Oracle® Communications Session Border Controller

TSCF SDK Guide

Release 2.0.0

F23719-01

October 2019

Oracle Communications Session Border Controller TSCF SDK Guide, Release 2.0.0

F23719-01

Contents

About This Guide

The Oracle® Communications Tunneled Session Controller SDK Guide describes the client-side SDK (software development kit) that facilitates the creation of secure tunnels between a client application and the Tunneled Session Controller Function (TSCF) of the Oracle Communications Session Border Controller. A client is typically a softphone application that utilizes the SDK software libraries and source code to create TLS tunnels to a TSCF service, thus achieving secure real time communications and ubiquitous firewall traversal.

This document specifically describes the SDK, functional libraries, and source code supplied with the SDK Version 2.0.0.

Documentation Set

The following table describes the documentation set for this release.

Document Name	Document Description
Acme Packet 4600 Hardware Installation Guide	Contains information about the components and installation of the Acme Packet 4600.
Acme Packet 6100 Hardware Installation Guide	Contains information about the components and installation of the Acme Packet 6100.
Acme Packet 6300 Hardware Installation Guide	Contains information about the components and installation of the Acme Packet 6300.
Acme Packet 6350 Hardware Installation Guide	Contains information about the components and installation of the Acme Packet 6350.
Release Notes	Contains information about the current documentation set release, including new features and management changes.
ACLI Configuration Guide	Contains information about the administration and software configuration of the Service Provider Oracle Communications Session Border Controller.
ACLI Reference Guide	Contains explanations of how to use the ACLI, as an alphabetical listings and descriptions of all ACLI commands and configuration parameters.
Maintenance and Troubleshooting Guide	Contains information about Oracle Communications Session Border Controller logs, performance announcements, system management, inventory management, upgrades, working with configurations, and managing backups and archives.
MIB Reference Guide	Contains information about Management Information Base (MIBs), Oracle Communication's enterprise MIBs, general trap information, including specific details about standard traps and enterprise traps, Simple Network Management Protocol (SNMP) GET query information (including standard and enterprise SNMP GET query names, object identifier names and numbers, and descriptions), examples of scalar and table objects.

Document Name	Document Description
Accounting Guide	Contains information about the Oracle Communications Session Border Controller’s accounting support, including details about RADIUS and Diameter accounting.
HDR Resource Guide	Contains information about the Oracle Communications Session Border Controller’s Historical Data Recording (HDR) feature. This guide includes HDR configuration and system-wide statistical information.
Administrative Security Essentials	Contains information about the Oracle Communications Session Border Controller’s support for its Administrative Security license.
SBC Family Security Guide	Contains information about security considerations and best practices from a network and application security perspective for the Oracle Communications Session Border Controller family of products.
Installation and Platform Preparation Guide	Contains information about upgrading system images and any pre-boot system provisioning.
Call Traffic Monitoring Guide	Contains information about traffic monitoring and packet traces as collected on the system. This guide also includes WebGUI configuration used for the SIP Monitor and Trace application.
HMR Resource Guide	Contains information about configuring and using Header Manipulation Rules to manage service traffic.
REST API Guide	Contains information about the supported REST APIs and how to use the REST API interface.

Revision History

Date	Description
September 2019	• Initial release

Revision History

Date	Description
September 2019	• Initial release
November 2019	• Adds iOS support for 2.0.0m1

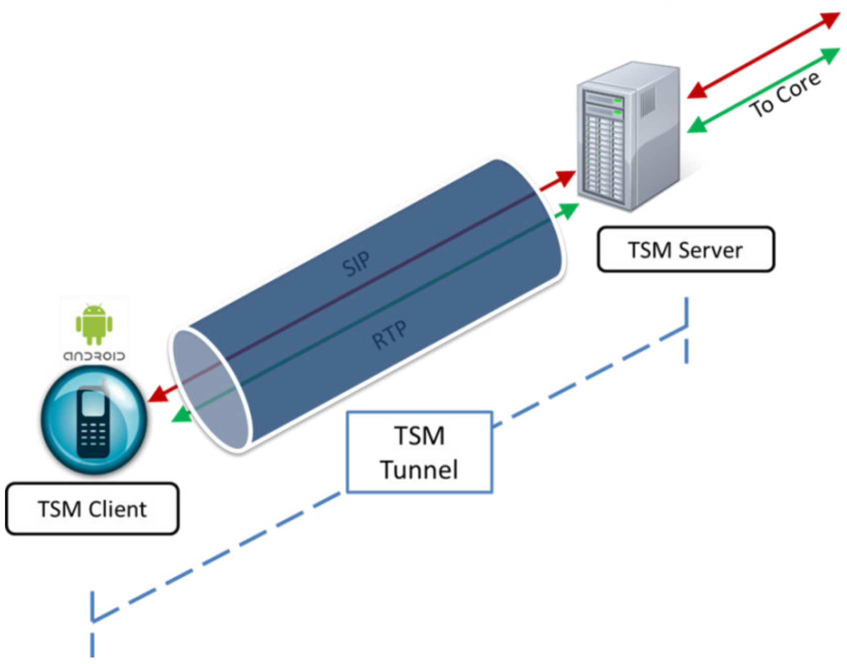
1

Overview

Tunnel Session Management (TSM) improves firewall traversal for real time communications for OTT VoIP applications and reduces the dependency on SIP/TLS and SRTP by encrypting access-side VoIP within standardized VPN tunnels. As calls or sessions traverse a TSM tunnel, the Oracle Communications Session Border Controller (OCSBC) will route all SIP and RTP traffic from within the TSM tunnel to the core (or appropriate destination).

Oracle Communications is working with other telecom providers and vendors to standardize TSM. Within the 3GPP, TSM is called a Tunneled Services Control Function (TSCF). Currently the 3GPP Technical Requirement draft is TR 33.8de V0.1.3 (2012-05) as a standardized approach for overcoming non-IMS aware firewall issues with supporting companies including China Mobile, Ericsson, Huawei, Intel, RIM, Vodafone, and ZTE. Beyond the standard, we provide exceptional tunnel performance & capacity within the OCSBC as well as high availability, DDoS protection and our patented TSM Tunnel Redundancy to improve audio quality in lossy networks such as the Internet.

