VERSION_PATCH**: Patch version of the HapticMediaPlayer.
- **PLAYER_PLAYBACK_TYPE**: Playback type of the HapticMediaPlayer.

Retrieving HapticEffect Information

Use the `getEffectInfo` method to obtain information about the player. The list of available information is listed in `HapticMediaPlayer.EffectInfo`:

- **RESOURCE_STATE**: State of a HapticResource.
- **EFFECT_STATE**: State of a HapticEffect.
- **EFFECT_MUTED_STATE**: Muted state of a HapticEffect.
- **RESOURCE_TYPE**: Type of HapticResource (Sync or Async).
- **RESOURCE_LOCATION**: Location of HapticResource (Local or Remote).
- **RESOURCE_DURATION**: Duration in milliseconds of HapticResource.

Checking HapticMediaPlayer's State

Use the `getPlayerInfo` method to obtain the **HapticMediaPlayer**'s state. The states are defined in `HapticMediaPlayer.PlayerState`:

- **INITIALIZED**: Initialization succeeded, **HapticMediaPlayer** is ready to use.
- **MISSING_PERMISSIONS**: The Android manifest is missing VIBRATE and/or INTERNET permissions.
- **INVALID_CREDENTIALS**: The supplied username, password and/or DNS is either null, empty or have invalid format.
- **BAD_PARAMETER**: The supplied context object is null, invalid or an instance with provided credentials already exists.

When the **HapticMediaPlayer** is not in the `INITIALIZED` state, any API calls return the error code `PLAYER_NOT_INITIALIZED`, with the exception of `getPlayerInfo`.

Retrieving the TS SDK/HapticMediaPlayer Version

Use the `getPlayerInfo` method to retrieve SDK version information:

```
// Retrieve the version as a string
public String getPlayerVersion(HapticMediaPlayer p) {
    int major = p.getPlayerInfo(HapticMediaPlayer.PlayerInfo.PLAYER_VERSION_MAJOR);
    int minor = p.getPlayerInfo(HapticMediaPlayer.PlayerInfo.PLAYER_VERSION_MINOR);
    int build = p.getPlayerInfo(HapticMediaPlayer.PlayerInfo.PLAYER_VERSION_BUILD);

    return String.format("%d.%d.%d", major, minor, build);
}
```

4. API Library

Table 4 defines the methods available in the TS SDK API library.

Table 4: TS SDK API Methods

| Method | Definition |
|--|---|
| <code>int addResource(String resourceURL, int hapticEffectType)</code> | Adds a haptic resource and returns its unique ID. |
| <code>synchronized static HapticMediaPlayer create(Context context, String username, String password)</code> | Create an instance of the HapticMediaPlayer with the default dns. |
| <code>synchronized static HapticMediaPlayer create(Context context, String username, String password, String dnsURL)</code> | Create an instance of the HapticMediaPlayer. |
| <code>int dispose()</code> | Disposes of the HapticMediaPlayer instance, all existing resources, and any playing effects. |
| <code>int getEffectInfo(int uid, int info)</code> | Retrieves information on a resource or an effect according to its unique ID. |
| <code>int getPlayerInfo(int info)</code> | Retrieves information on the HapticMediaPlayer according to the information code provided. |
| <code>int mute(int effectID)</code> | Mutes an effect identified by its unique ID. |
| <code>int pause(int effectID)</code> | Pauses an effect, identified by its unique ID. |
| <code>int play(int resourceID)</code> | Plays a haptic resource identified by its unique ID, with default NORMAL effect priority. Returns a unique effect ID. |
| <code>int play(int resourceID, int priority)</code> | Plays a haptic resource, identified by its unique ID, and returns a unique effect ID. |
| <code>int removeResource(int resourceID)</code> | Removes a haptic resource, identified by its unique ID, and stops any playing effects associated with it. |
| <code>int resume (int effectID)</code> | Resumes an effect, identified by its unique ID. |
| <code>int seek(int effectID, long timestampMS)</code> | Seeks an effect, identified by its unique ID, to a specific timestamp. |
| <code>int stop(int effectID)</code> | Stops an effect, identified by its unique ID. |
| <code>int unmute(int effectID)</code> | Unmutes an effect identified by its unique ID. |
| <code>int update(int effectID, long timestampMS)</code> | Updates an effect, identified by its unique ID, to a specific timestamp. |

5. Troubleshooting

TS SDK returns error codes to the calling application when it encounters a problem during operation. Almost all API calls to the SDK return an integer error code and do not throw exceptions. This section describes all possible errors generated by the player, their causes, and how to resolve them.

