Qualcomm Technologies, Inc.

Telematics SDK

User Guide
v1.46.35

80-PF458-1 Rev. H

February 11, 2022

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Revision History

Date	Description
Sep 2017	Initial Release
Dec 2017	Added subscription feature
Jun 2018	Added data services apps
Oct 2018	Update of Yocto Platform SDK instructions, Addition of Network Selection and Service System Manager
Nov 2018	Addition of C-V2X samples
Jan 2019	Addition of Audio Manager apps reference code
Mar 2019	Addition of Thermal Manager apps
May 2019	Addition of TCU Activity Management feature
Jun 2019	Addition of Audio play, capture apps reference code
Jul 2019	Updated the Location APIs apps
Jul 2019	Addition of Thermal shutdown manager APIs and call flows
Sep 2019	Addition of Remote SIM feature
Sep 2019	Addition of Audio APIs for Loopback, Tone Generator and related call flows
Sep 2019	Addition of Audio APIs for compressed audio format playback and related call flows
Sep 2019	Addition of modem config APIs and related call flows
Oct 2019	Addition of Data Filter APIs and call flows
Oct 2019	Addition of Location concurrent report APIs and call flows
Oct 2019	Addition of Location constraint time uncertainty APIs and call flows
Nov 2019	Addition of Audio Format Transcoding APIs and call flows
Nov 2019	Addition of Data Networking APIs and call flows
Nov 2019	Addition of Compressed audio format playback on voice paths APIs and call flows
Jan 2020	Addition of Data software bridge management APIs and call flows
Jan 2020	Addition of Socks Proxy to Data Networking APIs and call flows
Feb 2020	Addition of Location Configurator APIs and call flows
Mar 2020	Addition of Robust Location API in Location Configurator and call flows
May 2020	Addition of L2TP Feature to Data Networking APIs and call flows
Jan 2021	Addition of Location SDK Simulation libraries
Jan 2021	Updated Data Subsystem Readiness flow for new methodology
Feb 2021	Addition of new sub system Readiness APIs for Audio.
Apr 2021	Addition of new serving system APIs for Data.
Apr 2021	Addition of documentation for sensor subsystem.
Jun 2021	Addition of documentation for remote SIM provisioning APIs in Telematics SDK

Contents

1 Introduction

1.1 Purpose

This document serves as a User Guide for Telematics SDK APIs.

1.2 Scope

This document provides details on how to use Telematics SDK APIs to build stand-alone applications on Linux based Automotive platforms.

It contains information that depicts the usage of the APIs and demonstrate different use case scenarios through a set of sample applications such as make_call, make_ecall, send_sms, receive_sms, command_callback, make_audio_voice_call etc.

This document is intended for software developers who will be using the Telematics SDK.

This document assumes that the developers are familiar with Linux and C++11 programming.

2 Building Yocto Platform SDK

This section has instructions to build a yocto platform SDK using code aurora forum (CAF) / open source

- These instructions are applicable for QTI Processor builds only, **For external APs the platform SDK corresponding to specific external AP would have to be used.**

NOTE: This is not to be confused with the Telematics SDK. The Yocto platform SDK, includes the tool chain, and libraries necessary to be able to develop any program for a given device. It also includes the stub open source libraries for the Telematics SDK, allowing one to develop application using the Telematics SDK's APIs as well.

You need to have the following in order to proceed:

- Linux Ubuntu 14.04
- Install required packages

```
1 $ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib \
2 build-essential chrpath socat libsdl1.2-dev xterm
```

2.1 Get Started

- Sync the CAF build
  repo init and sync the sources

```
1 $ repo init -q -u git://codeaurora.org/quic/le/le/manifest.git -b release -m <caf_ insert label here. xml>
2 $ repo sync -j 32
```

  For example: Using AU label
  repo init -q -u git://codeaurora.org/quic/le/le/manifest.git -b release -m caf_AU_LINUX_EMBEDDED_LE.UM.4.1.1_RB1_TARGET_ALL.01.147.055.xml
  or using CRM build-id label
  repo init -q -u git://codeaurora.org/quic/le/le/manifest.git -b release -m LE.UM.4.1.1-05510-sa515m.xml
  NOTE: AU_LINUX_EMBEDDED_LE.UM.4.1.1_RB1_TARGET_ALL.01.147.055 should be replaced to the AU tag corresponding to the desired device build. "caf_" needs to be prefixed to the AU label.
  The tags are listed here -
  https://source.codeaurora.org/quic/le/le/manifest/refs/tags
- Update Telux and Telematics SDK
  Add the following lines at the end of poky/build/conf/local.conf:
  For all targets except LE.UM.1.3.r5, use

```
1 CORE_IMAGE_EXTRA_INSTALL += "telux"
2 CORE_IMAGE_EXTRA_INSTALL += "telux-lib"
```

  NOTE: For LE.UM.1.3.r5 target, use below.

```
1 CORE_IMAGE_EXTRA_INSTALL += "telux"
2 CORE_IMAGE_EXTRA_INSTALL += "telephony-lib"
```

- Setup the build environment
  Source the bitbake environment
  ```
  1 $ cd poky/
  2 $ source build/conf/set_bb_env.sh
  ```
  Set MACHINE and DISTRO values
  For LE.UM.3.2.1, Use:
  ```
  1 $ export MACHINE=sa415m
  2 $ export DISTRO=auto
  ```
  For LE.UM.4.1.1, Use:
  ```
  1 $ export MACHINE=sa515m
  2 $ export DISTRO=auto
  ```
  For LE.UM.3.2.3, Use:
  ```
  1 $ export MACHINE=sa2150p
  2 $ export DISTRO=msm
  ```
- Build and install the Yocto Platform SDK
  Run the following command to create a yocto platform sdk
  ```
  1 $ bitbake core-image-minimal -c do_populate_sdk
  ```
  NOTE: core-image-minimal provides least number of packages including telematics sdk, sufficient for allowing one to develop application using the telematics sdk API's. Any additional packages necessary for application development might need to be added in yocto platform (bitbake) before the build.
- Location of Yocto Platform SDK
  After successful compilation, you will have have sdk installer on this location
  ```
  1 poky/build/tmp-glibc/deploy/sdk/oecore-x86_64-<arch_name_here>-toolchain-nodistro.0.sh
  ```
  For LE.UM.3.2.1, it will be generated with name
  poky/build/tmp-glibc/deploy/sdk/oecore-x86_64-armv7at2hf-neon-toolchain-nodistro.0.sh

3 Using SDK APIs in simulation mode

Certain APIs in the SDK can be used in simulation mode, on any Linux based platform. This enables developers to develop and test their SDK applications on desktop environments, thus providing a simpler and faster development experience. This is achieved using the run time simulation libraries of the SDK.

This section has instructions to build the simulations of SDK APIs using CAF repository.

You need to have the following in order to proceed:

- Linux Ubuntu 14.04, Linux Ubuntu 20.04
- GCC (version > 9.3)

3.1 Get Started

- Sync the CAF build
  repo init and sync the sources
  ```
  1 $ repo init -q -u git://codeaurora.org/quic/le/le/manifest.git -b release
  2   -m <caf_ insert label here. xml>
  3 $ repo sync -j 32
  ```
  For example: Using AU label
  repo init -q -u git://codeaurora.org/quic/le/le/manifest.git -b release -m caf_AU_LINUX_EMBEDDED_LE.UM.4.1.1_RB1_TARGET_ALL.01.147.055.xml
  or using CRM build-id label
  repo init -q -u git://codeaurora.org/quic/le/le/manifest.git -b release -m LE.UM.4.1.1-05510-sa515m.xml
  NOTE: AU_LINUX_EMBEDDED_LE.UM.4.1.1_RB1_TARGET_ALL.01.147.055 should be replaced to the AU tag corresponding to the desired device build. "caf_" needs to be prefixed to the AU label.
  The tags are listed here -
  https://source.codeaurora.org/quic/le/le/manifest/refs/tags
- The following sections describe the way to compile these simulations for the respective libraries/APIs. All the simulation related source is located under "telux/public/simulation/" directory. Simulation is not available for all the SDK APIs and it is work in progress.

3.2 Location Simulation

The "telux/public/simulation/loc" folder contains the simulation implementation of Location Telematics SDK APIs.

The following simulations have been compiled and tested with GCC and G++ version 9.3.0 and Ubuntu versions 14.04, 20.04

- Navigate into the simulation folder of telux git project.
```
1 $ cd telux/public/simulation/
```
- The following instructions build and install the location simulation libraries.
  Run below instructions in telux/public/include folder
```
1 $ cmake .
2 $ cmake --build .
3 $ sudo make install
```
  Run below instructions in telux/public/simulation
```
1 $ cmake .
2 $ cmake --build .
3 $ sudo make install
```
- Check where the libraries are installed and ensure that it is included in the library path of the system environment.
  Note: Modify CMakeLists.txt in snaptel-sdk/apps folder to include only tests/location_test/app subdirectory Run below instructions in telux/public/apps
```
1 $ cmake .
2 $ cmake --build .
3 $ sudo make install
```
- After successful compilation, the location_test_app is installed into the system's default directory and is ready to be launched.

3.2.1 Configuration options

The location simulation is designed to be configurable as per the needs of the developer to allow various testing scenarios. The simulation reports can be generated using one of the two methods: Fixed canned data - which are pre-defined as default values in some headers, pre-recorded location data taken from a file such as PRE-RECORDED_LOCATION_DATA.csv. Simulation libraries also allow the developers to configure the subsystem readiness and callback delays from a config file. The motivation for allowing such flexibility is to allow developers to run various test case scenarios even without the hardware access. The following are the three important files that have configuration and default values of the location simulation libraries.

- 1) StubConfig : Allows users to configure delays and other aspects of simulation libraries.
- 2) CommonDef.hpp : Defines default values for the some of the fields in location reports.
- 3) StubDefines.hpp: Defines default values for the subsytem readiness and callback delays.

4 Steps to writing an App

This section has instructions to create a sample application using Telematics APIs.

You need to have the following in order to proceed:

- Linux Yocto Platform SDK Installer created by the system integrator or created using section 2 (Building Platform SDK)

4.1 Setup the build environment

Install the Linux platform SDK created by the platform developer for the intended device. Let's assume that the installer is named sdk_installer.sh.

- Open the terminal, Copy the sdk_installer.sh into your working directory, say for example "my_sdk_install".
```
1 $ mkdir my_sdk_install; cd my_sdk_install
2 $ ./sdk_installer.sh
3 // Choose to install to my_sdk_install when the SDK installer asks
```
- Setup the build environment as follows:
```
1 $ cd my_sdk_install; source environment-setup-cortexa8hf-vfp-neon-oe-linux-gnueabi
```

Now your development environment is set for the terminal, start your development.

4.2 To get started with a sample app

- Create a sample folder
```
1 $ mkdir sample_app
```
- Create a sample application as follows:
```
//  FileName:      SampleApp.cpp
//  Description:   Sample program to check the status of telux::tel::PhoneManager.

#include <iostream>
#include <memory>

#include <telux/tel/PhoneFactory.hpp>

// This is sample program to get an instance of PhoneManager
// and check for telephony sub system status

int main() {

   // Get the PhoneFactory and PhoneManager instances
   auto &phoneFactory = telux::tel::PhoneFactory::getInstance();
   auto phoneManager = phoneFactory.getPhoneManager();

   // Check if telephony subsystem is ready
   bool status = phoneManager->isSubsystemReady();
   std::cout << "PhoneManager Ready? " << (status? "Yes":"No") << std::endl;
```

```
    // If telephony subsystem is not ready, wait for it to be ready
    if(!status) {
       std::cout << "wait unconditionally for it to be ready " << std::endl;
       std::future<bool> f = phoneManager->onSubsystemReady();
       // If we want to wait unconditionally for telephony subsystem to be ready
       status = f.get();
    }
    std::cout << "PhoneManager Ready? " << (status? "Yes":"No") << std::endl;

    // Now the application ready for SDK services like Phone, SMS, CardServices
    return 0;
}
```

4.3 Compile the program

- Now compile your source code

```
1 user@machine:/sample_app$ ${CC} SampleApp.cpp -ltelux_tel -std=c++11 -lstdc++ -o sample_app
```

4.4 Install program on the device

- Push your binary to the /data location on the device

```
1 user@machine:/sample_app$ adb push sample_app /data/
2 user@machine:/sample_app$ adb shell
3 $ cd data
4 $ chmod +x sample_app
5 $ ./sample_app
```

5 Sample Apps

This section has instructions to run the sample applications that come with the Telematics SDK. Sample applications use the cmake build system. You need to have cmake version 2.8.9 or later on your host machine

5.1 Source the environment

- Source the build environment:

```
1 $ cd my_sdk_install; source environment-setup-cortexa8hf-vfp-neon-oe-linux-gnueabi
```

Now your development environment is set for the terminal, start your development.

5.2 Compile sample apps

- Goto telux/samples directory and create a build folder

```
1 $ cd telux/samples
2 $ mkdir -p build; cd build
```

- Compile all the samples:

```
$ cmake -DBUILD_ALL_SAMPLES:BOOL=ON -DCMAKE_INSTALL_PREFIX=./install ..
$ make clean && make install
```

Compile telsdk_console_app:

- Goto respective sample program for compiling individual programs

```
1 $ cd samples/telsdk_console_app
2 $ mkdir -p build; cd build
3 $ cmake -DCMAKE_INSTALL_PREFIX=./install ..
4 $ make clean && make install
```

5.3 Install apps on the device

- Push your binary to the /data location on the device

```
1 # this install all the sample apps
2 user@machine:/build/install/bin$ adb push . /data/
3 user@machine:/build/install/bin$ adb shell
4 # cd data
5 # ./telsdk_console_app
6
7 # to install telsdk_console app
8 user@machine:telsdk_console_app/build/install/bin$ adb push . /data/
```

The following sections provide further information

- Sample Applications Configuration

- Configuring Logs from the SDK
- audio
- common
- cv2x
- tel
- data
- loc
- config
- power
- thermal
- sensor

5.4 Sample Applications Configuration

Sample Applications Configuration

Telematics stand-alone applications like make_call_app, send_sms_app etc provides flexibility to configure application config parameters using either user defined config file (ex: appName.conf) or default config file (sample_app.conf).

Configuration file has application configurations in key-value pair. For make_call_app sample application user can provide dial number in the configuration file in order to make a voice call.

DIAL_NUMBER = +1234512345

For send_sms_app sample application user can provide receiver's phone number and text message in the configuration file in order to send a SMS.

RECEIVER_NUMBER = +1234512345

MESSAGE = Text message

Below are the steps to run the sample application.