Figure 1-1 Basic TSM Setup



TSM consists of two parts:

- the TSM server (often referred to as a TSCF or Tunneled Services Control Function)
- the TSM client

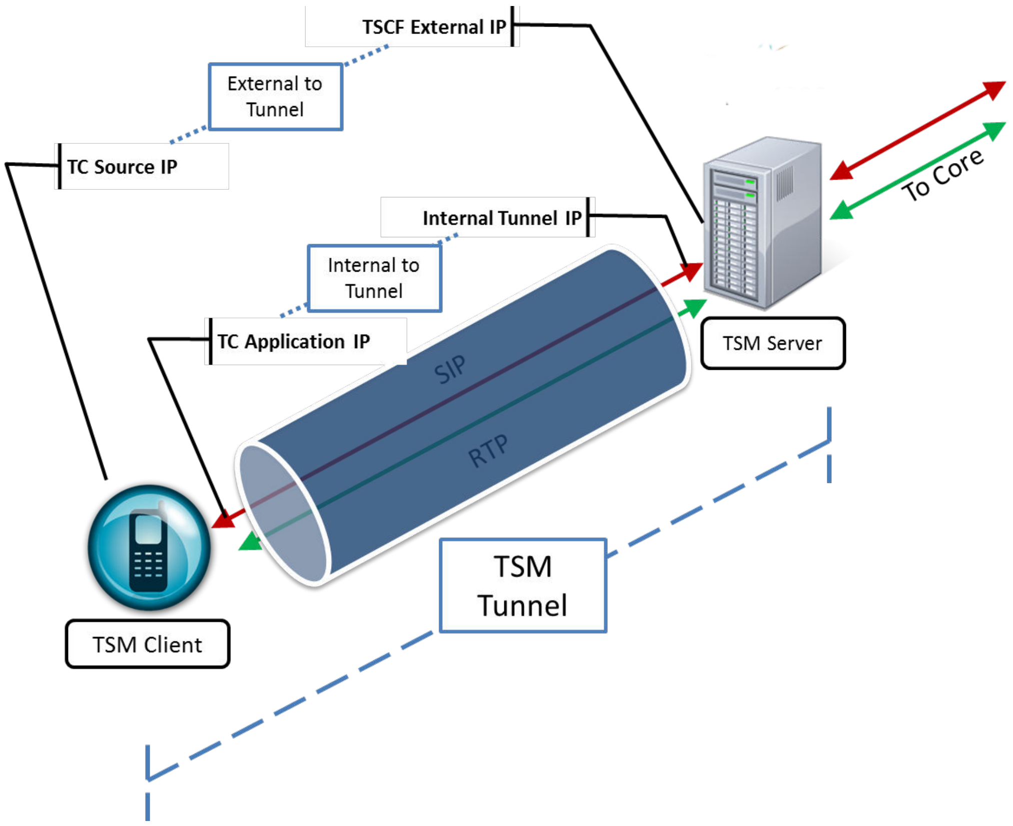
The TSM server resides and runs on the OCSBC and the TSM client runs within applications that reside on workstations, laptops, tablets, mobile devices and even network elements.

To deploy TSM-enabled clients such as softphones, SIP-enabled applications or contact center agent applications, customers and 3rd party ISVs will need to incorporate the open source TSM software libraries into their applications which will establish tunnels to the TSM server.

TSM Tunnel

The following diagram briefly explains the various IP addresses utilized during the TSM session.

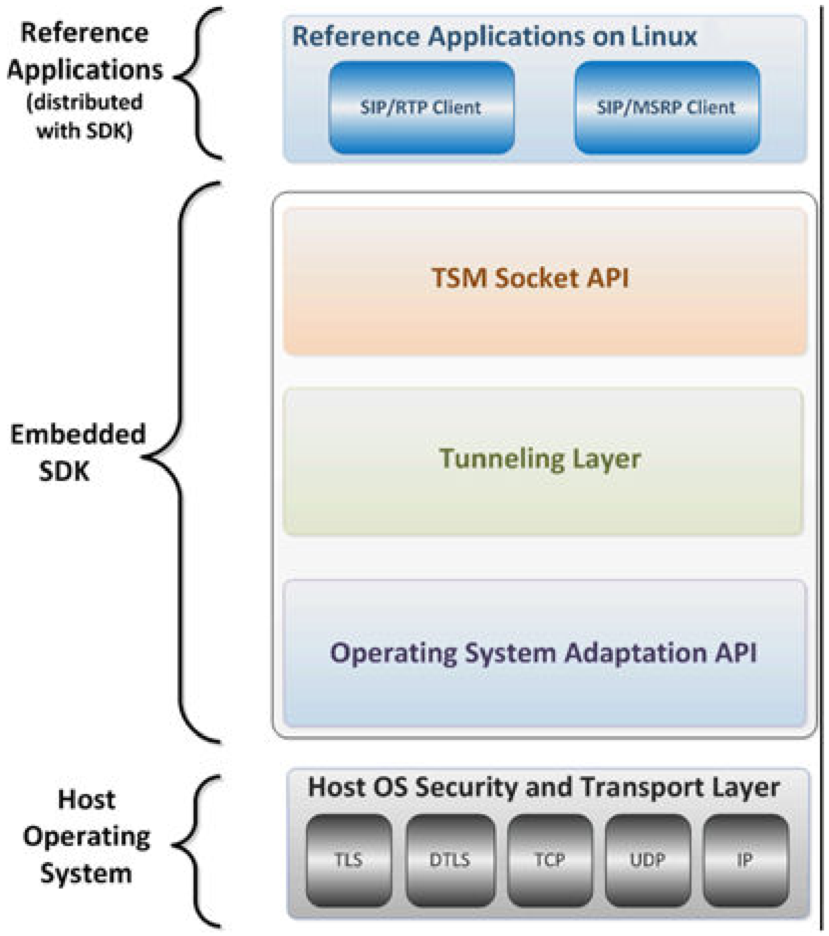
- TSCF External IP—This IP address is visible to any endpoint on the Internet and is used to initiate the TSM session between the TC and the TSCF. This may be configured under **security**, and then **tscf**, and then **tscf-interface**. See the TSCF chapter in the *ACLI Configuration Guide* to configure the TSCF function on the server.
- TC Source IP—This IP address corresponds to the source address of the TC in its respective access network or it could be the IP of the Proxy behind which it is located.
- Internal Tunnel IP—This IP address will be assigned to the TC (once TLS authentication is successful) from a configured pool of IP addresses on the TSCF. It will be used to facilitate communication with the core (P-CSCF). The address pool can be configured under **security**, and then **tscf**, and then **tscf-address-pool**.
- TC Application IP—This is the IP address associated with the respective application (SIP / RTP / other) at the TC. This is the same as the Internal Tunnel IP.