5.1 Error Codes

[Table 5](#) below provides the complete list of error codes defined in `HapticMediaPlayer.TouchSenseSDKError`.

Table 5: Error codes

| Name | Value | Description |
|-----------------------------|-------|---|
| SUCCESS | 0 | Call succeeded per API specification |
| INVALID_PARAMETER | -1 | Call failed, invalid parameter passed |
| INVALID_URI | -2 | Call failed, invalid uri passed |
| INVALID_EFFECT | -3 | Call failed, invalid effect |
| OUT_OF_MEMORY | -5 | Call failed, ran out of memory |
| IO_ERROR | -7 | Call failed, error while reading/writing file |
| HAPT_NOT_READY | -9 | Call failed, hapt file is not ready for playback |
| TOO_MANY_EFFECTS | -10 | Call failed, too many haptic resources were already loaded |
| PLAYER_NOT_INITIALIZED | -11 | Call failed, player not initialized |
| TOO_MANY_CONCURRENT_EFFECTS | -12 | Call failed, too many effects playing at the same time |
| INVALID_STATE | -13 | Call failed, invalid state |
| LIB_VERSION_NOT_FOUND | -14 | Call failed, Haptic Media Player version not found |
| SDK_INOPERATIVE | -15 | Call failed, TouchSense SDK suffered a fatal error and is now inoperative |

5.2 Return Codes

Depending on the action required, each API method provides different return codes. [Table 6](#) below lists all the return codes and error codes TS SDK generates. These are defined in `HapticMediaPlayer.TouchSenseSDKError`.

Table 6: Return codes

| API Call | Return Code | Description |
|--|-----------------------------|--|
| addResource | >0 | Resource ID (success) |
| | INVALID_URI | URI either does not point to a hapt or cannot be accessed |
| | INVALID_PARAMETER | Effect Type is invalid |
| | IO_ERROR | Could not read the local resource. Ensure you have obtained the READ_EXTERNAL_STORAGE permission if reading from the SD card |
| | OUT_OF_MEMORY | Not enough memory to create the resource |
| | TOO_MANY_EFFECTS | Exceeded the maximum number of resources |
| | PLAYER_NOT_INITIALIZED | Player disposed or player state not INITIALIZED |
| removeResource | SUCCESS | Resource successfully removed |
| | INVALID_PARAMETER | The given ID does not correspond to a resource |
| | PLAYER_NOT_INITIALIZED | Player disposed or player state not INITIALIZED |
| play | >1000 | Effect ID (success) |
| | INVALID_PARAMETER | The given ID does not correspond to a resource |
| | HAPT_NOT_READY | The resource is not in the READY state. A remote resource may need more time to download |
| | INVALID_EFFECT | The effect could not be created. See the log for details |
| | TOO_MANY_CONCURRENT_EFFECTS | The maximum number of simultaneous effects has been reached. Either stop some effects or wait for them to finish |
| | OUT_OF_MEMORY | Not enough memory to play the effect |
| | PLAYER_NOT_INITIALIZED | Player disposed or player state not INITIALIZED |
| pause resume stop mute unmute | SUCCESS | The operation completed successfully |
| | INVALID_PARAMETER | The given ID does not correspond to an existing effect |
| | PLAYER_NOT_INITIALIZED | Player disposed or player state not INITIALIZED |
| | | |
| | | |
| update seek | SUCCESS | The operation completed successfully |
| | INVALID_PARAMETER | The given ID does not correspond to an existing effect, or the given position is negative or longer than the effect duration |
| | PLAYER_NOT_INITIALIZED | Player disposed or player state not INITIALIZED |

| | | |
|----------------------|--|---|
| getPlayerInfo | >0 INVALID_PARAMETER LIB_VERSION_NOT_FOUND PLAYER_NOT_INITIALIZED | Requested information (success) The info parameter is not one of the values defined in HapticMediaPlayer.PlayerInfo The given PLAYER_VERSION_* info parameter is not set Player disposed or player state not INITIALIZED |
| getEffectInfo | >0 INVALID_PARAMETER PLAYER_NOT_INITIALIZED | Requested information (success) The given ID is not an existing effect or resource, or the info parameter is not one of the values defined in HapticMediaPlayer.EffectInfo Player disposed or player state not INITIALIZED |

5.3 Pitfalls and Recovery Steps

TS SDK outputs error messages any time there is a problem. [Table 7](#) below lists some of the more common error messages output to a log by the SDK during integration or playback, along with a recommended solution for each.