- Configure required parameters either in user defined config file (ex: appName.conf) or default config file.
- User provided config file should be placed where application is running.
- Execute below command to use user defined config
  ./make_call_app appName.conf
- Execute below command to leverage either default config file if present or use configuration defined in application itself.
  ./make_call_app

5.5 Configuring Logs from the SDK

Configuring Logs from the SDK

Please follow below steps to configure Logger settings.

Telematics SDK provides a configurable logger module that can be used to log messages from Telematics SDK library and applications at desired threshold levels into device console, diag and optionally into a log file.

By default, *tel.conf* will be placed under /etc location

The configuration file called "appName.conf" or "tel.conf" is used to configure logger settings such as logging threshold, enable/disable file logging and to change the log file name. These file have to be updated to override default behavior. These configuration file should be copied either in /etc or the folder where the application is running.

To modify tel.conf file under /etc, you need to mount partition on MDM A7 processor

```
1 adb shell mount -o rw,remount /
```

NOTE: The file path where the log file will be written to, need to be in a writable partition, accessible to the application that is running.

In the case of MDMs A7 processor the /data partition is writable.

Here is how the platform searches for the configuration file. If configuration file is found use the same to configure logger settings else keep continue to search in below order.

- Search for appName.conf in /etc folder. (i.e. telsdk_console_app.conf)
- Search for appName.conf in the folder that contains the application.
- Search for tel.conf in etc folder.
- Search for tel.conf in the folder that contains the application.

This allows flexibility for app's to either share the same log file or keep each apps log file separate.

1. Console and file level logging

CONSOLE_LOG_LEVEL, FILE_LOG_LEVEL specifies the threshold for console log messages. Possible LOG_LEVEL values are NONE, PERF, ERROR, WARNING, INFO, DEBUG

```
1 # NONE - No logging.
2 # PERF - Prints messages with nanoseconds precision timestamp.
3 # ERROR - Very minimal logging.Prints perf and error messages only.
4 # WARNING - Prints perf, error and warning messages.
5 # INFO - Prints perf, errors, warning and information messages.
6 # DEBUG - Full logging including debug messages.It is intended for debugging purposes only.
7
8 CONSOLE_LOG_LEVEL=INFO
9 FILE_LOG_LEVEL=DEBUG
10 DIAG_LOG_LEVEL=DEBUG
```

NOTE: For an applicaiton to be able to log to the tel.log file, it should have "system" linux group permissions.

2. Diag level logging

DIAG_LOG_LEVEL specifies the threshold for logs messages displayed in QXDM. Possible LOG_LEVEL values are NONE, PERF, ERROR, WARNING, INFO, DEBUG. The mapping of SDK log levels to QXDM log levels in shown below:

```
1 # SDK Log Levels --> QXDM LOG Levels
2 # PERF --> FATAL (MSG_LEGACY_FATAL)
3 # ERROR --> ERROR (MSG_LEGACY_ERROR)
4 # WARNING --> HIGH (MSG_LEGACY_HIGH)
5 # INFO --> MED (MSG_LEGACY_MED)
6 # DEBUG --> LOW (MSG_LEGACY_LOW)
```

NOTE: For an applicaiton to be able to log to the Diag, it should have "diag" linux group permissions.

3. Log filtering

TELUX_LOG_COMPONENT_FILTER allows one or more whitelist which SDK technology domain should be logged

```
1 # 0 - All logs are printed.
2 # 1 - Audio logs are printed.
3 # 2 - CV2X logs are printed.
4 # 3 - Data logs are printed.
5 # 4 - Location logs are printed.
6 # 5 - Power logs are printed.
7 # 6 - Telephony logs are printed.
8 # 7 - Thermal logs are printed.
9
10 # For logging all component
11 # use TELUX_LOG_COMPONENT_FILTER= 0
12
13 # For logging more than one component like cv2x and audio (comma separated)
14 # use TELUX_LOG_COMPONENT_FILTER= 2,1
```

4. Set Max file size

MAX_LOG_FILE_SIZE specifies the maximum allowed size(in bytes) of the log file When the log file reaches its maximum size, it is saved as tel.log.backup.

- If the log file again reaches the max, it will be saved again overwriting the previous tel.log.backup file.
- So at a given time only one tel.log and tel.backup will exist in system.
- Default MAX_LOG_FILE_SIZE is 5 Mega Bytes.

```
1 MAX_LOG_FILE_SIZE=5242880
```

5. Prefix date and time for the log message

Used to prefix date and time on every log Message

```
1 # FALSE - logs with filename and line number, this is default option
2 # TRUE - logs with date, time, filename and line number
3
4 LOG_PREFIX_DATE_TIME=TRUE
```

6. Set log file path

Specifies the path of the log file. In an external application processor, the path needs to be in a writable partition. If this default path does not exist in the system or it is not writable, this path needs to be updated accordingly.

```
1 LOG_FILE_PATH=/data/vendor/telsdk
```

7. Set log file name

Specifies the name of the log file to be used

```
LOG_FILE_NAME=tel.log
```

5.6 audio

The List of sample apps related to audio

- Audio Manager API
- Audio playback session
- Audio capture session
- Audio loopback session
- Audio tone generation
- Audio voice session start and stop
- Audio voice session volume/mute control
- Audio voice session device switch
- Using Audio Manager APIs to play DTMF tone in a voice call.
- Using Audio Manager APIs to detect DTMF tones in a voice call.
- Audio transcoding operation
- Compressed audio format playback
- Compressed audio format playback on voice paths

5.6.1 Audio Manager API

Audio Manager API Sample Reference

This Section demonstrates how to use the Audio Manager API for audio subsystem/stream operations.

1. Get the AudioFactory instance

```
#include "AudioFactory.hpp"
#include "AudioManager.hpp"

using namespace telux::common;
using namespace telux::audio;

// Globals
```

```
static std::shared_ptr<IAudioManager> audioManager;
static std::shared_ptr<IAudioVoiceStream> audioVoiceStream;
Status status;

auto &audioFactory = audioFactory::getInstance();
```

2. Get the AudioManager object and check for audio subsystem Readiness

```
std::promise<ServiceStatus> prom{};
//  Get AudioManager instance.
audioManager = audioFactory.getAudioManager([&prom](ServiceStatus serviceStatus) {
    prom.set_value(serviceStatus);
});
if (!audioManager) {
    std::cout << "Failed to get AudioManager object" << std::endl;
    return;
}

//  Check if audio subsystem is ready
//  If audio subsystem is not ready, wait for it to be ready
ServiceStatus managerStatus = audioManager->getServiceStatus();
if (managerStatus != ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "\nAudio subsystem is not ready, Please wait ..." << std::endl;
    managerStatus = prom.get_future().get();
}

//  Check the service status again.
if (managerStatus == ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "Audio Subsytem is Ready << std::endl;
} else {
    std::cout << "ERROR - Unable to initialize audio subsystem" << std::endl;
    return;
}
```

3. Query Supported Devices and Stream Types of Audio subsystem

Below methods provides details on supported Device Types and Stream Types

3.1 Query Supported Devices Types of Audio subsystem

```
//Callback to get supported device type details.
void getDevicesCallback(std::vector<std::shared_ptr<IAudioDevice>> devices, ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "getDevices() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    int i = 0;
    for (auto device_type : devices) {
        std::cout << "Device [" << i << "] type: "
            << static_cast<unsigned int>(device_type->getType()) << ", direction: "
            << static_cast<unsigned int>(device_type->getDirection()) << std::endl;
        i++;
    }
}

//Query Supported Device type details.
status = audioManager->getDevices(getDevicesCallback);
```

3.2 Query Supported Stream Types of Audio subsystem

```
//Callback to get supported stream type details.
void getStreamTypesCallback(std::vector<StreamType> streams, ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
```

```
        std::cout << "getStreamTypes() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    int i = 0;
    for (auto stream_type : streams) {
        std::cout << "Stream [" << i << "] type: " << static_cast<unsigned int>(stream_type)
            << std::endl;
        i++;
    }
}

//Query Supported stream type details.
status = audioManager->getStreamTypes(getStreamTypesCallback);
```

4. Create an Audio Stream (Voice Call Session)

```
//Callback which provides response to createStream, with pointer to base interface IAudioStream.
void createStreamCallback(std::shared_ptr<IAudioStream> &stream, ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "createStream() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    std::cout << "createStream() succeeded." << std::endl;
    audioVoiceStream = std::dynamic_pointer_cast<IAudioVoiceStream>(stream);
}

//Create an Audio Stream (Voice Call Session)
StreamConfig config;
config.type = StreamType::VOICE_CALL;
config.slotId = DEFAULT_SLOT_ID;
config.sampleRate = 16000;
config.format = AudioFormat::PCM_16BIT_SIGNED;
config.channelTypeMask = ChannelType::LEFT;
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
status = audioManager->createStream(config, createStreamCallback);
```

5. Delete an Audio Stream (Voice Call Session), which was created earlier

```
//Callback which provides response to deleteStream
void deleteStreamCallback(ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "deleteStream() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    std::cout << "deleteStream() succeeded." << std::endl;
    audioVoiceStream.reset();
}
//Delete an Audio Stream (Voice Call Session), which was created earlier
status = audioManager->deleteStream(std::dynamic_pointer_cast<IAudioStream>(audioVoiceStream),
                                    deleteStreamCallback);
```

5.6.2 Audio playback session

Audio Manager API Sample Reference for audio playback session

This Section demonstrates how to use the Audio Manager API for audio playback session.

1. Get the AudioFactory instance

```
auto &audioFactory = audioFactory::getInstance();
```

2. Get the AudioManager object and check for audio subsystem Readiness

```
std::promise<ServiceStatus> prom{};
//  Get AudioManager instance.
audioManager = audioFactory.getAudioManager([&prom](ServiceStatus serviceStatus) {
    prom.set_value(serviceStatus);
});
if (!audioManager) {
    std::cout << "Failed to get AudioManager object" << std::endl;
    return;
}

//  Check if audio subsystem is ready
//  If audio subsystem is not ready, wait for it to be ready
ServiceStatus managerStatus = audioManager->getServiceStatus();
if (managerStatus != ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "\nAudio subsystem is not ready, Please wait ..." << std::endl;
    managerStatus = prom.get_future().get();
}

//  Check the service status again.
if (managerStatus == ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "Audio Subsytem is Ready << std::endl;
} else {
    std::cout << "ERROR - Unable to initialize audio subsystem" << std::endl;
    return;
}
```

3. Create an Audio Stream (Audio Playback Session)

```
//Callback which provides response to createStream, with pointer to base interface IAudioStream.
 void createStreamCallback(std::shared_ptr<IAudioStream> &stream, ErrorCode error)
 {
     if (error != ErrorCode::SUCCESS) {
         std::cout << "createStream() returned with error " << static_cast<int>(error)
             << std::endl;
         return;
     }
     std::cout << "createStream() succeeded." << std::endl;
     audioPlayStream = std::dynamic_pointer_cast<IAudioPlayStream>(stream);
 }
 //Create an Audio Stream (Audio Playback Session)
 StreamConfig config;
 config.type = StreamType::PLAY;
 config.sampleRate = 48000;
 config.format = AudioFormat::PCM_16BIT_SIGNED;
 config.channelTypeMask = ChannelType::LEFT;
 config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
 status = audioManager->createStream(config, createStreamCallback);
```

4. Allocate Stream buffers for Playback operation

```
// Get an audio buffer (can get more than one)
auto streamBuffer = audioPlayStream->getStreamBuffer();
if (streamBuffer != nullptr) {
    // Setting the size that is to be written to stream as the minimum size
    // required by stream. In any case if size returned is 0, using the Maximum
    // Buffer Size, any buffer size between min and max can be used
    size = streamBuffer->getMinSize();
    if (size == 0) {
        size =  streamBuffer->getMaxSize();
    }
    streamBuffer->setDataSize(size);
} else {
```

```
    std::cout << "Failed to get Stream Buffer " << std::endl;
}
```

5. Start write operation for playback to start

```
//Callback which provides response to write operation.
void writeCallback(std::shared_ptr<IStreamBuffer> buffer, uint32_t size, ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "write() returned with error " << static_cast<int>(error) << std::endl;
    } else {
        std::cout << "Successfully written " << size << " bytes" << std::endl;
    }
    buffer->reset();
    return;
}
//Write desired data into the buffer
//First write starts Playback Session.
memset(streamBuffer->getRawBuffer(),0x1,size);
auto status = audioPlayStream->write(streamBuffer, writeCallback);
if(status != telux::common::Status::SUCCESS) {
    std::cout << "write() failed with error" << static_cast<int>(status) << std::endl;
} else {
    std::cout << "Request to write to stream sent" << std::endl;
}
```

6. Delete an Audio Stream (Audio Playback Session), once reached end of operation

```
//Callback which provides response to deleteStream
void deleteStreamCallback(ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "deleteStream() returned with error " << static_cast<int>(error)
            << std::endl;
        return;
    }
    std::cout << "deleteStream() succeeded." << std::endl;
    audioPlayStream.reset();
}
//Delete an Audio Stream (Audio Playback Session), once reached end of operation.
Status  status = audioManager->deleteStream(
           std::dynamic_pointer_cast<IAudioStream>(audioPlayStream), deleteStreamCallback);
if (status != Status::SUCCESS) {
    std::cout << "deleteStream failed with error" << static_cast<int>(status) << std::endl;
}
```

5.6.3 Audio capture session

Audio Manager API Sample Reference for audio capture session

This Section demonstrates how to use the Audio Manager API for audio capture session.