SDK Host Operating System Relationship

The following illustrations depict the relationship between the SDK and the host operating system:

Figure 1-2 SDK/Host OS Relationship (Simplified View)


Reference Applications (distributed with SDK)
Reference Applications on Linux
SIP/RTP Client
SIP/MSRP Client
Embedded SDK
TSM Socket API
Tunneling Layer
Operating System Adaptation API
Host Operating System
Host OS Security and Transport Layer
TLS
DTLS
TCP
UDP
IP

Provided Functionality

Operating Systems Support

This SDK release supports the following operating systems:

- Linux flavors (using GCC version 4.4.7 or 4.8.5)
- Android 9 Pie (64-bit)
- iOS 12

Platform Support

This SDK supports any platform running S-CZ8.3.0.

Proxy Support

This SDK release supports the following proxy authentication types:

- Basic
- Digest
- NTLMv2
- SPNEGO

If proxy authentication is enabled, the SDK will try to use SPNEGO authentication. If that fails, the SDK tries to use NTLMv2.

Additional Features

This SDK release also supports:

- On-the-fly integration of downloaded OpenSSL with TSCF libraries.
- Server Assigned Configuration mode
- Security Traversing Gateway (STG)
- Payload multiplexing within a tunnel
- Each SDK instance can support:
  - Up to 3 concurrent voice calls
  - Up to 10 MSRP chat sessions
  - 1 MSRP file transfer session
- Tunnel Transport
  - TCP
  - UDP
  - TLS
  - DTLS
- IP version
  - IPv4
  - IPv6

> **Note:**
>
> When used in Decoupled Mode, the TSCF also supports mixing IPv4 and IPv6. For example, you can use an IPv6 external address outside the tunnel and an IPv4 address inside the tunnel, or vice versa.

2

Compile the TSM Library and Documentation

Read the documentation that corresponds to your application's target operating system.

Operating System	Description	Location
android	This file provides information on how to compile the TSM SDK for Android.	`sdk/lib/README.android`
iOS	This file provides information on how to compile the TSM SDK for iOS.	`sdk/lib/README.ios`
Linux	This file provides information on how to compile the TSM SDK for Linux.	`sdk/lib/README`

WARNING:

The OpenSSL library must be downloaded before proceeding with development.

SDK Directories

SDK directories are shown below. Note that not all listed directories may be present (or supported) in the current release.

Path	Description
`apps`	SDK based applications
`apps/tsc_sip`	Reference demonstration/development guide app (tsc_sip_client.c)
`docs`	SDK Documentation
`docs/html`	Authoritative API HTML-based documentation. Access via “.../html/index.html” after running `make doxygen`.
`extlib`	External, optional libraries
`lib`	SDK Library source – to be linked with the target application
`lib/android-ndk`	Android Specific library instructions and precompiled libs
`lib/CSM`	Tunneling Client State Machine
`lib/EIP`	Embedded TCP/UDP/IP Stack
`lib/include`	SDK API definitions
`lib/OSAA`	Operating System Application Adaptation APIs
`lib/TAPI`	Tunnel Data and Control APIs
`lib/TPL`	Tunnel Control and Data Message Parsing Libraries
`tools`	Development Tools
`tools/wireshark`	TSCF protocol dissector

Download and Compile OpenSSL

The default version of OpenSSL has been removed. Developers should download the desired version of OpenSSL and modify the build script to allow on-the-fly integration with the SDK.

1. Download the version of OpenSSL you want to integrate into the SDK.

   The customer is responsible for selecting a secure version of OpenSSL from https://www.openssl.org/. Oracle can confirm OpenSSL version 1.1.1a works with the SDK.

2. In the build script for your target operating system, set the `VERSION` variable to the version number of OpenSSL.

   The build scripts are located in the `sdk/extlib` directory and the `VERSION` variable is found at the top of the script.

   ```
   VERSION="1.1.1a"
   ```

3. Run the build script for your target operating system.

   For example:

   ```
   ./build_androidlib.sh
   ```

Android Development

Use the following method when developing for Android.

Install Android Build Environment Pre-Requisites

Before setting up the Android build environment, extract the SDK tar file and install the following pre-requisites.

1. Extract the TSM SDK 2.0 tar file.

```
tar xvf nnTSC200.tar.gz
```

2. Install the latest version of the Java Development Kit.
3. Download and extract the latest Android Studio with the Android SDK.
   a. Navigate to https://developer.android.com/studio/#downloads.
   b. Download the package for your operating system.
   c. Extract the downloaded package.

      ```
      tar xvf android-studio-ide-191.5791312-linux.tar.gz
      ```
4. Download and extract the latest Android NDK.
   a. Navigate to https://developer.android.com/ndk/downloads.
   b. Download the package for your operating system.
   c. Extract the downloaded package.

      ```
      unzip android-ndk-r20-linux-x86_64.zip
      ```
5. Download the desired version of OpenSSL into the `sdk/extlib` directory.

   TSM SDK 2.0 supports the 64-bit OpenSSL version 1.1.1a.

   ```
   cd sdk/extlib/
   curl -O https://www.openssl.org/source/old/1.1.1/openssl-1.1.1a.tar.gz
   ```

Build the TSC SDK Libraries for Android

1. In the `sdk/extlib` directory, update the `android_env.sh` script based on the setup, and source it to set the environment. Then run the `build_androidlib.sh` script with the OpenSSL version as a parameter.

   ```
   source android_env.sh
   ./build_androidlib.sh 1.1.1a
   ```
2. Run an NDK build in the `sdk/lib/android-ndk` directory.
   a. Set the *NDK_PROJECT_PATH* variable to the `sdk/lib/android-ndk` directory and navigate to that directory.

      ```
      cd ..
      export NDK_PROJECT_PATH="$PWD/lib/android-ndk"
      cd $NDK_PROJECT_PATH
      ```
   b. Run the NDK build.

      ```
      ndk-build APP_ABI=arm64-v8a APP_PLATFORM=android-28 clean
      ndk-build APP_ABI=arm64-v8a APP_PLATFORM=android-28 build
      ```
3. Run an NDK build in the `sdk/apps/tsc_sip/tsc_sip_client/` directory.
   a. Now set the *NDK_PROJECT_PATH* variable to the `sdk/apps/tsc_sip/tsc_sip_client/` directory and navigate to that directory.

      ```
      cd ../..
      export NDK_PROJECT_PATH="$PWD/apps/tsc_sip/tsc_sip_client"
      cd $NDK_PROJECT_PATH
      ```
   b. Run the NDK build again.