Table 7: Error messages and Exceptions

| Error Message | Occurrence | Solution |
|--|---|---|
| "Policy forbids playback for resource id ..." | Two possible causes: 1) The supplied credentials do not allow use of the SDK or that specific resource. OR 2) The device has been too long without an internet connection and the credentials cannot be validated. | Ensure that you are using the correct username, password and possibly DNS URL supplied by Immersion. Contact your Immersion representative if this error persists. |
| "Error loading libTouchSenseSDK.so. Please make sure it is in the libs/armeabi directory of your project." | The Android runtime was unable to find the libTouchSense.so native library. | Ensure libTouchSenseSDK.so is located as advised in the error message. Note that if your app uses any 64-bit library (even if it's only one), then Android will load only 64-bit libraries. Contact Immersion to obtain a 64-bit version of libTouchSenseSDK.so . |
| "Error: Invalid hapt format (a.b), we expected x.y!" | Occurs when the API expects a different version of the .hapt file. | Make sure you are using the correct .hapt file. If this problem persists, please contact your Immersion sales representative. |

6. FAQs

➤ How do I open the SDK API documentation in a browser?

To open the SDK API documentation in a browser, simply unzip the file **javadoc.zip** located under the **Documentation** directory. Go to the **javadoc** directory and double-click on the file **index.html**.

➤ How do I import the SDK API documentation into an Android Studio project?

To view the SDK API documentation in your Android Studio project:

1. Copy **javadoc.zip** from under the SDK package's **Documentation** directory.
2. Paste **javadoc.zip** under your project's **libs** directory.
3. Open the **.xml** file located under your project's **.idea/libraries/** path.
4. Type in the location of your **javadoc.zip** directory under the **<JAVADOC>** tag, as shown below. This activates context-sensitive documentation for the SDK API:

```
<component name="libraryTable">
  <library name="TouchSenseSDK-vX.Y.Z">
    <CLASSES>
      <root url="jar://$PROJECT_DIR$/libs/TouchSenseSDK-vX.Y.Z.jar!/" />
    </CLASSES>
    <JAVADOC>
      <root url="jar://$PROJECT_DIR$/libs/javadoc.zip!/" />
    </JAVADOC>
    <SOURCES />
  </library>
</component>
```

5. Hover your cursor over an SDK API method in your code and type **CTRL+Q**. A context-sensitive window pops up with a definition for the method ([Figure](#)):

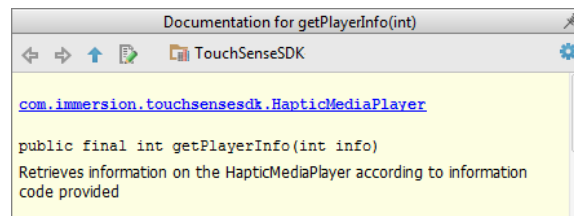


Figure 3: TS SDK API Documentation in Android Studio

➤ How do I update my SDK?

If your Immersion sales representative provides you with an updated version of TS SDK, it is important you update your libraries with the new ones. Immersion is always looking for ways to improve its products and upgrading guarantees you are using the most secure and reliable version of the SDK.

TS SDK is distributed in the form of two libraries: a **.jar** (java library) and a **.so** (native library). To update your SDK, you must replace both libraries together: Replace the old **.jar** and **.so** libraries with the new ones obtained from Immersion. In Android Studio, simply drag and drop the two files into their corresponding locations to overwrite the existing ones.

Immersion recommends always cleaning the project after updating the libraries to make sure your application uses the latest version of the libraries.

➤ What platforms are supported by the SDK?

The SDK supports every Android device that ships with an actuator and runs Android v4.0+.

➤ What is the size of the current SDK?

The current SDK is less than 400 kB in size.

➤ How much CPU and memory does the SDK use?

CPU usage is usually very low - typically less than 3%. The SDK consumes less than what is normally needed for audio playback. Memory usage depends on the size of the **.hapt** resource(s) open at a given time.

➤ **How much battery does haptic feedback consume?**

Power usage is minimal at a few milliwatts. The SDK accounts for less than 3% of overall CPU usage on most Android devices.