1. Get the AudioFactory instance

```
auto &audioFactory = audioFactory::getInstance();
```

2. Get the AudioManager object and check for audio subsystem Readiness

```
std::promise<ServiceStatus> prom{};
//  Get AudioManager instance.
audioManager = audioFactory.getAudioManager([&prom](ServiceStatus serviceStatus) {
    prom.set_value(serviceStatus);
});
if (!audioManager) {
    std::cout << "Failed to get AudioManager object" << std::endl;
```

```
    return;
}

//  Check if audio subsystem is ready
//  If audio subsystem is not ready, wait for it to be ready
ServiceStatus managerStatus = audioManager->getServiceStatus();
if (managerStatus != ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "\nAudio subsystem is not ready, Please wait ..." << std::endl;
    managerStatus = prom.get_future().get();
}

//  Check the service status again.
if (managerStatus == ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "Audio Subsytem is Ready << std::endl;
} else {
    std::cout << "ERROR - Unable to initialize audio subsystem" << std::endl;
    return;
}
```

3. Create an Audio Stream (Audio Capture Session)

```
//Callback which provides response to createStream, with pointer to base interface IAudioStream.
void createStreamCallback(std::shared_ptr<IAudioStream> &stream, ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "createStream() returned with error " << static_cast<int>(error)
            << std::endl;
        return;
    }
    std::cout << "createStream() succeeded." << std::endl;
    audioCaptureStream = std::dynamic_pointer_cast<IAudioCaptureStream>(stream);
}

//Create an Audio Stream (Audio Capture Session)
StreamConfig config;
config.type = StreamType::CAPTURE;
config.sampleRate = 48000;
config.format = AudioFormat::PCM_16BIT_SIGNED;
config.channelTypeMask = ChannelType::LEFT;
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
status = audioManager->createStream(config, createStreamCallback);
```

4. Allocate Stream Buffers for Capture Operation

```
// Get an audio buffer (can get more than one)
auto streamBuffer = audioCaptureStream->getStreamBuffer();
if(streamBuffer != nullptr) {
    // Setting the bytesToRead (bytes to be read from stream) as minimum size
    // required by stream. In any case if size returned is 0, using the Maximum Buffer
    // Size, any buffer size between min and max can be used
    bytesToRead = streamBuffer->getMinSize();
    if(bytesToRead == 0) {
        bytesToRead = streamBuffer->getMaxSize();
    }
} else {
    std::cout << "Failed to get Stream Buffer " << std::endl;
    return EXIT_FAILURE;
}
```

5. Start read operation for the capture to Start

```
//Callback which provides response to read operation
void readCallback(std::shared_ptr<IStreamBuffer> buffer, ErrorCode error)
{
    uint32_t bytesWrittenToFile = 0;
    if (error != ErrorCode::SUCCESS) {
        std::cout << "read() returned with error " << static_cast<int>(error) << std::endl;
    } else {
```

```
        uint32_t size = buffer->getDataSize();
        std::cout << "Successfully read " << size << " bytes" << std::endl;
    }
    buffer->reset();
    return;
}
//Read from Capture
//First read starts Capture Session.
auto status = audioCaptureStream->read(streamBuffer, bytesToRead, readCallback);
if(status != telux::common::Status::SUCCESS) {
    std::cout << "read() failed with error" << static_cast<int>(status) << std::endl;
} else {
    std::cout << "Request to read stream sent" << std::endl;
}
```

6. Delete an Audio Stream (Audio Capture Session), once required bytes captured

```
//Callback which provides response to deleteStream
void deleteStreamCallback(ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "deleteStream() returned with error " << static_cast<int>(error)
                  << std::endl;
        return;
    }
    std::cout << "deleteStream() succeeded." << std::endl;
    audioCaptureStream.reset();
}
//Delete an Audio Stream (Audio Capture Session), once reached end of operation.
Status  status = audioManager->deleteStream(
           std::dynamic_pointer_cast<IAudioStream>(audioCaptureStream), deleteStreamCallback);
if (status != Status::SUCCESS) {
    std::cout << "deleteStream failed with error" << static_cast<int>(status) << std::endl;
}
```

5.6.4 Audio loopback session

Audio Manager API Sample Reference for audio loopback session

Please follow the below steps to start/stop loopback on a loopback session.

1. Get the AudioFactory instance

```
auto &audioFactory = audioFactory::getInstance();
```

2. Get the AudioManager object and check for audio subsystem Readiness

```
std::promise<ServiceStatus> prom{};
//  Get AudioManager instance.
audioManager = audioFactory.getAudioManager([&prom](ServiceStatus serviceStatus) {
    prom.set_value(serviceStatus);
});
if (!audioManager) {
    std::cout << "Failed to get AudioManager object" << std::endl;
    return;
}

//  Check if audio subsystem is ready
//  If audio subsystem is not ready, wait for it to be ready
ServiceStatus managerStatus = audioManager->getServiceStatus();
if (managerStatus != ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "\nAudio subsystem is not ready, Please wait ..." << std::endl;
    managerStatus = prom.get_future().get();
}
```

```
//  Check the service status again.
if (managerStatus == ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "Audio Subsytem is Ready << std::endl;
} else {
    std::cout << "ERROR - Unable to initialize audio subsystem" << std::endl;
    return;
}
```

3. Create an audio Stream (to be associated with loopback)

```
// Implement a response function to get the request status
void createStreamCallback(std::shared_ptr<IAudioStream> &stream, ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "createStream() failed with error" << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "createStream() succeeded." << std::endl;
    audioLoopbackStream = std::dynamic_pointer_cast<IAudioLoopbackStream>(stream);
}
// Create a loopback stream with required configuration
config.type = telux::audio::StreamType::LOOPBACK;
config.sampleRate = 48000;
config.format = AudioFormat::PCM_16BIT_SIGNED;
config.channelTypeMask = ChannelType::LEFT;
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_MIC);
status = audioManager->createStream(config, createStreamCallback);
```

4. Start loopback between the specified source and sink devices

```
// Implement a response function to get the request status
void startLoopbackCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "startLoopback() failed with error" << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "startLoopback() succeeded." << std::endl;
}
// start loopback
status = audioLoopbackStream->startLoopback(startLoopbackCallback);
```

5. Stop loopback between the specified source and sink devices

```
// Implement a response function to get the request status
void stopLoopbackCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "stopLoopback() failed with error " << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "stopLoopback() succeeded." << std::endl;
}
// Stop the loopback, which was started earlier
status = audioLoopbackStream->stopLoopback(stopLoopbackCallback);
```

6. Delete the audio stream associated with the Loopback session

```
// Implement a response function to get the request status
void deleteStreamCallback(ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "deleteStream() failed with error" << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "deleteStream() succeeded." << std::endl;
    audioLoopbackStream.reset();
```

```
}
// Delete the Audio Stream
status = audioManager->deleteStream(std::dynamic_pointer_cast<IAudioStream>(audioLoopbackStream),
                                    deleteStreamCallback);
```

5.6.5 Audio tone generation

Audio Manager API Sample Reference for audio tone generation

Please follow the below steps to play a tone in an active tone generator stream.

1. Get the AudioFactory instance

```
auto &audioFactory = audioFactory::getInstance();
```

2. Get the AudioManager object and check for audio subsystem Readiness

```
std::promise<ServiceStatus> prom{};
//  Get AudioManager instance.
audioManager = audioFactory.getAudioManager([&prom](ServiceStatus serviceStatus) {
    prom.set_value(serviceStatus);
});
if (!audioManager) {
    std::cout << "Failed to get AudioManager object" << std::endl;
    return;
}

//  Check if audio subsystem is ready
//  If audio subsystem is not ready, wait for it to be ready
ServiceStatus managerStatus = audioManager->getServiceStatus();
if (managerStatus != ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "\nAudio subsystem is not ready, Please wait ..." << std::endl;
    managerStatus = prom.get_future().get();
}

//  Check the service status again.
if (managerStatus == ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "Audio Subsytem is Ready << std::endl;
} else {
    std::cout << "ERROR - Unable to initialize audio subsystem" << std::endl;
    return;
}
```

3. Create an audio Stream (to be associated with tone generator)

```
// Implement a response function to get the request status
void createStreamCallback(std::shared_ptr<IAudioStream> &stream, ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "createStream() failed with error" << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "createStream() succeeded." << std::endl;
    audioToneGeneratorStream = std::dynamic_pointer_cast<IAudioToneGeneratorStream>(stream);
}
// Create a tone generator stream with required configuration
config.type = telux::audio::StreamType::TONE_GENERATOR;
config.sampleRate = 48000;
config.format = AudioFormat::PCM_16BIT_SIGNED;
config.channelTypeMask = ChannelType::LEFT;
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
status = audioManager->createStream(config, createStreamCallback);
```

4. Play tone on a sink device

```
// Implement a response function to get the request status
void playToneCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "playTone() failed with error" << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "playTone() succeeded." << std::endl;
}
// Play the tone with required configuration
status = audioToneGeneratorStream->playTone(freq, duration, gain, playToneCallback);
```

5. Optionally, you can stop the tone being played before the specified duration elapses

```
// Implement a response function to get the request status
void stopToneCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "stopTone() failed with error " << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "stopTone() succeeded." << std::endl;
}
// Stop the tone play, which was started earlier
status = audioToneGeneratorStream->stopTone(stopToneCallback);
```

6. Delete the audio stream associated with the Tone Generator session

```
// Implement a response function to get the request status
void deleteStreamCallback(ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "deleteStream() failed with error" << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "deleteStream() succeeded." << std::endl;
    audioToneGeneratorStream.reset();
}
//Delete the Audio Stream
status = audioManager->deleteStream(
    std::dynamic_pointer_cast<IAudioStream>(audioToneGeneratorStream), deleteStreamCallback);
```

5.6.6 Audio voice session start and stop

Audio Manager API Sample Reference for voice session start and stop

This Section demonstrates how to use the Audio Manager API for voice session start and stop.

1. Get the AudioFactory instance

```
auto &audioFactory = audioFactory::getInstance();
```

2. Get the AudioManager object and check for audio subsystem Readiness

```
std::promise<ServiceStatus> prom{};
//  Get AudioManager instance.
audioManager = audioFactory.getAudioManager([&prom](ServiceStatus serviceStatus) {
    prom.set_value(serviceStatus);
});
if (!audioManager) {
    std::cout << "Failed to get AudioManager object" << std::endl;
```

```
    return;
}

//  Check if audio subsystem is ready
//  If audio subsystem is not ready, wait for it to be ready
ServiceStatus managerStatus = audioManager->getServiceStatus();
if (managerStatus != ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "\nAudio subsystem is not ready, Please wait ..." << std::endl;
    managerStatus = prom.get_future().get();
}

//  Check the service status again.
if (managerStatus == ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "Audio Subsytem is Ready << std::endl;
} else {
    std::cout << "ERROR - Unable to initialize audio subsystem" << std::endl;
    return;
}
```

3. Create an Audio Stream (Voice Call Session)

```
//Callback which provides response to createStream, with pointer to base interface IAudioStream.
void createStreamCallback(std::shared_ptr<IAudioStream> &stream, ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "createStream() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    std::cout << "createStream() succeeded." << std::endl;
    audioVoiceStream = std::dynamic_pointer_cast<IAudioVoiceStream>(stream);
}

//Create an Audio Stream (Voice Call Session)
StreamConfig config;
config.type = StreamType::VOICE_CALL;
config.slotId = DEFAULT_SLOT_ID;
config.sampleRate = 16000;
config.format = AudioFormat::PCM_16BIT_SIGNED;
config.channelTypeMask = ChannelType::LEFT;
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
status = audioManager->createStream(config, createStreamCallback);
```

4. Start Created Audio Stream (Voice Call Session)

```
//Callback which provides response to startAudio.
void startAudioCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "startAudio() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    std::cout << "startAudio() succeeded." << std::endl;
}

//Start an Audio Stream (Voice Call Session)
status = audioVoiceStream->startAudio(startAudioCallback);
```

5. Stop Created Audio Stream (Voice Call Session)

```
//Callback which provides response to stopAudio.
void stopAudioCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "stopAudio() returned with error " << static_cast<unsigned int>(error)
```

```
            << std::endl;
        return;
    }

    std::cout << "stopAudio() succeeded." << std::endl;
}

//Stop an Audio Stream (Voice Call Session), which was started earlier
status = audioVoiceStream->stopAudio(stopAudioCallback);
```

6. Delete an Audio Stream (Voice Call Session), which was created earlier

```
//Callback which provides response to deleteStream
void deleteStreamCallback(ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "deleteStream() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    std::cout << "deleteStream() succeeded." << std::endl;
    audioVoiceStream.reset();
}
//Delete an Audio Stream (Voice Call Session), which was created earlier
status = audioManager->deleteStream(std::dynamic_pointer_cast<IAudioStream>(audioVoiceStream),
                                    deleteStreamCallback);
```

5.6.7 Audio voice session volume/mute control

Audio Manager API Sample Reference for voice session volume/mute control

This Section demonstrates how to use the Audio Manager API for voice session volume/mute control.

1. Get the AudioFactory instance

```
auto &audioFactory = audioFactory::getInstance();
```

2. Get the AudioManager object and check for audio subsystem Readiness

```
std::promise<ServiceStatus> prom{};
//  Get AudioManager instance.
audioManager = audioFactory.getAudioManager([&prom](ServiceStatus serviceStatus) {
    prom.set_value(serviceStatus);
});
if (!audioManager) {
    std::cout << "Failed to get AudioManager object" << std::endl;
    return;
}

//  Check if audio subsystem is ready
//  If audio subsystem is not ready, wait for it to be ready
ServiceStatus managerStatus = audioManager->getServiceStatus();
if (managerStatus != ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "\nAudio subsystem is not ready, Please wait ..." << std::endl;
    managerStatus = prom.get_future().get();
}

//  Check the service status again.
if (managerStatus == ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "Audio Subsytem is Ready << std::endl;
} else {
    std::cout << "ERROR - Unable to initialize audio subsystem" << std::endl;
    return;
```

```
}
```

3. Create an Audio Stream (Voice Call Session)

```
//Callback which provides response to createStream, with pointer to base interface IAudioStream.
void createStreamCallback(std::shared_ptr<IAudioStream> &stream, ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "createStream() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    std::cout << "createStream() succeeded." << std::endl;
    audioVoiceStream = std::dynamic_pointer_cast<IAudioVoiceStream>(stream);
}

//Create an Audio Stream (Voice Call Session)
StreamConfig config;
config.type = StreamType::VOICE_CALL;
config.slotId = DEFAULT_SLOT_ID;
config.sampleRate = 16000;
config.format = AudioFormat::PCM_16BIT_SIGNED;
config.channelTypeMask = ChannelType::LEFT;
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
status = audioManager->createStream(config, createStreamCallback);
```

4. Start Created Audio Stream (Voice Call Session)

```
//Callback which provides response to startAudio.
void startAudioCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "startAudio() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    std::cout << "startAudio() succeeded." << std::endl;
}

//Start an Audio Stream (Voice Call Session)
status = audioVoiceStream->startAudio(startAudioCallback);
```

5. Set volume on Started Audio Stream (Voice Call Session) for specified direction

```
//Callback which provides response to setVolume.
void setStreamVolumeCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "setVolume() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    std::cout << "setVolume() succeeded." << std::endl;
}

//Set volume on an Audio Stream (Voice Call Session) for RX direction
StreamVolume streamVol;
ChannelVolume channelVol;
streamVol.dir = StreamDirection::RX;
channelVol.channelType = ChannelType::LEFT;
channelVol.vol = 0.5;
streamVol.volume.emplace_back(channelVol);
status = audioVoiceStream->setVolume(streamVol, setStreamVolumeCallback);
```

6. Get volume on Started Audio Stream (Voice Call Session)

```
//Callback which provides response to getVolume.
void getStreamVolumeCallback(StreamVolume volume, ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "getVolume() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    std::cout << "Volume direction: " << static_cast<uint32_t>(volume.dir) << std::endl;

    int i = 0;
    for (auto channel_volume : volume.volume) {
        std::cout << "ChannelVolume [" << i << "] channel type: "
            << static_cast<uint32_t>(channel_volume.channelType) << ", " << "volume: "
            << channel_volume.vol << std::endl;
    }
}

//Get volume on an Audio Stream (Voice Call Session) for RX direction
status = audioVoiceStream->getVolume(StreamDirection::RX, getStreamVolumeCallback);
```