Note:

Do not include `build` at the end of the second command.

```
ndk-build APP_ABI=arm64-v8a APP_PLATFORM=android-28 clean
ndk-build APP_ABI=arm64-v8a APP_PLATFORM=android-28
```

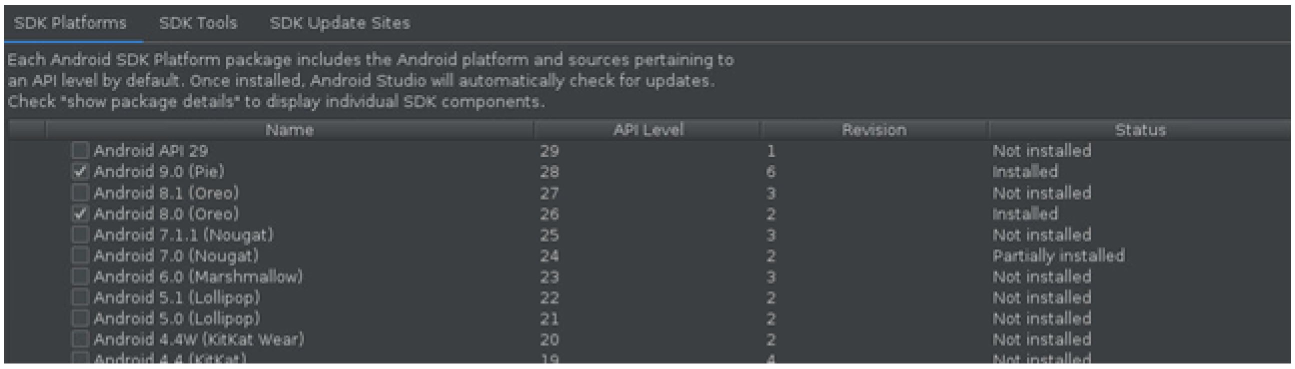
Create the Android Application with Android Studio

After following the steps in the `sdk/lib/README.android` file, follow these steps to complete the set up of the Android build environment.

1. Start Android Studio and open the SDK Manager.
   a. Click **File**, then **Settings**.
   b. Expand **Appearance & Behavior**, then expand **System Settings**.
   c. Click **Android SDK**.
2. From the SDK Manager, select Android 9.0 (Pie).

   If you plan to develop for more than one Android release, select that release as well.

Figure 2-1 SDK Manager

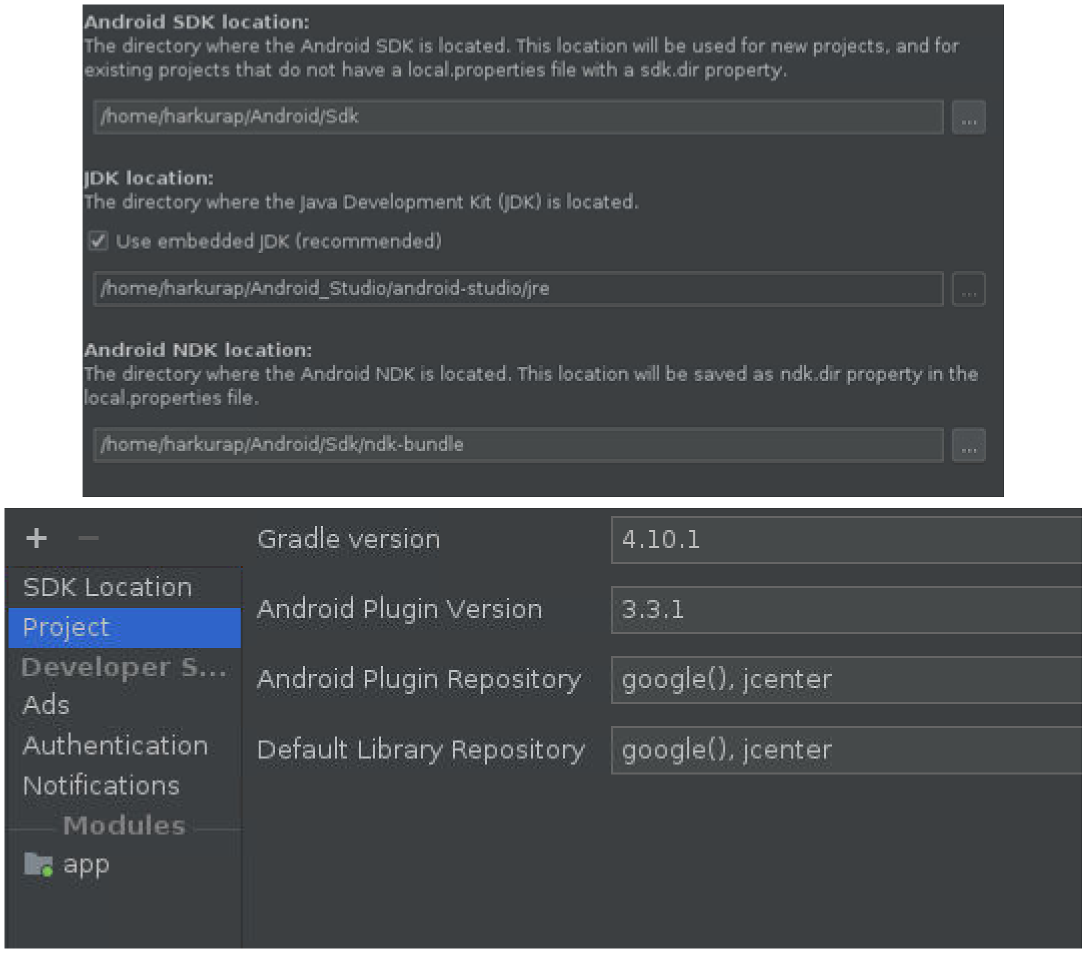


Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, Android Studio will automatically check for updates. Check "show package details" to display individual SDK components.

	Name	API Level	Revision	Status
☐	Android API 29	29	1	Not installed
✓	Android 9.0 (Pie)	28	6	Installed
☐	Android 8.1 (Oreo)	27	3	Not installed
✓	Android 8.0 (Oreo)	26	2	Installed
☐	Android 7.1.1 (Nougat)	25	3	Not installed
☐	Android 7.0 (Nougat)	24	2	Partially installed
☐	Android 6.0 (Marshmallow)	23	3	Not installed
☐	Android 5.1 (Lollipop)	22	2	Not installed
☐	Android 5.0 (Lollipop)	21	2	Not installed
☐	Android 4.4W (KitKat Wear)	20	2	Not installed
☐	Android 4.4 (KitKat)	19	4	Not installed

3. Confirm the locations and versions of the Android SDK and Gradle.

Figure 2-2 Android SDK and Gradle



Android SDK location:
The directory where the Android SDK is located. This location will be used for new projects, and for existing projects that do not have a local.properties file with a sdk.dir property.