➤ **Is the SDK stable and secure?**

Immersion software must pass rigorous robustness, quality, and reliability testing before being released to a customer.

7. Additional Support Information

7.1 Support & Services

Immersion provides a variety of integration and design support services to commercial licensees. Refer to your executed license agreement or speak to your Immersion sales representative to learn more about support options available to you.

If you have comments regarding this or other Immersion documentation, or if you need to contact Immersion for technical assistance, please contact your Immersion sales representative or send an email to content@immersion.com.

7.2 License Expiry

Your license expiry is validated by a cloud service. The product will cease to function when your license has expired. To determine the exact duration of your license, refer to your executed Immersion evaluation or Immersion software license agreement.

7.3 Supporting Documentation

In addition to this guide, please find the following documents in the **Documentation** directory of the TS SDK package:

- *TouchSense® SDK for Mobile Apps Quick Start Guide (TouchSenseSDK_Quick_Start_Guide.pdf)*
- *TouchSense® SDK for Mobile Apps Release Notes (release_notes.txt)*
- *TouchSense® SDK for Mobile Apps Javadoc (javadoc.zip)*

8. About Immersion

Immersion (NASDAQ: IMMR) provides high quality, low-power tactile feedback for the consumer environment. At Immersion, touch is believed to be an essential part of the human experience. It provides critical confirmation feedback, enhances the sense of realism, and enables a rich communication experience between users and their devices.

Immersion technology has been broadly adopted in over 1 billion consumer devices, including console game controllers, mobile phones, and wearable and automotive interfaces. The company offers software solutions and IP licenses to OEMs to create tactile user experiences, as well as SDKs and APIs to help developers and content providers enhance their applications and services with haptics. Immersion was founded in 1993 and holds more than 2300 issued and pending patents in the U.S. and other countries. <http://www.immersion.com>.



50 Rio Robles
San Jose, CA 95134
immersion.com
touchsense.com

LEGAL DISCLAIMERS

General Disclaimer.

Although Immersion has attempted to provide accurate information in this document, Immersion assumes no responsibility for the accuracy of the information. Immersion may change the products or services (or information relating thereto) mentioned at any time without notice. Mention of non-Immersion products or services is for information purposes only and constitutes neither an endorsement nor a recommendation.

EXCEPT WHERE EXPRESSLY PROVIDED OTHERWISE, THIS DOCUMENT, INCLUDING ALL CONTENT, MATERIALS, INFORMATION, SOFTWARE, PRODUCTS AND SERVICES DESCRIBED THEREIN, ARE PROVIDED ON AN "AS IS" AND "AS AVAILABLE" BASIS. IMMERSION EXPRESSLY DISCLAIMS ALL WARRANTIES OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT AND WARRANTIES ARISING FROM A COURSE OF DEALING, USAGE OR TRADE PRACTICE.

IMMERSION SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL OR INCIDENTAL DAMAGES, INCLUDING WITHOUT LIMITATION, LOST PROFITS OR REVENUES, COSTS OF REPLACEMENT GOODS, LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT OR ANY IMMERSION PRODUCT, DAMAGES RESULTING FROM USE OF OR RELIANCE ON CONTENT, EVEN IF IMMERSION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. YOUR SOLE REMEDY FOR DISSATISFACTION WITH THIS DOCUMENT AND/OR THE INFORMATION CONTAINED HEREIN IS TO CEASE ALL OF YOUR USE OF THIS DOCUMENT AND SUCH INFORMATION. To the extent required by law in other jurisdictions, certain of the above liability limitations or warranty exclusions may not apply to you. You agree that the warranties and the liability of Immersion would in such case be limited to the greatest extent permitted by law. The information in this document may be changed at any time.

Intellectual Property.

The trademarks, logos and service marks ("Marks") displayed in this document are the property of Immersion or other third parties. You are not permitted to use these Marks without the prior written consent of Immersion or such appropriate third party.

All information in this document is (and shall continue to be) owned exclusively by Immersion or other third parties owners, and is protected under applicable copyrights, patents, trademarks, trade dress, and/or other proprietary rights, and the copying, redistribution, use or publication by you of any such information or any part of this document is prohibited. Under no circumstances will you acquire any ownership rights or other interest in any information contained herein by or through your use of this document.

©2018 Immersion Corporation. All rights reserved. Immersion, the Immersion logo, and TouchSense are trademarks of Immersion Corporation in the United States and other countries. All other trademarks are the property of their respective owners.