7. Set Mute on Started Audio Stream (Voice Call Session) for specified direction

```
//Callback which provides response to setMute.
void setStreamMuteCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "setMute() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    std::cout << "setMute() succeeded." << std::endl;
}

//Set Mute on an Audio Stream (Voice Call Session) for TX direction
StreamMute mute;
mute.dir = StreamDirection::TX;
mute.enable = true; //true: enable, false: disable
status = audioVoiceStream->setMute(mute, setStreamMuteCallback);
```

8 Get Mute on Started Audio Stream (Voice Call Session) for specified direction

```
//Callback which provides response to getMute.
void getStreamMuteCallback(StreamMute mute, ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "getMute() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    std::cout << "Mute enable: " << mute.enable << ", direction: "
        << static_cast<uint32_t>(mute.dir) << std::endl;

}

//Get Mute on an Audio Stream (Voice Call Session) for TX direction
status = audioVoiceStream->getMute(StreamDirection::TX, getStreamMuteCallback);
```

9. Stop Created Audio Stream (Voice Call Session)

```
//Callback which provides response to stopAudio.
void stopAudioCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "stopAudio() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    std::cout << "stopAudio() succeeded." << std::endl;
}

//Stop an Audio Stream (Voice Call Session), which was started earlier
status = audioVoiceStream->stopAudio(stopAudioCallback);
```

10. Delete an Audio Stream (Voice Call Session), which was created earlier

```
//Callback which provides response to deleteStream
void deleteStreamCallback(ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "deleteStream() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    std::cout << "deleteStream() succeeded." << std::endl;
    audioVoiceStream.reset();
}
//Delete an Audio Stream (Voice Call Session), which was created earlier
status = audioManager->deleteStream(std::dynamic_pointer_cast<IAudioStream>(audioVoiceStream),
                                    deleteStreamCallback);
```

5.6.8 Audio voice session device switch

Audio Manager API Sample Reference for voice session device switch

This Section demonstrates how to use the Audio Manager API for voice session device switch.

1. Get the AudioFactory instance

```
auto &audioFactory = audioFactory::getInstance();
```

2. Get the AudioManager object and check for audio subsystem Readiness

```
std::promise<ServiceStatus> prom{};
//  Get AudioManager instance.
audioManager = audioFactory.getAudioManager([&prom](ServiceStatus serviceStatus) {
    prom.set_value(serviceStatus);
});
if (!audioManager) {
    std::cout << "Failed to get AudioManager object" << std::endl;
    return;
}

//  Check if audio subsystem is ready
//  If audio subsystem is not ready, wait for it to be ready
ServiceStatus managerStatus = audioManager->getServiceStatus();
if (managerStatus != ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "\nAudio subsystem is not ready, Please wait ..." << std::endl;
    managerStatus = prom.get_future().get();
}
```

```
//  Check the service status again.
if (managerStatus == ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "Audio Subsytem is Ready << std::endl;
} else {
    std::cout << "ERROR - Unable to initialize audio subsystem" << std::endl;
    return;
}
```

3. Create an Audio Stream (Voice Call Session)

```
//Callback which provides response to createStream, with pointer to base interface IAudioStream.
void createStreamCallback(std::shared_ptr<IAudioStream> &stream, ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "createStream() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    std::cout << "createStream() succeeded." << std::endl;
    audioVoiceStream = std::dynamic_pointer_cast<IAudioVoiceStream>(stream);
}

//Create an Audio Stream (Voice Call Session)
StreamConfig config;
config.type = StreamType::VOICE_CALL;
config.slotId = DEFAULT_SLOT_ID;
config.sampleRate = 16000;
config.format = AudioFormat::PCM_16BIT_SIGNED;
config.channelTypeMask = ChannelType::LEFT;
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
status = audioManager->createStream(config, createStreamCallback);
```

4. Start Created Audio Stream (Voice Call Session)

```
//Callback which provides response to startAudio.
void startAudioCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "startAudio() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    std::cout << "startAudio() succeeded." << std::endl;
}

//Start an Audio Stream (Voice Call Session)
status = audioVoiceStream->startAudio(startAudioCallback);
```

5. Device switch on Started Audio Stream (Voice Call Session)

```
//Callback which provides response to setDevice.
void setStreamDeviceCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "setDevice() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    std::cout << "setDevice() succeeded." << std::endl;
}

//Set New Device for an Audio Stream (Voice Call Session)
std::vector<DeviceType> devices;
devices.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER); //Set new device type
```

```
status = audioVoiceStream->setDevice(devices, setStreamDeviceCallback);
```

6. Query Device details on Started Audio Stream (Voice Call Session)

```
//Callback which provides response to getDevice.
void getStreamDeviceCallback(std::vector<DeviceType> devices, ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "getDevice() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    int i = 0;
    for (auto device_type : devices) {
        std::cout << "Device [" << i << "] type: " << static_cast<uint32_t>(device_type)
            << std::endl;
        i++;
    }
}

//get Device details of an Audio Stream (Voice Call Session)
status = audioVoiceStream->getDevice(getStreamDeviceCallback);
```

7. Stop Created Audio Stream (Voice Call Session)

```
//Callback which provides response to stopAudio.
void stopAudioCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "stopAudio() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    std::cout << "stopAudio() succeeded." << std::endl;
}

//Stop an Audio Stream (Voice Call Session), which was started earlier
status = audioVoiceStream->stopAudio(stopAudioCallback);
```

8. Delete an Audio Stream (Voice Call Session), which was created earlier

```
//Callback which provides response to deleteStream
void deleteStreamCallback(ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "deleteStream() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

    std::cout << "deleteStream() succeeded." << std::endl;
    audioVoiceStream.reset();
}
//Delete an Audio Stream (Voice Call Session), which was created earlier
status = audioManager->deleteStream(std::dynamic_pointer_cast<IAudioStream>(audioVoiceStream),
                                    deleteStreamCallback);
```

5.6.9 Using Audio Manager APIs to play DTMF tone in a voice call.

Using Audio Manager APIs to play DTMF tone in a voice call

Please follow the below steps to play a DTMF tone in an active voice call. Note that only Rx direction is supported now.

1. Get the AudioFactory instance

```
auto &audioFactory = audioFactory::getInstance();
```

2. Get the AudioManager object and check for audio subsystem Readiness

```
std::promise<ServiceStatus> prom{};
//  Get AudioManager instance.
audioManager = audioFactory.getAudioManager([&prom](ServiceStatus serviceStatus) {
    prom.set_value(serviceStatus);
});
if (!audioManager) {
    std::cout << "Failed to get AudioManager object" << std::endl;
    return;
}

//  Check if audio subsystem is ready
//  If audio subsystem is not ready, wait for it to be ready
ServiceStatus managerStatus = audioManager->getServiceStatus();
if (managerStatus != ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "\nAudio subsystem is not ready, Please wait ..." << std::endl;
    managerStatus = prom.get_future().get();
}

//  Check the service status again.
if (managerStatus == ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "Audio Subsytem is Ready << std::endl;
} else {
    std::cout << "ERROR - Unable to initialize audio subsystem" << std::endl;
    return;
}
```

3. Create an audio Stream (to be associated with Voice call session)

```
// Implement a response function to get the request status
void createStreamCallback(std::shared_ptr<IAudioStream> &stream, ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "createStream() failed with error" << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "createStream() succeeded." << std::endl;
    audioVoiceStream = std::dynamic_pointer_cast<IAudioVoiceStream>(stream);
}
// Create a voice stream with required configuration
status = audioManager->createStream(config, createStreamCallback);
```

4. Start the Voice call session

```
// Implement a response function to get the request status
void startAudioCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "startAudio() failed with error " << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "startAudio() succeeded." << std::endl;
}
// Start the Voice call session
status = audioVoiceStream->startAudio(startAudioCallback);
```

5. Play a DTMF tone

```
// Implement a response function to get the request status
void playDtmfCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "playDtmfTone() failed with error" << static_cast<int>(error) << std::endl;
```

```
        return;
    }
    std::cout << "playDtmfTone() succeeded." << std::endl;
}
// Play the DTMF tone with required configuration
status = audioVoiceStream->playDtmfTone(dtmfTone, duration, gain, playDtmfCallback);
```

6. Stop the Voice call session

```
// Implement a response function to get the request status
void stopAudioCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "stopAudio() failed with error " << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "stopAudio() succeeded." << std::endl;
}
// Stop the Voice call session, which was started earlier
status = audioVoiceStream->stopAudio(stopAudioCallback);
```

7. Delete the audio stream associated with the Voice call session

```
// Implement a response function to get the request status
void deleteStreamCallback(ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "deleteStream() failed with error" << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "deleteStream() succeeded." << std::endl;
    audioVoiceStream->reset();
}
//Delete the Audio Stream
status = audioManager->deleteStream(std::dynamic_pointer_cast<IAudioStream>(audioVoiceStream),
                                    deleteStreamCallback);
```

5.6.10 Using Audio Manager APIs to detect DTMF tones in a voice call.

Using Audio Manager APIs to detect DTMF tones in a voice call

Please follow the below steps to detect DTMF tones in an active voice call. Note that only Rx direction is supported now.

1. Get the AudioFactory instance

```
auto &audioFactory = audioFactory::getInstance();
```

2. Get the AudioManager object and check for audio subsystem Readiness

```
std::promise<ServiceStatus> prom{};
//  Get AudioManager instance.
audioManager = audioFactory.getAudioManager([&prom](ServiceStatus serviceStatus) {
    prom.set_value(serviceStatus);
});
if (!audioManager) {
    std::cout << "Failed to get AudioManager object" << std::endl;
    return;
}

//  Check if audio subsystem is ready
//  If audio subsystem is not ready, wait for it to be ready
ServiceStatus managerStatus = audioManager->getServiceStatus();
if (managerStatus != ServiceStatus::SERVICE_AVAILABLE) {
```

```
    std::cout << "\nAudio subsystem is not ready, Please wait ..." << std::endl;
    managerStatus = prom.get_future().get();
}

//  Check the service status again.
if (managerStatus == ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "Audio Subsytem is Ready << std::endl;
} else {
    std::cout << "ERROR - Unable to initialize audio subsystem" << std::endl;
    return;
}
```

3. Exit the application, if SDK is unable to initialize Audio subsystem

```
if(isReady) {
    std::cout << " *** Audio subsystem is Ready *** " << std::endl;
} else {
    std::cout << " *** ERROR - Unable to initialize Audio subsystem " << std::endl;
    return 1;
}
```

4. Create an audio Stream (to be associated with Voice call session)

```
// Implement a response function to get the request status
void createStreamCallback(std::shared_ptr<IAudioStream> &stream, ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "createStream() failed with error" << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "createStream() succeeded." << std::endl;
    audioVoiceStream = std::dynamic_pointer_cast<IAudioVoiceStream>(stream);
}
// Create a voice stream with required configuration
status = audioManager->createStream(config, createStreamCallback);
```

5. Start the Voice call session

```
// Implement a response function to get the request status
void startAudioCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "startAudio() failed with error " << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "startAudio() succeeded." << std::endl;
}
// Start the Voice call session
status = audioVoiceStream->startAudio(startAudioCallback);
```

6. Register a listener to get notifications on DTMF tone detection

```
// Implement a response function to get the request status
void registerListenerCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "registerListener() failed with error " << static_cast<int>(error)
                          << std::endl;
        return;
    }
    std::cout << "registerListener() succeeded." << std::endl;
}
// Instantiate MyVoiceListener
auto myDtmfToneListener = std::make_shared<MyVoiceListener>();
// Register the listener
status = audioVoiceStream->registerListener(myDtmfToneListener, registerListenerCallback);
```

7. De-register the listener to stop getting the notifications

```
status = audioVoiceStream->deRegisterListener(myDtmfToneListener);
```

8. Stop the Voice call session

```
// Implement a response function to get the request status
void stopAudioCallback(ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "stopAudio() failed with error " << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "stopAudio() succeeded." << std::endl;
}
// Stop the Voice call session, which was started earlier
status = audioVoiceStream->stopAudio(stopAudioCallback);
```

9. Delete the audio stream associated with the Voice call session

```
// Implement a response function to get the request status
void deleteStreamCallback(ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "deleteStream() failed with error" << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "deleteStream() succeeded." << std::endl;
    audioVoiceStream->reset();
}
//Delete the Audio Stream
status = audioManager->deleteStream(std::dynamic_pointer_cast<IAudioStream>(audioVoiceStream),
                                    deleteStreamCallback);
```

5.6.11 Audio transcoding operation

Audio Manager APIs Sample Reference for audio transcoding operation

This Section demonstrates how to use the Audio Manager and Audio Transcoder APIs for transcoding operation.