/home/harkurap/Android/Sdk

JDK location:
The directory where the Java Development Kit (JDK) is located.

✓ Use embedded JDK (recommended)

/home/harkurap/Android_Studio/android-studio/jre

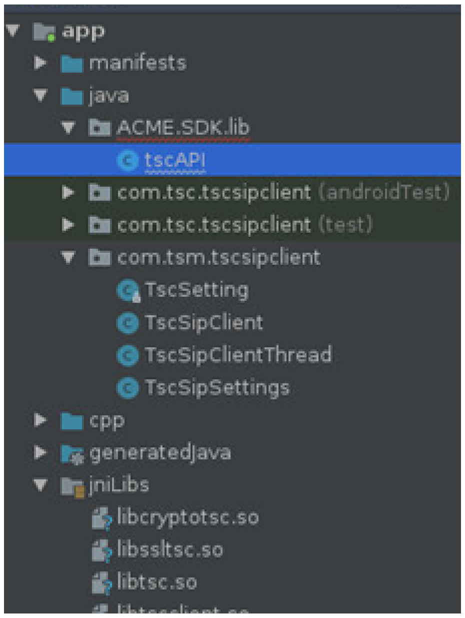
Android NDK location:
The directory where the Android NDK is located. This location will be saved as ndk.dir property in the local.properties file.

/home/harkurap/Android/Sdk/ndk-bundle

Gradle version	4.10.1
Android Plugin Version	3.3.1
Android Plugin Repository	google(), jcenter
Default Library Repository	google(), jcenter

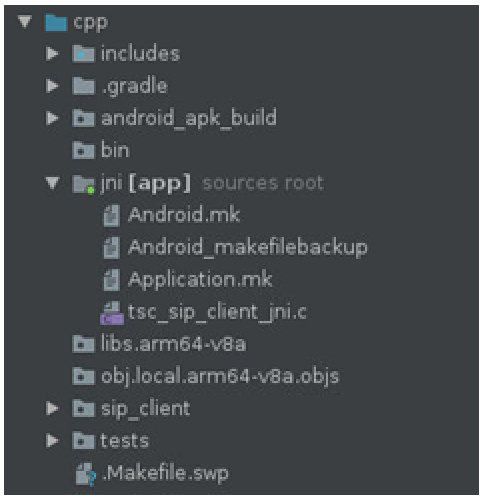
4. Create an Android project Tscsipclient using the Java files available in the `sdk/apps/tsc_sip/tsc_sip_client/android_apk_build/src/main/java`.

Figure 2-3 Android Project



5. Add the JNI file `sdk/apps/tsc_sip/tsc_sip_client/jni/tsc_sip_client_jni.c`.

Figure 2-4 JNI File

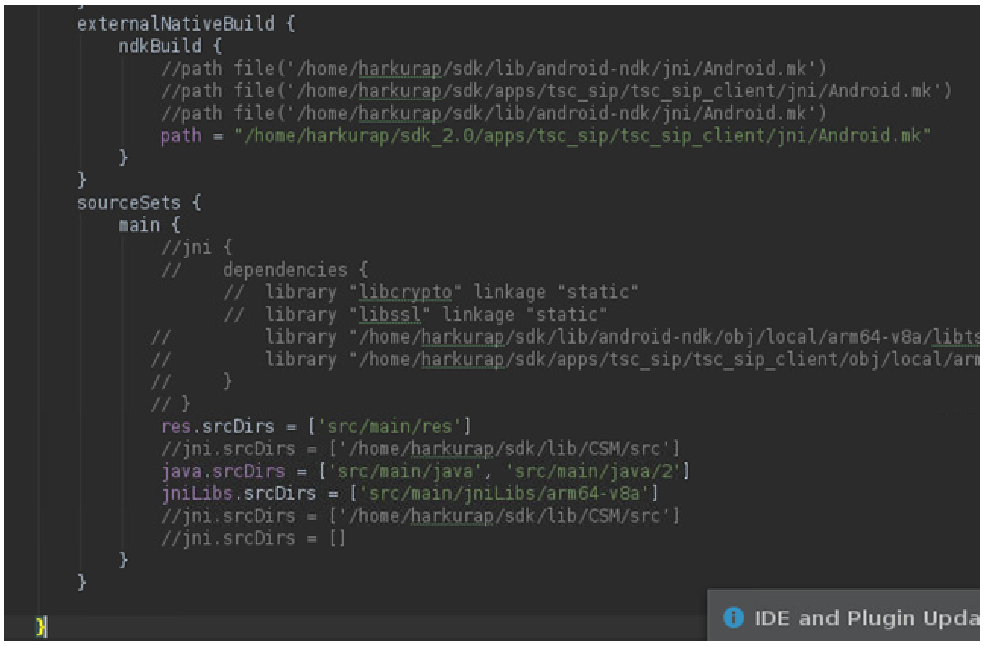


6. Verify your `build.gradle` file looks similar to the following:

```
apply plugin: 'com.android.application'
android {
        compileSdkVersion 28
        defaultConfig {
                applicationId "com.tsm.tscsipclient"
                minSdkVersion 26
                targetSdkVersion 28
                versionCode 1
                versionName "1.0"
                testInstrumentationRunner
"android.support.test.runner.AndroidJUnitRunner"
        }
```

7. Update the location of the tsc_sip_client folder and set the ANDROID_NDK variable to point to `sdk/apps/tsc_sip/tsc_sip_client/jni/Android.mk`

Figure 2-5 build.gradle



```
    externalNativeBuild {
        ndkBuild {
            //path file('/home/harkurap/sdk/lib/android-ndk/jni/Android.mk')
            //path file('/home/harkurap/sdk/apps/tsc_sip/tsc_sip_client/jni/Android.mk')
            //path file('/home/harkurap/sdk/lib/android-ndk/jni/Android.mk')
            path = "/home/harkurap/sdk_2.0/apps/tsc_sip/tsc_sip_client/jni/Android.mk"
        }
    }
    sourceSets {
        main {
            //jni {
            //    dependencies {
                  //  library "libcrypto" linkage "static"
                  //  library "libssl" linkage "static"
           //         library "/home/harkurap/sdk/lib/android-ndk/obj/local/arm64-v8a/libts
           //         library "/home/harkurap/sdk/apps/tsc_sip/tsc_sip_client/obj/local/arm
           //     }
           // }
            res.srcDirs = ['src/main/res']
            //jni.srcDirs = ['/home/harkurap/sdk/lib/CSM/src']
            java.srcDirs = ['src/main/java', 'src/main/java/2']
            jniLibs.srcDirs = ['src/main/jniLibs/arm64-v8a']
            //jni.srcDirs = ['/home/harkurap/sdk/lib/CSM/src']
            //jni.srcDirs = []
        }
    }

}
```

```
export ANDROID_NDK=sdk/apps/tsc_sip/tsc_sip_client/jni/Android.mk
```