1. Get the AudioFactory instance

```
auto &audioFactory = audioFactory::getInstance();
```

2. Get the AudioManager object and check for audio subsystem Readiness

```
std::promise<ServiceStatus> prom{};
//  Get AudioManager instance.
audioManager = audioFactory.getAudioManager([&prom](ServiceStatus serviceStatus) {
    prom.set_value(serviceStatus);
});
if (!audioManager) {
    std::cout << "Failed to get AudioManager object" << std::endl;
    return;
}

//  Check if audio subsystem is ready
//  If audio subsystem is not ready, wait for it to be ready
ServiceStatus managerStatus = audioManager->getServiceStatus();
if (managerStatus != ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "\nAudio subsystem is not ready, Please wait ..." << std::endl;
```

```
    managerStatus = prom.get_future().get();
}

//  Check the service status again.
if (managerStatus == ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "Audio Subsytem is Ready << std::endl;
} else {
    std::cout << "ERROR - Unable to initialize audio subsystem" << std::endl;
    return;
}
```

3. Create an Audio Transcoder

```
FormatInfo inputConfig_;
FormatInfo outputConfig_;
// input and output parameters are configured for AMRWB_PLUS to PCM_16BIT_SIGNED transcoding
// operation as an example.
AmrwbpParams inputParams{};
inputConfig_.sampleRate = SAMPLE_RATE;
inputConfig_.mask = CHANNEL_MASK;
inputConfig_.format = AudioFormat::AMRWB_PLUS;
inputParams.bitWidth = 16;
inputParams.frameFormat = AmrwbpFrameFormat::FILE_STORAGE_FORMAT;
inputConfig_.params = &inputParams;

inputConfig_.sampleRate = SAMPLE_RATE;
outputConfig_.mask = CHANNEL_MASK;
outputConfig_.format = AudioFormat::PCM_16BIT_SIGNED;
outputConfig_.params = nullptr;

audioManager->createTranscoder(inputConfig_, outputConfig_,
[&p,this](std::shared_ptr<telux::audio::ITranscoder> &transcoder,
    telux::common::ErrorCode error) {
    if (error == telux::common::ErrorCode::SUCCESS) {
        transcoder_ = transcoder;
        registerListener();
        p.set_value(true);
    } else {
        p.set_value(false);
        std::cout << "failed to create transcoder" <<std::endl;
    }
});
if (p.get_future().get()) {
    std::cout<< "Transcoder Created" << std::endl;
}
```

4.1 Allocate Audio buffers for write operation

```
// Get an audio buffer for write operation (can get more than one)
auto audioBuffer = transcoder_->getWriteBuffer();
if (audioBuffer != nullptr) {
    // Setting the size of buffer that need to be supplied for write operation as the minimum
    // size required by transcoder. In any case if size returned is 0, using the Maximum
    // Buffer Size, any buffer size between min and max can be used
    size = audioBuffer->getMinSize();
    if (size == 0) {
        size =  audioBuffer->getMaxSize();
    }
    audioBuffer->setDataSize(size);
} else {
    std::cout << "Failed to get Audio Buffer for write operation " << std::endl;
}
```

4.2 Allocate Audio buffers for read operation

```
// Get an audio buffer for read operation (can get more than one)
auto audioBuffer = transcoder_->getReadBuffer();
if (audioBuffer != nullptr) {
    // Setting the size of buffer that need to be supplied for read operation as the minimum
    // size required by transcoder. In any case if size returned is 0, using the Maximum
    // Buffer Size, any buffer size between min and max can be used
    size = audioBuffer->getMinSize();
    if (size == 0) {
        size =  audioBuffer->getMaxSize();
    }
    audioBuffer->setDataSize(size);
} else {
    std::cout << "Failed to get Audio Buffer for read operation " << std::endl;
}
```

5. Start write operation in one thread for transcoding

```
// Callback which provides response to write operation.
void writeCallback(std::shared_ptr<IAudioBuffer> buffer,
    uint32_t bytes, ErrorCode error) {
    std::cout << "Bytes Written : " << bytes << std::endl;
    if (error != ErrorCode::SUCCESS || buffer->getDataSize() != bytes) {
        // Application needs to resend the Bitstream buffer from leftover position if bytes
        // consumed are not equal to requested number of bytes to be written.
        pipeLineEmpty_ = false;
    }
    buffer->reset();
    writeBuffers_.push(buffer);
    cv_.notify_all();
    return;
}
// Indiction Received only when callback returns with error that bytes written are not equal to
// bytes requested to write. It notifies that pipeline is ready to accept new buffer to write.

void onReadyForWrite() {
    pipeLineEmpty_ = true;
}

// Write request for transcoding
auto writeCb = std::bind(&TranscoderApp::writeCallback, this, std::placeholders::_1,
                std::placeholders::_2, std::placeholders::_3);
telux::common::Status status = telux::common::Status::FAILED;
// EOF_REACHED denotes flag which indicated EOF is reached
// EOF_NOT_REACHED denotes flag which indicated EOF is not reached
if (EOF_REACHED) {
    status = transcoder_->write(audioBuffer, EOF_REACHED,  writeCb);
} else {
    status = transcoder_->write(audioBuffer, EOF_NOT_REACHED,  writeCb);
}
if (status != telux::common::Status::SUCCESS) {
    std::cout << "write() failed with error" << static_cast<unsigned int>(status) << std::endl;
} else {
    std::cout << "Request to transcode buffers sent " << std::endl;
}
```

6. Start read operation in another thread for transcoding

```
// Callback which provides response to read operation.
void readCallback(std::shared_ptr<telux::audio::IAudioBuffer> buffer,
    uint32_t isLastBuffer, telux::common::ErrorCode error) {
    if (isLastBuffer) {
        // Stop reading from now onwards as this is the last transcoded buffer
    }
    if (error != telux::common::ErrorCode::SUCCESS) {
        std::cout << "read() returned with error " << static_cast<unsigned int>(error)
                << std::endl;
    } else {
```

```
        // readed buffer can be stored on a file if required.
    }
    buffer->reset();
    readBuffers_.push(buffer);
    cv_.notify_all();
    return;
}

// Read request for transcoding
auto readCb =  std::bind(&TranscoderApp::readCallback, this,
        std::placeholders::_1, std::placeholders::_2, std::placeholders::_3);
telux::common::Status status = transcoder_->read(audioBuffer, bytesToRead, readCb);
if (status != telux::common::Status::SUCCESS) {
    std::cout << "read() failed with error" << static_cast<unsigned int>(status) << std::endl;
}
```

7. Tear down the audio transcoder instance

```
// Tear down is supposed to be called after the last buffer is received for write operation
// It is supposed to be called after every transcoding operation as transcoder instance can not
// be used for  multiple transcoding operations.

std::promise<bool> p;
auto status = transcoder_->tearDown([&p](telux::common::ErrorCode error) {
    if (error == telux::common::ErrorCode::SUCCESS) {
        p.set_value(true);
    } else {
        p.set_value(false);
        std::cout << "Failed to tear down" << std::endl;
    }
});
if (status == telux::common::Status::SUCCESS) {
    std::cout << "Request to Teardown transcoder sent" << std::endl;
} else {
    std::cout << "Request to Teardown transcoder failed" << std::endl;
}
if (p.get_future().get()) {
    transcoder_ = nullptr;
    std::cout << "Tear Down successful !!" << std::endl;
}
```

5.6.12 Compressed audio format playback

Audio Manager APIs Sample Reference for compressed audio format playback

This Section demonstrates how to use the Audio Manager API for compressed audio format playback.

1. Get the AudioFactory instance

```
auto &audioFactory = audioFactory::getInstance();
```

2. Get the AudioManager object and check for audio subsystem Readiness

```
std::promise<ServiceStatus> prom{};
//  Get AudioManager instance.
audioManager = audioFactory.getAudioManager([&prom](ServiceStatus serviceStatus) {
    prom.set_value(serviceStatus);
});
if (!audioManager) {
    std::cout << "Failed to get AudioManager object" << std::endl;
    return;
}
```

```
//  Check if audio subsystem is ready
//  If audio subsystem is not ready, wait for it to be ready
ServiceStatus managerStatus = audioManager->getServiceStatus();
if (managerStatus != ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "\nAudio subsystem is not ready, Please wait ..." << std::endl;
    managerStatus = prom.get_future().get();
}

//  Check the service status again.
if (managerStatus == ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "Audio Subsytem is Ready << std::endl;
} else {
    std::cout << "ERROR - Unable to initialize audio subsystem" << std::endl;
    return;
}
```

3. Create an Audio Stream (Audio Playback Session)

```
// Callback which provides response to createStream with pointer to base interface IAudioStream.
void createStreamCallback(std::shared_ptr<IAudioStream> &stream, ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "createStream() returned with error " << static_cast<int>(error)
            << std::endl;
        return;
    }
    std::cout << "createStream() succeeded." << std::endl;
    audioPlayStream = std::dynamic_pointer_cast<IAudioPlayStream>(stream);
}
// Create an Audio Stream (Audio Playback Session)
StreamConfig config;
config.type = StreamType::PLAY;
config.sampleRate = 48000;
config.format = AudioFormat::AMRWB_PLUS;
config.channelTypeMask = ChannelType::LEFT;
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
AmrwbpParams params;
params.bitWidth = 16;
params.frameFormat = AmrwbpFrameFormat::FILE_STORAGE_FORMAT;
config.formatParams = &params;
status = audioManager->createStream(config, createStreamCallback);
```

4. Allocate Stream buffers for Playback operation

```
// Get an audio buffer (can get more than one)
auto streamBuffer = audioPlayStream->getStreamBuffer();
if (streamBuffer != nullptr) {
    // Setting the size that is to be written to stream as the minimum size
    // required by stream. In any case if size returned is 0, using the Maximum
    // Buffer Size, any buffer size between min and max can be used
    size = streamBuffer->getMinSize();
    if (size == 0) {
        size =  streamBuffer->getMaxSize();
    }
    streamBuffer->setDataSize(size);
} else {
    std::cout << "Failed to get Stream Buffer " << std::endl;
}
```

5. Start write operation for playback to start

```
// Callback which provides response to write operation.
void writeCallback(std::shared_ptr<IStreamBuffer> buffer, uint32_t bytes, ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "write() returned with error " << static_cast<int>(error) << std::endl;
        // Application needs to resend the Bitstream buffer from leftover position if bytes
        // consumed are not equal to requested number of bytes to be written.
```

```
        pipeLineEmpty_ = false;
    }
    buffer->reset();
    return;
}

// Indiction Received only when callback returns with error that bytes written are not equal to
// bytes requested to write. It notifies that pipeline is ready to accept new buffer to write.
void onReadyForWrite() {
    pipeLineEmpty_ = true;
}

// Write desired data into the buffer, the bytes sent as 0x1 for example purpose only.
// First write starts Playback Session.
memset(streamBuffer->getRawBuffer(),0x1,size);
auto status = audioPlayStream->write(streamBuffer, writeCallback);
if (status != telux::common::Status::SUCCESS) {
    std::cout << "write() failed with error" << static_cast<int>(status) << std::endl;
} else {
    std::cout << "Request to write to stream sent" << std::endl;
}
```

6.1 Stop playback operation(STOP_AFTER_PLAY : Stops after playing pending buffers in pipeline)

```
std::promise<bool> p;
auto status = audioPlayStream_->stopAudio(StopType::STOP_AFTER_PLAY, [&p](ErrorCode error) {
    if (error == ErrorCode::SUCCESS) {
        p.set_value(true);
    } else {
        p.set_value(false);
        std::cout << "Failed to stop after playing buffers" << std::endl;
    }
});
if (status == Status::SUCCESS) {
    std::cout << "Request to stop playback after pending buffers Sent" << std::endl;
} else {
    std::cout << "Request to stop playback after pending buffers failed" << std::endl;
}
if (p.get_future().get()) {
    std::cout << "Pending buffers played successful !!" << std::endl;
}
```

6.2 Stop playback operation(FORCE_STOP : Stops immediately, all buffers in pipeline are flushed)

```
std::promise<bool> p;
    auto status = audioPlayStream_->stopAudio(
        StopType::FORCE_STOP, [&p](telux::common::ErrorCode error) {
        if (error == telux::common::ErrorCode::SUCCESS) {
            p.set_value(true);
        } else {
            p.set_value(false);
            std::cout << "Failed to force stop" << std::endl;
        }
        });
    if(status == telux::common::Status::SUCCESS){
        std::cout << "Request to force stop Sent" << std::endl;
    } else {
        std::cout << "Request to force stop failed" << std::endl;
    }
    if (p.get_future().get()) {
            std::cout << "Force Stop successful !!" << std::endl;
    }
```

7. Delete an Audio Stream (Audio Playback Session), once reached end of operation

```
// Callback which provides response to deleteStream
void deleteStreamCallback(ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "deleteStream() returned with error " << static_cast<int>(error)
            << std::endl;
        return;
    }
    std::cout << "deleteStream() succeeded." << std::endl;
    audioPlayStream = nullptr;
}
// Delete an Audio Stream (Audio Playback Session), once reached end of operation.
Status  status = audioManager->deleteStream(audioPlayStream, deleteStreamCallback);
if (status != Status::SUCCESS) {
    std::cout << "deleteStream failed with error" << static_cast<int>(status) << std::endl;
}
```

5.6.13 Compressed audio format playback on voice paths

Audio Manager APIs Sample Reference for compressed audio format playback on voice paths

This Section demonstrates how to use the Audio Manager API for compressed audio format playback on voice paths.

1. Get the AudioFactory instance

```
auto &audioFactory = audioFactory::getInstance();
```

2. Get the AudioManager object and check for audio subsystem Readiness

```
std::promise<ServiceStatus> prom{};
//  Get AudioManager instance.
audioManager = audioFactory.getAudioManager([&prom](ServiceStatus serviceStatus) {
    prom.set_value(serviceStatus);
});
if (!audioManager) {
    std::cout << "Failed to get AudioManager object" << std::endl;
    return;
}

//  Check if audio subsystem is ready
//  If audio subsystem is not ready, wait for it to be ready
ServiceStatus managerStatus = audioManager->getServiceStatus();
if (managerStatus != ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "\nAudio subsystem is not ready, Please wait ..." << std::endl;
    managerStatus = prom.get_future().get();
}

//  Check the service status again.
if (managerStatus == ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "Audio Subsytem is Ready << std::endl;
} else {
    std::cout << "ERROR - Unable to initialize audio subsystem" << std::endl;
    return;
}
```

3. Create an Audio Stream (Audio Playback Session) with Voice Paths direction

```
StreamConfig config;
config.type = StreamType::PLAY;
config.slotId = DEFAULT_SLOT_ID;
config.sampleRate = SAMPLE_RATE;
config.format = AudioFormat::AMRWB_PLUS;
// here both channel selected, this can be selected according to requirement
config.channelTypeMask = (ChannelType::LEFT | ChannelType::RIGHT);
// Since the voice path is selected we dont need to provide any device
// Voice path direction TX is for Voice uplink while direction RX is for Voice downlink
config.voicePaths.emplace_back(Direction::TX);
// Passing Decoder Specific Configuration, refer header file for more details.
AmrwbpParams amrParams{};
if (config.format == AudioFormat::AMRWB_PLUS) {
    amrParams.bitWidth = 16;
    amrParams.frameFormat = AmrwbpFrameFormat::FILE_STORAGE_FORMAT;
    config.formatParams = &amrParams;
} else {
    config.formatParams = nullptr;
}

std::promise<bool> p;
auto status = audioManager->createStream(config,
    [&p,this](std::shared_ptr<IAudioStream> &audioStream, ErrorCode error) {
        if (error == ErrorCode::SUCCESS) {
            audioPlayStream_ = std::dynamic_pointer_cast<IAudioPlayStream>(audioStream);
            p.set_value(true);
        } else {
            p.set_value(false);
            std::cout << "failed to Create a stream" <<std::endl;
        }
    });
if (status == Status::SUCCESS) {
    std::cout << "Request to create stream sent" << std::endl;
} else {
    std::cout << "Request to create stream failed"  << std::endl;
}

if (p.get_future().get()) {
    std::cout<< "Audio Play Stream is Created" << std::endl;
}
```

4. Allocate Stream buffers for Playback operation

```
// Get an audio buffer (can get more than one)
auto streamBuffer = audioPlayStream->getStreamBuffer();
if (streamBuffer != nullptr) {
    // Setting the size that is to be written to stream as the minimum size
    // required by stream. In any case if size returned is 0, using the Maximum
    // Buffer Size, any buffer size between min and max can be used
    size = streamBuffer->getMinSize();
    if (size == 0) {
        size =  streamBuffer->getMaxSize();
    }
    streamBuffer->setDataSize(size);
} else {
    std::cout << "Failed to get Stream Buffer " << std::endl;
}
```