8. Copy the following libraries into the `jniLibs` folder.
   - libcryptotsc.so
   - libssltsc.so
   - libtsc.so
   - libtscclient.so

```
cp sdk/apps/tsc_sip/tsc_sip_client/libs/arm64-v8a/*.so android-ndk-r20/
sources/third_party/vulkan/src/build-android/jniLibs
```

9. Build the SDK and then copy the generated APK file to an Android 9.x phone.

NOT_SUPPORTED:

The Android phone should be set to 'Developer Mode' with 'USB debug mode' enabled.

Note:

View the Logcat in Android Studio to see process information from the connected device.

iOS Development

Use the following method when developing for iOS.

Set up iOS Environment

1. On your Mac machine, install XCode IDE and the iOS SDK for 12.x.

   Note:

   An Apple ID is required to develop iOS applications.

2. Extract the nnTSC200m1.tar.gz file.
3. Download OpenSSL 1.1.1a and place it in the extlib folder.

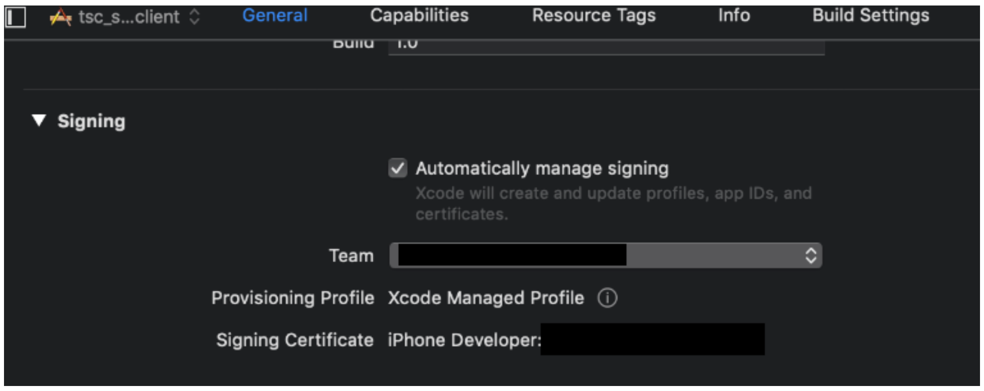
Build TSC SDK Libraries for iOS

1. Execute the `sdk/exlib/build_ioslib.sh` script to compile the OpenSSL libraries.
2. Execute the `sdk/lib/build_ioslib.sh` script to compile the TSC SDK library.
3. Verify libtsc.a compiled for the arm64 architecture.

   ```
   xcrun --sdk iphoneos lipo -info libtsc.a
   ```

4. Open the project `sdk/apps/tsc_sip/tsc_sip_client/ios_xcode_app/tsc_sip_client.xcodeproj` in XCode IDE.
5. Click **Target**, and then **General** to provide credentials for certificate signing.

Figure 2-6 General Tab

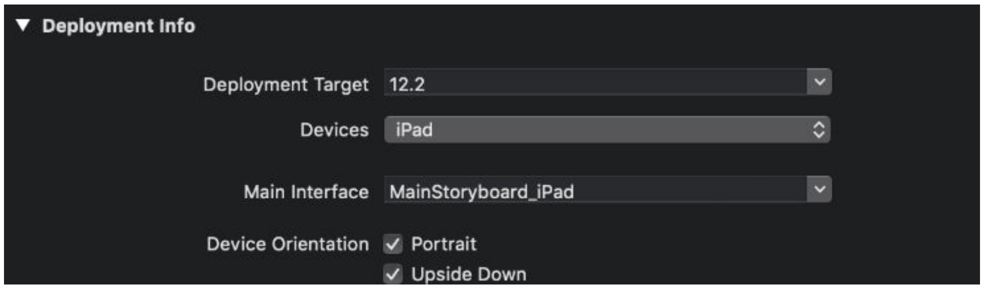


☑ Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.

Team [redacted]
Provisioning Profile Xcode Managed Profile
Signing Certificate iPhone Developer: [redacted]

6. Click **General**, and then **Development Info** to set the build devices.

Figure 2-7 Build Devices

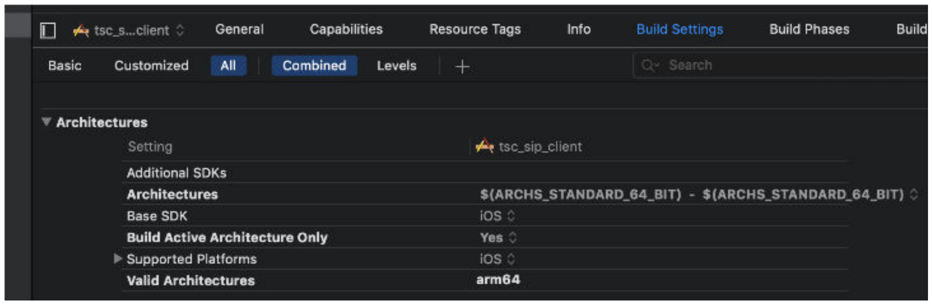


Deployment Info

Deployment Target	12.2
Devices	iPad
Main Interface	MainStoryboard_iPad
Device Orientation	✓ Portrait
	✓ Upside Down

7. Copy the libssl.1.1.dylib , libcrypto.1.1.dylib and libtsc.a libraries, which are compiled in earlier steps, to the `Libraries` subfolder under the ios_sdk_app directory or the project directory.
8. Click **Build Settings**, and then **All**:
   a. Verify that arm64 is listed as a valid architecture.