5. Start write operation for playback to start

```
// We need an active voice session to play on voice paths.
// Callback which provides response to write operation.
void writeCallback(std::shared_ptr<IStreamBuffer> buffer, uint32_t bytes, ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "write() returned with error " << static_cast<int>(error) << std::endl;
        // Application needs to resend the Bitstream buffer from leftover position if bytes
```

```
        // consumed are not equal to requested number of bytes to be written.
        pipeLineEmpty_ = false;
    }
    buffer->reset();
    return;
}

// Indication Received only when callback returns with error that bytes written are not equal to
// bytes requested to write. It notifies that pipeline is ready to accept new buffer to write.
void onReadyForWrite() {
    pipeLineEmpty_ = true;
}

// Write desired data into the buffer, the bytes sent as 0x1 for example purpose only.
// First write starts Playback Session.
memset(streamBuffer->getRawBuffer(),0x1,size);
auto status = audioPlayStream->write(streamBuffer, writeCallback);
if (status != telux::common::Status::SUCCESS) {
    std::cout << "write() failed with error" << static_cast<int>(status) << std::endl;
} else {
    std::cout << "Request to write to stream sent" << std::endl;
}
```

6.1 Stop playback operation(STOP_AFTER_PLAY : Stops after playing pending buffers in pipeline)

```
std::promise<bool> p;
auto status = audioPlayStream_->stopAudio(StopType::STOP_AFTER_PLAY, [&p](ErrorCode error) {
    if (error == ErrorCode::SUCCESS) {
        p.set_value(true);
    } else {
        p.set_value(false);
        std::cout << "Failed to stop after playing buffers" << std::endl;
    }
});
if (status == Status::SUCCESS) {
    std::cout << "Request to stop playback after pending buffers Sent" << std::endl;
} else {
    std::cout << "Request to stop playback after pending buffers failed" << std::endl;
}
if (p.get_future().get()) {
    std::cout << "Pending buffers played successful !!" << std::endl;
}
```

6.2 Stop playback operation(FORCE_STOP : Stops immediately, all buffers in pipeline are flushed)

```
std::promise<bool> p;
    auto status = audioPlayStream_->stopAudio(
        StopType::FORCE_STOP, [&p](telux::common::ErrorCode error) {
        if (error == telux::common::ErrorCode::SUCCESS) {
            p.set_value(true);
        } else {
            p.set_value(false);
            std::cout << "Failed to force stop" << std::endl;
        }
        });
    if(status == telux::common::Status::SUCCESS){
        std::cout << "Request to force stop Sent" << std::endl;
    } else {
        std::cout << "Request to force stop failed" << std::endl;
    }
    if (p.get_future().get()) {
            std::cout << "Force Stop successful !!" << std::endl;
    }
```

7. Delete an Audio Stream (Audio Playback Session), once reached end of operation

```
std::promise<bool> p;
Status status = audioManager-> deleteStream(
audioPlayStream_, [&p,this](ErrorCode error) {
    if (error == ErrorCode::SUCCESS) {
        p.set_value(true);
    } else {
        p.set_value(false);
        std::cout << "Failed to delete a stream" << std::endl;
    }
});
if (status == Status::SUCCESS) {
    std::cout << "Request to delete stream sent" << std::endl;
} else {
    std::cout << "Request to delete stream failed"  << std::endl;
}
if (p.get_future().get()) {
    audioPlayStream_= nullptr;
    std::cout << "Audio Play Stream is Deleted" << std::endl;
}
```

5.7 common

The List of sample apps related to common utilities:

- Public Logging API

5.7.1 Public Logging API

Public Logging API

This Section demonstrates how to use the Public Logging API for SDK Applications.

1. Including the Header file

```
#include <telux/common/Log.hpp>
```

2. Calling the Logging API

```
LOG(DEBUG, "startCallResponse: errorCode: ", static_cast<int>(errorCode));
```

5.8 cv2x

The List of sample apps related to cv2x:

- C-V2X Get Status Sample App
- C-V2X RX Sample App
- C-V2X TX Sample App
- Setting verification load using C-V2X Throttle Manager API
- Obtaining filter rate adjustment notification from C-V2X Throttle Manager API

5.8.1 C-V2X Get Status Sample App

C-V2X Get Status Sample App

This Document walks through the cv2x_get_status_app. It demonstrates how to use the C-V2X Radio Manager API to get the C-V2X status.

1. Create a RequestCv2xStatusCallback function

```
// Globals
static Cv2xStatus gCv2xStatus;
static promise<ErrorCode> gCallbackPromise;
static map<Cv2xStatusType, string> gCv2xStatusToString = {
    {Cv2xStatusType::INACTIVE, "Inactive"},
    {Cv2xStatusType::ACTIVE, "Active"},
    {Cv2xStatusType::SUSPENDED, "SUSPENDED"},
    {Cv2xStatusType::UNKNOWN, "UNKNOWN"},
};

// Callback function for Cv2xRadioManager->requestCv2xStatus
static void cv2xStatusCallback(Cv2xStatus status, ErrorCode error) {
    if (ErrorCode::SUCCESS == error) {
        gCv2xStatus = status;
    }
    gCallbackPromise.set_value(error);
}
```

Note: as an alternative, we can use a Lambda function which would eliminate the need for this global scope function.

2. Get a handle to the ICv2xRadioManager object

```
int main {
    // Get handle to Cv2xRadioManager
    auto & cv2xFactory = Cv2xFactory::getInstance();
    auto cv2xRadioManager = cv2xFactory.getCv2xRadioManager();
```

3. Request the C-V2X status

```
    if (Status::SUCCESS != cv2xRadioManager->requestCv2xStatus(cv2xStatusCallback)) {
        cout << "Error : request for C-V2X status failed." << endl;
        return EXIT_FAILURE;
    }
    if (ErrorCode::SUCCESS != gCallbackPromise.get_future().get()) {
        cout << "Error : failed to retrieve C-V2X status." << endl;
        return EXIT_FAILURE;
    }

    // Print status
    if (Cv2xStatusType::ACTIVE == gCv2xStatus.rxStatus) {
        cout << "C-V2X Status:" << endl
             << "  RX : " << gCv2xStatusToString[gCv2xStatus.rxStatus] << endl
             << "  TX : " << gCv2xStatusToString[gCv2xStatus.txStatus] << endl;
    }

    return EXIT_SUCCESS;

} // main
```

5.8.2 C-V2X RX Sample App

C-V2X RX Sample App

This Document walks through the cv2x_rx_app sample application.

1. Create Callback functions for ICv2xRadio and ICv2xRadioManager methods

```
// Globals
static Cv2xStatus gCv2xStatus;
static promise<ErrorCode> gCallbackPromise;
static shared_ptr<ICv2xRxSubscription> gRxSub;
static uint32_t gPacketsReceived = 0u;
static array<char, G_BUF_LEN> gBuf;

// Resets the global callback promise
static inline void resetCallbackPromise(void) {
    gCallbackPromise = promise<ErrorCode>();
}

// Callback function for Cv2xRadioManager->requestCv2xStatus()
static void cv2xStatusCallback(Cv2xStatus status, ErrorCode error) {
    if (ErrorCode::SUCCESS == error) {
        gCv2xStatus = status;
    }
    gCallbackPromise.set_value(error);
}

// Callback function for Cv2xRadio->createRxSubscription() and Cv2xRadio->closeRxSubscription()
static void rxSubCallback(shared_ptr<ICv2xRxSubscription> rxSub, ErrorCode error) {
    if (ErrorCode::SUCCESS == error) {
        gRxSub = rxSub;
    }
    gCallbackPromise.set_value(error);
}
```

Note: We can also use Lambda functions instead of defining global scope callback functions.

2. Get a handle to the ICv2xRadioManager object

```
int main {
    // Get handle to Cv2xRadioManager
    auto & cv2xFactory = Cv2xFactory::getInstance();
    auto cv2xRadioManager = cv2xFactory.getCv2xRadioManager();
```

3. Request the C-V2X status

We want to verify that the C-V2X RX status is ACTIVE before we try to receive data.

```
// Get C-V2X status and make sure Rx is enabled
assert(Status::SUCCESS == cv2xRadioManager->requestCv2xStatus(cv2xStatusCallback));
assert(ErrorCode::SUCCESS == gCallbackPromise.get_future().get());

if (Cv2xStatusType::ACTIVE == gCv2xStatus.rxStatus) {
    cout << "C-V2X RX status is active" << endl;
}
else {
    cerr << "C-V2X RX is inactive" << endl;
    return EXIT_FAILURE;
}
```

4. Get handle to C-V2X Radio

```
auto cv2xRadio = cv2xRadioManager->getCv2xRadio(TrafficCategory::SAFETY_TYPE);
```

5. Wait for C-V2X Radio to be ready

```
if (not cv2xRadio->isReady()) {
    if (Status::SUCCESS == cv2xRadio->onReady().get()) {
        cout << "C-V2X Radio is ready" << endl;
    }
    else {
        cerr << "C-V2X Radio initialization failed." << endl;
        return EXIT_FAILURE;
    }
}
```

6. Create RX Subscription and receive data using RX socket

```
resetCallbackPromise();
assert(Status::SUCCESS == cv2xRadio->createRxSubscription(TrafficIpType::TRAFFIC_NON_IP,
                                                          RX_PORT_NUM,
                                                          rxSubCallback));
assert(ErrorCode::SUCCESS == gCallbackPromise.get_future().get());

// Read from the RX socket in a loop
for (uint32_t i = 0; i < NUM_TEST_ITERATIONS; ++i) {
    // Receive from RX socket
    sampleRx();
}
```

7. Close RX Subcription

We supply the callback in this sample and check its status, but note that it is optional.

```
    resetCallbackPromise();
    assert(Status::SUCCESS == cv2xRadio->closeRxSubscription(gRxSub,
                                                             rxSubCallback));
    assert(ErrorCode::SUCCESS == gCallbackPromise.get_future().get());

    return EXIT_SUCCESS;
} // main
```

5.8.3 C-V2X TX Sample App

C-V2X TX Sample App

This Document walks through the cv2x_tx_app sample application.

1. Create Callback functions for ICv2xRadio and ICv2xRadioManager methods

```
// Globals
static Cv2xStatus gCv2xStatus;
static promise<ErrorCode> gCallbackPromise;
static shared_ptr<ICv2xTxFlow> gSpsFlow;
static array<char, G_BUF_LEN> gBuf;

// Resets the global callback promise
static inline void resetCallbackPromise(void) {
    gCallbackPromise = promise<ErrorCode>();
}

// Callback function for ICv2xRadioManager->requestCv2xStatus()
static void cv2xStatusCallback(Cv2xStatus status, ErrorCode error) {
    if (ErrorCode::SUCCESS == error) {
        gCv2xStatus = status;
    }
    gCallbackPromise.set_value(error);
}

// Callback function for ICv2xRadio->createTxSpsFlow()
static void createSpsFlowCallback(shared_ptr<ICv2xTxFlow> txSpsFlow,
                                  shared_ptr<ICv2xTxFlow> unusedFlow,
                                  ErrorCode spsError,
                                  ErrorCode unusedError) {
    if (ErrorCode::SUCCESS == spsError) {
        gSpsFlow = txSpsFlow;
    }
    gCallbackPromise.set_value(spsError);
}

// Callback for ICv2xRadio->closeTxFlow()
static void closeFlowCallback(shared_ptr<ICv2xTxFlow> flow, ErrorCode error) {
    gCallbackPromise.set_value(error);
```

```
}
```

Note: We can also use Lambda functions instead of defining global scope callback functions.

2. Get a handle to the ICv2xRadioManager object

```
int main {
    // Get handle to Cv2xRadioManager
    auto & cv2xFactory = Cv2xFactory::getInstance();
    auto cv2xRadioManager = cv2xFactory.getCv2xRadioManager();
```

3. Request the C-V2X status

We want to verify that the C-V2X TX status is ACTIVE before we try to send data.

```
// Get C-V2X status and make sure Rx is enabled
assert(Status::SUCCESS == cv2xRadioManager->requestCv2xStatus(cv2xStatusCallback));
assert(ErrorCode::SUCCESS == gCallbackPromise.get_future().get());

if (Cv2xStatusType::ACTIVE == gCv2xStatus.txStatus) {
    cout << "C-V2X TX status is active" << endl;
}
else {
    cerr << "C-V2X TX is inactive" << endl;
    return EXIT_FAILURE;
}
```

4. Get handle to C-V2X Radio

```
auto cv2xRadio = cv2xRadioManager->getCv2xRadio(TrafficCategory::SAFETY_TYPE);
```

5. Wait for C-V2X Radio to be ready

```
if (not cv2xRadio->isReady()) {
    if (Status::SUCCESS == cv2xRadio->onReady().get()) {
        cout << "C-V2X Radio is ready" << endl;
    }
    else {
        cerr << "C-V2X Radio initialization failed." << endl;
        return EXIT_FAILURE;
    }
}
```

6. Create TX SPS flow and send data using TX socket

```
// Set SPS parameters
SpsFlowInfo spsInfo;
spsInfo.priority = Priority::PRIORITY_2;
spsInfo.periodicity = Periodicity::PERIODICITY_100MS;
spsInfo.nbytesReserved = G_BUF_LEN;
spsInfo.autoRetransEnabledValid = true;
spsInfo.autoRetransEnabled = true;

// Create new SPS flow
resetCallbackPromise();
assert(Status::SUCCESS == cv2xRadio->createTxSpsFlow(TrafficIpType::TRAFFIC_NON_IP,
                                                     SPS_SERVICE_ID,
                                                     spsInfo,
                                                     SPS_SRC_PORT_NUM,
                                                     false,
                                                     0,
                                                     createSpsFlowCallback));
assert(ErrorCode::SUCCESS == gCallbackPromise.get_future().get());

// Send message in a loop
for (uint16_t i = 0; i < NUM_TEST_ITERATIONS; ++i) {
    fillBuffer();
    sampleSpsTx();
    usleep(100000u);
}
```

7. Close TX SPS flow

```
    // Deregister SPS flow
    resetCallbackPromise();
    assert(Status::SUCCESS == cv2xRadio->closeTxFlow(gSpsFlow, closeFlowCallback));
    assert(ErrorCode::SUCCESS == gCallbackPromise.get_future().get());

    return EXIT_SUCCESS;
} // main
```

5.8.4 Setting verification load using C-V2X Throttle Manager API

Please follow bellow steps to set the verificaton load

1. Create verification load callback function

```
static std::promise<telux::common::ErrorCode> gCallbackPromise;

// Callback function for Cv2xThrottleManager->setVerificationLoad()
static void cv2xsetVerificationLoadCallback(telux::common::ErrorCode error) {
    std::cout << "error=" << static_cast<int>(error) << std::endl;
    gCallbackPromise.set_value(error);
}
```

2. Create a initialization status callback function

```
bool cv2xTmStatusUpdated = false;
telux::common::ServiceStatus cv2xTmStatus =
    telux::common::ServiceStatus::SERVICE_UNAVAILABLE;
std::condition_variable cv;
std::mutex mtx;

auto statusCb = [&](telux::common::ServiceStatus status) {
    std::lock_guard<std::mutex> lock(mtx);
    cv2xTmStatusUpdated = true;
    cv2xTmStatus = status;
    cv.notify_all();
};
```

3. Get a handle to the ICv2xThrottleManager object

```
// Get handle to Cv2xThrottleManager
auto & cv2xFactory = Cv2xFactory::getInstance();
auto cv2xThrottleManager = cv2xFactory.getCv2xThrottleManager(statusCb);
```

4. Wait for throttle manager to complete initialization

```
std::unique_lock<std::mutex> lck(mtx);
cv.wait(lck, [&] { return cv2xTmStatusUpdated; });
if (telux::common::ServiceStatus::SERVICE_AVAILABLE !=
    cv2xTmStatus) {
    std::cout << "Error: failed to initialize Cv2xThrottleManager." << std::endl;
    return EXIT_FAILURE;
}
```

5. Set the verification load

```
cv2xThrottleManager->setVerificationLoad(load, cv2xsetVerificationLoadCallback);
if (telux::common::ErrorCode::SUCCESS != gCallbackPromise.get_future().get()) {
    std::cout << "Error : failed to set verification load" << std::endl;
    return EXIT_FAILURE;
} else {
    std::cout << "set verification load success" << std::endl;
}
gCallbackPromise = std::promise<telux::common::ErrorCode>();
```

5.8.5 Obtaining filter rate adjustment notification from C-V2X Throttle Manager API

Please follow bellow steps to obtain filter rate adjustment notification

1. Implement ICv2xThrottleManagerListener interface

```
class Cv2xTmListener : public ICv2xThrottleManagerListener {
    public:
        void onFilterRateAdjustment(int rate) override;
 };
```

2. Create a initialization status callback function

```
bool cv2xTmStatusUpdated = false;
telux::common::ServiceStatus cv2xTmStatus =
    telux::common::ServiceStatus::SERVICE_UNAVAILABLE;
std::condition_variable cv;
std::mutex mtx;

std::cout << "Running TM app" << std::endl;

auto statusCb = [&](telux::common::ServiceStatus status) {
    std::lock_guard<std::mutex> lock(mtx);
    cv2xTmStatusUpdated = true;
    cv2xTmStatus = status;
    cv.notify_all();
};
```

3. Get a handle to the ICv2xThrottleManager object

```
int main {
    // Get handle to Cv2xThrottleManager
    auto & cv2xFactory = Cv2xFactory::getInstance();
    auto cv2xThrottleManager = cv2xFactory.getCv2xThrottleManager(statusCb);
```

4. Wait for throttle manager to complete initialization

```
std::unique_lock<std::mutex> lck(mtx);
cv.wait(lck, [&] { return cv2xTmStatusUpdated; });
if (telux::common::ServiceStatus::SERVICE_AVAILABLE !=
    cv2xTmStatus) {
    std::cout << "Error: failed to initialize Cv2xThrottleManager." << std::endl;
    return EXIT_FAILURE;
}
```

5. Instantiate Cv2xTmListener

```
auto listener = std::make_shared<Cv2xTmListener>();
```

6. Register listener

```
if (cv2xThrottleManager->registerListener(listener) !=
    telux::common::Status::SUCCESS) {
        std::cout << "Failed to register listener" << std::endl;
        return EXIT_FAILURE;
}
```

7. Wait for filter rate adjustment notification

~~~~~~{.cpp} void Cv2xTmListener::onFilterRateAdjustment(int rate) { std::cout << "Updated rate: " << rate << std::endl; }

5.9 tel

The List of sample apps related to telephony:

- Make Call
- Make eCall
- Make Request Voice Service State of the device
- Set radio power of the device
- Using Request Service Domain Preference API
- Using Subscription Manager APIs
- Using Card Service APIs
- Using SAP APIs
- Using Request Network Selection Mode API
- Remote SIM Manager API Sample Reference
- Using Remote SIM Reference Apps
- Sending SMS
- Listening to Incoming SMS
- rsp

5.9.1 Make Call

Making a Voice Call

Please follow below steps to make a voice call

1. Get the PhoneFactory and PhoneManager instances

```
auto &phoneFactory = PhoneFactory::getInstance();
auto phoneManager = phoneFactory.getPhoneManager();
```

2. Check if telephony subsystem is ready

```
bool subSystemsStatus = phoneManager->isSubsystemReady();
```

2.1 If telephony subsystem is not ready, wait for it to be ready

Telephony subsystems is to make sure that device is ready for services like Phone, SMS and others. if subsystems were not ready, wait for unconditionally.

```
if(!subSystemsStatus) {
```
~~~~~~

```
   std::future<bool> f = phoneManager->onSubsystemReady();
   subSystemsStatus = f.get();
}
```

3. Instantiate Phone and call manager

```
auto phone = phoneManager->getPhone();
std::shared_ptr<ICallManager> callManager = phoneFactory.getCallManager();
```

4. Initialize phoneId with default value

```
int phoneId = DEFAULT_PHONE_ID;
```

5. Instantiate dial call instance - this is optional

```
std::shared_ptr<DialCallback> dialCb = std::make_shared<DialCallback> ();
```

5.1 Implement IMakeCallCallback interface to receive response for the dial request optional

```
class DialCallback : public IMakeCallCallback {
public:
   void makeCallResponse(ErrorCode error, std::shared_ptr<ICall> call) override;
};

void DialCallback::makeCallResponse(ErrorCode error, std::shared_ptr<ICall> call) {
   // will be invoked with response of makeCall operation
}
```

6. Send a dial request

```
if(callManager) {
   std::string phoneNumber("+18989531755");
   auto makeCallStatus = callManager->makeCall(phoneId, phoneNumber, dialCb);
   std::cout << "Dial Call Status:" << (int)makeCallStatus << std::endl;
}
```

5.9.2 Make eCall

Making eCall (Emergency E112)

Please follow below steps to make an emergency call(eCall).

1. Get the PhoneFactory and PhoneManager instances.

```
auto &phoneFactory = PhoneFactory::getInstance();
auto phoneManager = phoneFactory.getPhoneManager();
```

2. Check if telephony subsystem is ready

```
bool subSystemsStatus = phoneManager->isSubsystemReady();
```

2.1 If telephony subsystem is not ready, wait for it to be ready

Telephony subsystems is to make sure that device is ready for services like Phone, SMS and others. if subsystems were not ready, wait for unconditionally.

```
if(!subSystemsStatus) {
   std::future<bool> f = phoneManager->onSubsystemReady();
   subSystemsStatus = f.get();
```

```
}
```

3. Instantiate Phone and call manager

```
auto phone = phoneManager->getPhone();
std::shared_ptr<ICallManager> callManager = phoneFactory.getCallManager();
```

5. Initialize phoneId with default value

```
int phoneId = DEFAULT_PHONE_ID;
```

6. Instantiate dial callback instance - this is optional

```
std::shared_ptr<DialCallback> dialCb = std::make_shared<DialCallback> ();
```

6.1. implement IMakeCallCallback interface to receive response for the dial request - optional

```
class DialCallback : public IMakeCallCallback {
public:
   void makeCallResponse(ErrorCode error, std::shared_ptr<ICall> call) override;
};

void DialCallback::makeCallResponse(ErrorCode error, std::shared_ptr<ICall> call) {
   // will be invoked with response of makeECall operation
}
```

7. Initialize the data required for eCall such as eCallMsdData,emergencyCategory and eCallVariant

```
ECallCategory emergencyCategory = ECallCategory::VOICE_EMER_CAT_AUTO_ECALL;
ECallVariant eCallVariant = ECallVariant::ECALL_TEST;
// Instantiate ECallMsdData structure and populate it with valid information
// such as Latitude, Longitude etc.
// Parameter values mentioned here are for illustrative purposes only.
ECallMsdData eCallMsdData;
eCallMsdData.msdData.messageIdentifier = 1; // Each MSD message should bear a unique id
eCallMsdData.optionals.recentVehicleLocationN1Present = true;
eCallMsdData.optionals.recentVehicleLocationN2Present = true;
eCallMsdData.optionals.numberOfPassengersPresent = 2;
eCallMsdData.msdData.control.automaticAvtivation = true;
eCallMsdData.control.testCall = true;
eCallMsdData.control.positionCanBeTrusted = true;
eCallMsdData.control.vehicleType = ECallVehicleType::PASSENGER_VEHICLE_CLASS_M1;
eCallMsdData.msdData.vehicleIdentificationNumber.isowmi = "ECA";
eCallMsdData.msdData.vehicleIdentificationNumber.isovds = "LLEXAM";
eCallMsdData.msdData.vehicleIdentificationNumber.isovisModelyear = "P";
eCallMsdData.msdData.vehicleIdentificationNumber.isovisSeqPlant = "LE02013";
eCallMsdData.msdData.vehiclePropulsionStorage.gasolineTankPresent = true;
eCallMsdData.msdData.vehiclePropulsionStorage.dieselTankPresent = false;
eCallMsdData.vehiclePropulsionStorage.compressedNaturalGas = false;
eCallMsdData.vehiclePropulsionStorage.liquidPropaneGas = false;
eCallMsdData.vehiclePropulsionStorage.electricEnergyStorage = false;
eCallMsdData.vehiclePropulsionStorage.hydrogenStorage = false;
eCallMsdData.vehiclePropulsionStorage.otherStorage = false;
eCallMsdData.timestamp = 1367878452;
eCallMsdData.vehicleLocation.positionLatitude = 123;
eCallMsdData.vehicleLocation.positionLongitude = 1234;
eCallMsdData.msdData.vehicleDirection = 4;
eCallMsdData.recentVehicleLocationN1.latitudeDelta = false;
eCallMsdData.recentVehicleLocationN1.longitudeDelta = 0;
eCallMsdData.recentVehicleLocationN2.latitudeDelta = true;
eCallMsdData.recentVehicleLocationN2.longitudeDelta = 0;
```

8. Send a eCall request

```
if(callManager) {
   auto makeCallStatus = callManager->makeECall(phoneId, eCallMsdData, emergencyCategory,
                                                eCallVariant, dialCb);
   std::cout << "Dial ECall Status:" << (int)makeCallStatus << std::endl;
}
```

5.9.3 Make Request Voice Service State of the device

Request Voice Service State of the device

Please follow below steps to get voice service state notifications.

1. Get the PhoneFactory and PhoneManager instances

```
auto &phoneFactory = PhoneFactory::getInstance();
auto phoneManager = phoneFactory.getPhoneManager();
```

2. Check if telephony subsystem is ready

```
bool subSystemsStatus = phoneManager->isSubsystemReady();
```

2.1 If telephony subsystem is not ready, wait for it to be ready

If subsystem is not ready, wait unconditionally.

```
if (!subSystemsStatus) {
   std::future<bool> f = phoneManager->onSubsystemReady();
   subSystemsStatus = f.get();
}
```

3. Instantiate Phone

```
auto phone = phoneManager->getPhone();
```

4. Check for radio state

If radio is in OFF state turn it to ON in order to perform any operations on the phone. Either wait for radio to be turned on or else pass the callback to receive the response for setRadioPower

```
RadioState radioState = phone->getRadioState();
if(radioState == RadioState::RADIO_STATE_OFF){
   phone->setRadioPower(true);
}
```

5. Implement IVoiceServiceStateCallback interface

```
class MyVoiceServiceStateCallback : public telux::tel::IVoiceServiceStateCallback {
public:
   void voiceServiceStateResponse(const std::shared_ptr<telux::tel::VoiceServiceInfo> &serviceInfo,
      telux::common::ErrorCode error) override;
};
```

6. Instantiate MyVoiceServiceStateCallback

```
auto myVoiceServiceStateCallback = std::make_shared<MyVoiceServiceStateCallback>();
```

7. Send voice service state request.

```
phone->requestVoiceServiceState(myVoiceServiceStateCallback);
```

8. After receiving voiceServiceStateResponse in MyVoiceServiceStateCallback, the status of voice registration can be accessed by using the VoiceServiceInfo.

5.9.4 Set radio power of the device

Set radio power of the device

Please follow below steps to Radio Power state notifications.