Figure 2-8 Build Settings



Architectures

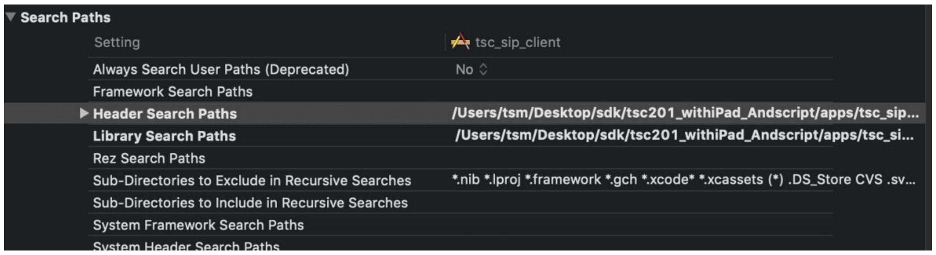
Setting	tsc_sip_client
Additional SDKs	
Architectures	$(ARCHS_STANDARD_64_BIT) - $(ARCHS_STANDARD_64_BIT)
Base SDK	iOS
Build Active Architecture Only	Yes
Supported Platforms	iOS
Valid Architectures	arm64

   b. Verify the library search and header search paths.

   If the project is run directly from the tar file, append these paths:

   - ${project_dir}/../../../../lib/OSAA/include
   - ${project_dir}/../../../../lib/include

Figure 2-9 Search Paths

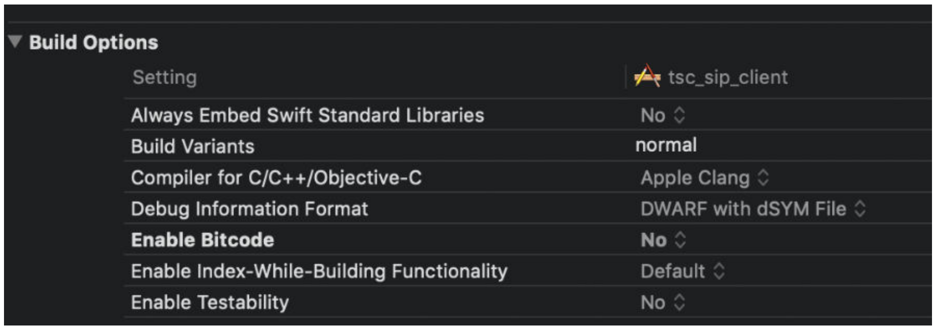


Search Paths

Setting	tsc_sip_client
Always Search User Paths (Deprecated)	No
Framework Search Paths	
Header Search Paths	/Users/tsm/Desktop/sdk/tsc201_withiPad_Andscript/apps/tsc_sip...
Library Search Paths	/Users/tsm/Desktop/sdk/tsc201_withiPad_Andscript/apps/tsc_si...
Rez Search Paths	
Sub-Directories to Exclude in Recursive Searches	*.nib *.lproj *.framework *.gch *.xcode* *.xcassets (*) .DS_Store CVS .sv...
Sub-Directories to Include in Recursive Searches	
System Framework Search Paths	
System Header Search Paths	

   c. Verify **Enable Bitcode** is set to **No**.

Figure 2-10 Disable Bitcode



Setting	tsc_sip_client
Always Embed Swift Standard Libraries	No
Build Variants	normal
Compiler for C/C++/Objective-C	Apple Clang
Debug Information Format	DWARF with dSYM File
Enable Bitcode	**No**
Enable Index-While-Building Functionality	Default
Enable Testability	No

d. Click **Build Phases**, and then **Link Binary with Libraries** to verify that libtsc.a is added.

e. Click **Build Phases**, and then **Copy Files** to verify that libcrypo.1.1.dylib and libssl.1.1.dylib are added.

f. Click **Product**, and then **Destination** and set the destination to the connected iPhone or iPad device.

   The connected iPhone or iPad should have the updated iOS SDK.

g. Click **Product**, and then **Run** to build the application.

Generate the API Documentation

1. Navigate to the `lib` folder.

   ```
   cd sdk/lib
   ```

2. Generate the API documentation.

   ```
   doxygen ../docs/doxygen/doxygen.conf
   ```

3. Open the documentation using a browser pointing to `sdk/docs/html/index.html`.

3

Accessing and Using the TSM SDK APIs

Sample TSM SDK-based Applications

A number of small application templates are found throughout the apps directory. Each of these files contains a small, well-defined set of functionality that enables a software developer to easily understand its implementation via the TSCF client-side SDK.

```
SDK:
+---apps/tsc_sip/tsc_sip_client
|   +---tsc_sip_client.c           Provides a basic SIP client utilizing
                                   TCP/TLS/UDP/DTLS for tunnel transport
                                   of inner UDP sockets. Supports Linux,
                                   Windows, and Android operating systems.
+---apps/tsc_sip/tsc_sip_server
|   +---tsc_sip_server.c           Provides a basic SIP server utilizing
                                   TCP/TLS/UDP/DTLS for tunnel transport
                                   of inner UDP sockets. Supports Linux,
                                   Windows, and Android operating systems.
+---apps/tsc_sip/tsc_sip_inner_tcp
|   +---tsc_sip_inner_tcp.c        Provides a basic SIP client utilizing
                                   TCP for tunnel transport and
                                   demonstrating usage of TCP sockets
                                   for applications such as HTTP --
                                   supports Linux operating systems.
```

All of the above files contain extensive comments making it an easy task to navigate through the code. Using tsc_sip_client.c as an example, you can readily proceed through the file.

1. Search for tsc_ctrl_init () and examine the code immediately following this function for the details of tunnel initialization.
2. Search for Create a Tunnel and examine the code immediately following for the details of tunnel creation and the configuration exchange between the TSCF server and client.
3. Search for SIP SOCKET CREATION and examine the code that creates and binds TSCF sockets.
4. Search for REGISTER TRANSACTION and examine the code that builds a SIP REGISTRAR request and processes the REGISTRAR response.
5. Search for INVITE TRANSACTION and examine the code that builds a SIP INVITE request and processes the REGISTRAR response.
6. Search for Build and send ACK to examine ACK creation and processing code.
7. Search for RTP Exchange to examine RTP code.
8. Search for BYE TRANSACTION to find code that terminates a SIP connection.
9. Search for TEST DONE to find code that terminates a tunnel.

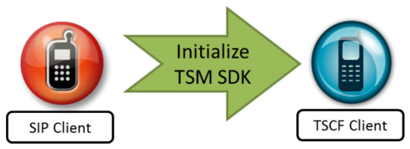
Using The SDK To Create A TSM Tunnel

The following steps provide an outline on integrating a SIP client (in this example tsc_sip_client) with the TSM SDK. Please refer to the file tsc_sip_client.c (located at sdk/apps/tsc_sip/tsc_sip_client) which contains working code references on establishing a TSM tunnel and making a SIP/RTP based call.

Initialize the TSCF-Client side API

Use the TSCF Server IP address, port, transport type, wireshark tracing and certificate parameters (if using TLS/DTLS) to initialize the client. Populate the required information in a tsc_tunnel_params type structure.

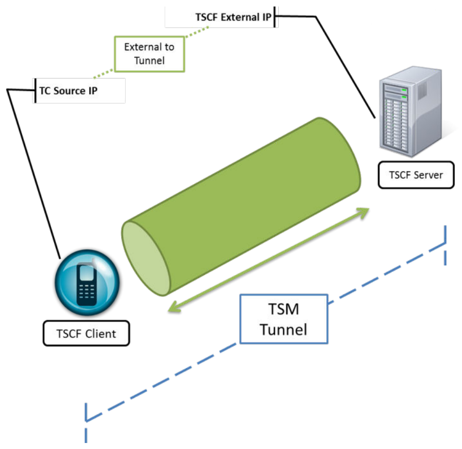
Search for "tsc_ctrl_init()" in the reference file for actual code implementation.



Create a TSM Tunnel

Once initialized, create a TSM tunnel between the TSCF client and server, register for callbacks and obtain the SIP server IP address.

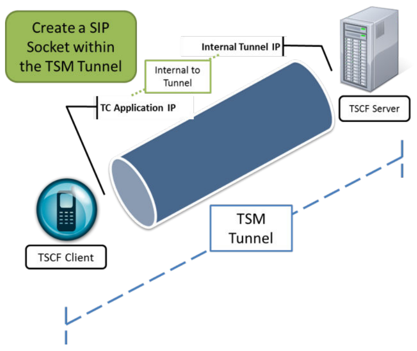
Search for the text "Create a tunnel" in the reference file for actual code implementation.



Create a SIP Socket within the TSM Tunnel

With the TSM tunnel established, create a SIP socket and bind the local address assigned by the TSCF server to it.

Search for the text "SIP SOCKET CREATION" in the reference file for actual code implementation.



Send SIP messages over the TSM Tunnel

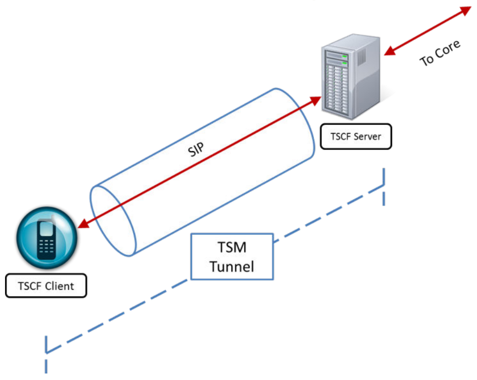
You can now send SIP messages over the TSM tunnel. Based on the kind of SIP applications, you may need to send a REGISTER message or directly initiate a peer-to-peer call through an INVITE message.

You can find examples of both messages being sent by looking for the following text. Since this is only sample code, please use the same as reference for sending and receiving SIP messages via the TSM tunnel.

REGISTER TRANSACTION: Constructs and sends a REGISTER message to the tsc_sip_server. The code currently doesn't check for a 200 OK message.

INVITE TRANSACTION: Constructs and sends an INVITE message and handles a 200 OK.

Build and send ACK (to INVITE transaction) : This code builds and sends an ACK to the INVITE received.

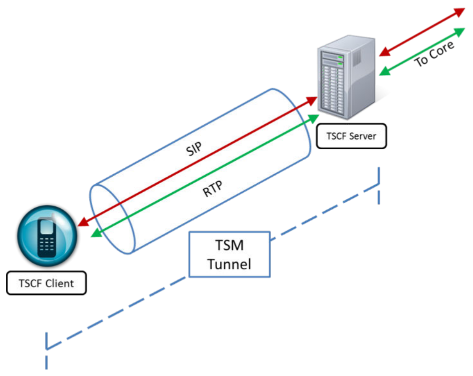


Send RTP message over the TSM Tunnel

As part of the call, there is also sample code to send and receive RTP packets over the TSM tunnel. Search for the text "RTP EXCHANGE" within the reference file for a code implementation.

The code builds RTP packets purely for the purpose of simulation.

At this point you should be able to make calls through the TSM tunnel.



Terminate a SIP call over the TSM Tunnel

After you are done with the call, you may terminate it using sample code in the reference file. Search for the text "BYE TRANSACTION".

Terminate a TSM Tunnel

If your application has determined that it doesn't require the use of a TSM tunnel, you may terminate the tunnel using the sample code. Search for "Finished, Socket, Tunnel Cleanup".

Enabling Redundancy

TSM enables an application to improve media quality under adverse network packet loss through the tunnel redundancy feature.

Once enabled, tunnel redundancy comes with two options for TCP/TLS connections:

1. Original TSM tunnel + 1 redundant tunnel
2. Original TSM tunnel + 2 redundant tunnels

And two options for UDP/DTLS connections:

1. Original TSM packet + 1 redundant packet
2. Original TSM packet + 2 redundant packets

If the redundancy factor is set to 2 for a UDP connection, then duplicates of the two previous packets are sent with each packet.

1. Send packet 1
2. Send packet 2 + duplicate of packet 1
3. Send packet 3 + duplicate of packet 2 + duplicate of packet 1
4. Send packet 4+ duplicate of packet 3+ duplicate of packet 2
5. Send packet 5+ duplicate of packet 4+ duplicate of packet 3

Increasing the number of tunnels increases media quality at the cost of network bandwidth. Decide what kind of redundancy you require before enabling this feature.

Tunnel redundancy can be enabled on a per-socket basis.

1. Create a notification handler function. Once requested, the TSCF notifies the application whether the redundancy was enabled successfully.

2. Set a socket option with the type of redundancy factor (1 or 2).

Search for the text “RTP socket created” and "TSC_REDUNDANCY" in the reference file for actual code implementation that creates redundant tunnels for RTP packets.

Error Codes

The following are error codes you can check against when calling the TSCF layer to ensure you can handle all success and failure scenarios:

Return Value	TSC_ERROR_CODE
0	tsc_error_code_ok
1	tsc_error_code_error
2	tsc_error_code_not_logged
3	tsc_error_code_cannot_connect
4	tsc_error_code_cannot_configure
5	tsc_error_code_keepalive_failure
6	tsc_error_code_service_failure
7	tsc_error_code_cannot_recv_data
8	tsc_error_code_no_data
9	tsc_error_code_cannot_send_data
10	tsc_error_code_authenticate
11	tsc_error_code_cannot_release
12	tsc_error_code_queue_overflow