1. Get the PhoneFactory and PhoneManager instances

```
auto &phoneFactory = PhoneFactory::getInstance();
auto phoneManager = phoneFactory.getPhoneManager();
```

2. Check if telephony subsystem is ready

```
bool subSystemsStatus = phoneManager->isSubsystemReady();
```

2.1 If telephony subsystem is not ready, wait for it to be ready

If subsystem is not ready, wait unconditionally.

```
if (!subSystemsStatus) {
   std::future<bool> f = phoneManager->onSubsystemReady();
   subSystemsStatus = f.get();
}
```

3. Instantiate Phone

```
auto phone = phoneManager->getPhone();
```

4. Implement IPhoneListener interface to receive service state change notifications

```
class MyPhoneListener : public telux::tel::IPhoneListener {
public:
    void onRadioStateChanged(int phoneId, telux::tel::RadioState radiostate) {
    }
   ~MyPhoneListener() {
   }
}
```

4.1 Instantiate MyPhoneListener

```
auto myPhoneListener = std::make_shared<MyPhoneListener>();
```

5. Register for phone info updates

```
phoneManager->registerListener(myPhoneListener);
```

6. Implement ICommandResponseCallback to receive the status of setRadioPower API call

```
class MyPhoneCommandResponseCallback : public ICommandResponseCallback {
public:
   MyPhoneCommandResponseCallback() {
   }
   void commandResponse(ErrorCode error) override;
};
```

7. Instantiate MyPhoneCommandResponseCallback

```
auto myPhoneCommandCb = std::make_shared<MyPhoneCommandResponseCallback>();
```

8. Set the Radio power ON/OFF.

```
phone->setRadioPower(true, myPhoneCommandCb);
```

9. Command response callback is invoked with error code indicating SUCCESS or FAILURE of the operation.

```
MyPhoneCommandResponseCallback::commandResponse(ErrorCode error) {
   if(error == ErrorCode::SUCCESS) {
      std::cout << "Set Radio Power On request is successful ";
   } else {
      std::cout << "Set Radio Power On request failed with error " << static_cast<int>(error);
   }
}
```

5.9.5 Using Request Service Domain Preference API

Using Request Service Domain Preference API

Please follow below steps to request service domain preference

1. Get phone factory and serving system manager instances

```
auto &phoneFactory = telux::tel::PhoneFactory::getInstance();
auto servingSystemMgr
   = phoneFactory.getServingSystemManager(DEFAULT_SLOT_ID);
```

2. Wait for the serving subsystem initialization

```
bool subSystemStatus = servingSystemMgr->isSubsystemReady();
```

2.1 If serving subsystem is not ready, wait for it to be ready

```
if(!subSystemsStatus) {
   std::cout << "Serving subsystem is not ready" << std::endl;
   std::cout << "wait unconditionally for it to be ready " << std::endl;
   std::future<bool> f = servingSystemMgr->onSubsystemReady();
   subSystemsStatus = f.get();
}
```

3. Exit the application, if SDK is unable to initialize serving subsystem

```
if(subSystemsStatus) {
   std::cout << " *** Serving subsystem ready *** " << std::endl;
} else {
   std::cout << " *** ERROR - Unable to initialize serving subsystem"
             << std::endl;
   return 1;
}
```

6. Implement response callback to receive response for request service domain preference

```
class ServiceDomainResponseCallback {
public:
   void serviceDomainResponse(telux::tel::ServiceDomainPreference preference,
                              telux::common::ErrorCode errorCode) {
      if(errorCode == telux::common::ErrorCode::SUCCESS) {
         std::cout << "Service domain preference: "
                   << static_cast<int>(preference)
                   << std::endl;
      } else {
         std::cout << "\n setServiceDomainPreference failed, ErrorCode: "
                   << static_cast<int>(errorCode)
                   << std::endl;
      }
   }
};
```

7. Send request service domain preference request along with required function object

```
if(servingSystemMgr) {
   servingSystemMgr->requestServiceDomainPreference(
      ServiceDomainResponseCallback::serviceDomainResponse);
   std::cout << static_cast<int>(status) <<std::endl;
}
```

8. Receive callback for request service domain preference request in serviceDomainResponse function

5.9.6 Using Subscription Manager APIs

Using Subscription Manager APIs

Please follow below steps to use Subscription Manager APIs to get Subscription Information.

1. Get the PhoneFactory and SubscriptionManager instances

```
auto &phoneFactory = PhoneFactory::getInstance();
std::shared_ptr<ISubscriptionManager> subscriptionMgr = phoneFactory.getSubscriptionManager();
```

2. Wait for the Subscription subsystem to ready

```
if(!subscriptionMgr->isSubsystemReady()) {
   auto subSystemStatus = subscriptionMgr->onSubsystemReady().get();
   if(!subSystemStatus){
      // if Subscription subsystem fails, then exit the application.
      exit(1);
   }
}
```

3. Get the Subscription information

```
std::shared_ptr<ISubscription> subscription = subscriptionMgr->getSubscription();

if(subscription != nullptr) {
   std::cout << "Subscription Details" << std::endl;
   std::cout << " CarrierName : " << subscription->getCarrierName() << std::endl;
   std::cout << " CountryISO : " << subscription->getCountryISO() << std::endl;
   std::cout << " PhoneNumber : " << subscription->getPhoneNumber() << std::endl;
   std::cout << " IccId : " << subscription->getIccId() << std::endl;
   std::cout << " Mcc : " << subscription->getMcc() << std::endl;
```

```
   std::cout << " Mnc : " << subscription->getMnc() << std::endl;
   std::cout << " SlotId : " << subscription->getSlotId() << std::endl;
   std::cout << " SubscriptionId : " << subscription->getSubscriptionId() << std::endl;
}
```

5.9.7 Using Card Service APIs

Using Card Service APIs

Please follow below steps to use Card Service APIs to transmit APDU

1. Get the PhoneFactory and CardManager instances.

```
auto &phoneFactory = PhoneFactory::getInstance();
std::shared_ptr<ICardManager> cardManager = phoneFactory.getCardManager();
```

2. Wait for the telephony subsystem initialization.

```
bool subSystemsStatus = cardManager->isSubsystemReady();
if(!subSystemsStatus) {
   std::cout << "Telephony subsystem is not ready, wait for it to be ready " << std::endl;
   std::future<bool> f = cardManager->onSubsystemReady();
   auto status = f.wait_for(std::chrono::seconds(5));
   if(status == std::future_status::ready) {
      subSystemsStatus = true;
   }
}
```

3. Get SlotCount, SlotIds and Card instance

```
int slotCount;
cardManager->getSlotCount(slotCount);
std::cout << "Slots Count is :" << slotCount << std::endl;
std::vector<int> slotIds;
cardManager->getSlotIds(slotIds);
std::cout << "Slot Ids are : { ";
for(auto id : slotIds) {
   std::cout << id << " ";
}
std::cout << "}" << std::endl;
std::shared_ptr<ICard> cardImpl = cardManager->getCard(slotIds.front());
```

4. Get supported applications from the card

```
std::vector<std::shared_ptr<ICardApp>> applications;
if(cardImpl) {
   std::cout << "\nApplications available are : " << std::endl;
   applications = cardImpl->getApplications();
   for(auto cardApp : applications) {
      std::cout << "AppId : " << cardApp->getAppId() << std::endl;
   }
}
```

5. Instantiate optional IOpenLogicalChannelCallback, ICommandResponseCallback and ITransmitApduResponseCallback

```
auto myOpenLogicalCb = std::make_shared<MyOpenLogicalChannelCallback>();
auto myCloseLogicalCb = std::make_shared<MyCloseLogicalChannelCallback>();
auto myTransmitApduResponseCb = std::make_shared<MyTransmitApduResponseCallback>();
```

5.1 Implementation of ICardChannelCallback interface for receiving notifications on card event like open logical channel

```
class MyOpenLogicalChannelCallback : public ICardChannelCallback {
public:
   void onChannelResponse(int channel, IccResult result, ErrorCode error) override;
};

void MyOpenLogicalChannelCallback::onChannelResponse(int channel, IccResult result,
                                                     ErrorCode error) {
   std::cout << "onChannelResponse, error: " << (int)error << std::endl;
   std::unique_lock<std::mutex> lock(eventMutex);
   errorCode = error;
   openChannel = channel;
   std::cout << "onChannelResponse: " << result.toString() << std::endl;
   if(cardEventExpected == CardEvent::OPEN_LOGICAL_CHANNEL) {
      std::cout << "Card Event OPEN_LOGICAL_CHANNEL found with code :" << int(error) << std::endl;
      eventCV.notify_one();
   }
}
```

5.2. Implementation of ICommandResponseCallback interface for receiving notifications on card event like close logical channel

```
class MyCloseLogicalChannelCallback : public ICommandResponseCallback {
public:
   void commandResponse(ErrorCode error) override;
};

void MyCloseLogicalChannelCallback::commandResponse(ErrorCode error) {
   std::cout << "commandResponse, error: " << (int)error << std::endl;
   std::unique_lock<std::mutex> lock(eventMutex);
   errorCode = error;
   if(cardEventExpected == CardEvent::CLOSE_LOGICAL_CHANNEL) {
      std::cout << "Card Event CLOSE_LOGICAL_CHANNEL found with code :" << int(error) << std::endl;
      eventCV.notify_one();
   }
}
```

5.3. Implementation of ICardCommandCallback interface for receiving notifications on card event like transmit apdu logical channel and transmit apdu basic channel

```
class MyTransmitApduResponseCallback : public ICardCommandCallback {
public:
   void onResponse(IccResult result, ErrorCode error) override;
};

void MyTransmitApduResponseCallback::onResponse(IccResult result, ErrorCode error) {
   std::cout << "onResponse, error: " << (int)error << std::endl;
   std::unique_lock<std::mutex> lock(eventMutex);
   errorCode = error;
   std::cout << "onResponse:  " << result.toString() << std::endl;
   if(cardEventExpected == CardEvent::TRANSMIT_APDU_CHANNEL) {
      std::cout << "Card Event TRANSMIT_APDU_CHANNEL found with code :" << int(error) << std::endl;
      eventCV.notify_one();
   }
}
```

6. Open Logical Channel and wait for request to complete

```
std::string aid;
for(auto app : applications) {
   if(app->getAppType() == APPTYPE_USIM) {
      aid = app->getAppId();
```

```
      break;
   }
}
cardImpl->openLogicalChannel(aid, myOpenLogicalCb);
std::cout << "Opening Logical Channel to Transmit the APDU..." << std::endl;
```

7. Transmit Apdu on Logical Channel, wait for request to complete

```
cardImpl->transmitApduLogicalChannel(openChannel, CLA, INSTRUCTION, P1, P2, P3, DATA,
                                     myTransmitApduResponseCb);
std::cout << "Transmit APDU request made..." << std::endl;
```

8. Close the opened logical channel and wait for the completion

```
cardImpl->closeLogicalChannel(openChannel, myCloseLogicalCb);
std::cout << "Close the Logical Channel..." << std::endl;
```

9. Transmit Apdu on Basic Channel and wait for completion

```
cardImpl->transmitApduBasicChannel(CLA, INSTRUCTION, P1, P2, P3, DATA, myTransmitApduResponseCb);
std::cout << "Transmit APDU request on Basic channel made..." << std::endl;
```

5.9.8 Using SAP APIs

Using SAP APIs

Please follow below steps to use SAP APIs to send APDU and listen to SAP events

1. Get the PhoneFactory and PhoneManager instances.

```
auto &phoneFactory = PhoneFactory::getInstance();
auto phoneManager = phoneFactory.getPhoneManager();
```

2. Wait for the telephony subsystem initialization.

```
bool subSystemsStatus = cardManager->isSubsystemReady();
if(!subSystemsStatus) {
   std::cout << "Telephony subsystem is not ready, wait for it to be ready " << std::endl;
   std::future<bool> f = cardManager->onSubsystemReady();
   auto status = f.wait_for(std::chrono::seconds(5));
   if(status == std::future_status::ready) {
      subSystemsStatus = true;
   }
}
```

3. Get default Sap Card Manager instance

```
std::shared_ptr<ISapCardManager> sapCardMgr = phoneFactory.getSapCardManager();
```

4. Instantiate ICommandResponseCallback, IAtrResponseCallback and ISapCardCommandCallback

```
auto mySapCmdResponseCb = std::make_shared<MySapCommandResponseCallback>();
auto myAtrCb = std::make_shared<MyAtrResponseCallback>();
auto myTransmitApduResponseCb = std::make_shared<MySapTransmitApduResponseCallback>();
```

4.1 Implementation of ICommandResponseCallback interface for receiving notifications on sap events like open connection and close connection

```
class MySapCommandResponseCallback : public ICommandResponseCallback {
public:
   void commandResponse(ErrorCode error);
};

void MySapCommandResponseCallback::commandResponse(ErrorCode error) {
   std::cout << "commandResponse, error: " << (int)error << std::endl;
}
```

4.2 Implementation of IAtrResponseCallback interface for receiving notification on sap event like request answer to reset(ATR)

```
class MyAtrResponseCallback : public IAtrResponseCallback {
public:
   void atrResponse(std::vector<int> responseAtr, ErrorCode error);
};

void MyAtrResponseCallback::atrResponse(std::vector<int> responseAtr, ErrorCode error) {
   std::cout << "atrResponse, error: " << (int)error << std::endl;
}
```

4.3 Implementation of ISapCardCommandCallback interface for receiving notification on sap event like transmit apdu.

```
class MySapTransmitApduResponseCallback : public ISapCardCommandCallback {
public:
   void onResponse(IccResult result, ErrorCode error);
};

void MySapTransmitApduResponseCallback::onResponse(IccResult result, ErrorCode error) {
   std::cout << "transmitApduResponse, error: " << (int)error << std::endl;
}
```

5. Open Sap connection and wait for request to complete

```
sapCardMgr->openConnection(SapCondition::SAP_CONDITION_BLOCK_VOICE_OR_DATA, mySapCmdResponseCb);
std::cout << "Opening SAP connection to Transmit the APDU..." << std::endl;
```

6. request sap ATR and wait for complete

```
sapCardMgr->requestAtr(myAtrCb);
```

7. send sap apdu and wait for the request to complete

```
std::cout << "Transmit Sap APDU request made..." << std::endl;
Status ret = sapCardMgr->transmitApdu(CLA, INSTRUCTION, P1, P2, LC, DATA, 0,
                                      myTransmitApduResponseCb);
```

8. close sap connection and wait for the request to complete

```
sapCardMgr->closeConnection(mySapCmdResponseCb);
```

5.9.9 Using Request Network Selection Mode API

Using Request Network Selection Mode API

Please follow below steps to request network selection mode

1. Get phone factory and network selection manager instances

```
auto &phoneFactory = telux::tel::PhoneFactory::getInstance();
auto networkMgr
   = phoneFactory.getNetworkSelectionManager(DEFAULT_SLOT_ID);
```

2. Wait for the network selection subsystem initialization

```
bool subSystemStatus = networkMgr->isSubsystemReady();
```

2.1 If network selection subsystem is not ready, wait for it to be ready

```
if(!subSystemStatus) {
   std::cout << "network selection subsystem is not ready" << std::endl;
   std::cout << "wait unconditionally for it to be ready " << std::endl;
   std::future<bool> f = networkMgr->onSubsystemReady();
   // If we want to wait unconditionally for network selection subsystem to be ready
   subSystemStatus = f.get();
}
```

3. Exit the application, if SDK is unable to initialize network selection subsystem

```
if(subSystemsStatus) {
   std::cout << " *** Network selection subsystem ready *** " << std::endl;
} else {
   std::cout << " *** ERROR - Unable to initialize network selection subsystem" << std::endl;
   return 1;
}
```

4. Implement response callback to receive response for request network selection mode

```
class SelectionModeResponseCallback {
public:
   void selectionModeResponse(
      telux::tel::NetworkSelectionMode networkSelectionMode,
      telux::common::ErrorCode errorCode) {
      if(errorCode == telux::common::ErrorCode::SUCCESS) {
         std::cout << "Network selection mode: "
                   << static_cast<int>(networkSelectionMode)
                   << std::endl;
      } else {
         std::cout << "\n requestNetworkSelectionMode failed, ErrorCode: "
                   << static_cast<int>(errorCode)
                   << std::endl;
      }
   }
};
```

5. Send requestNetworkSelectionMode along with required function object

```
if(networkMgr) {
   auto status = networkMgr->requestNetworkSelectionMode(
      SelectionModeResponseCallback::selectionModeResponse);
   std::cout << static_cast<int>(status) <<std::endl;
   }
}
```

6. Receive callback for get network selection mode request in selectionModeResponse function

5.9.10 Remote SIM Manager API Sample Reference

Remote SIM Manager API Sample Reference

This section demonstrates how to use the Remote SIM Manager API for remote SIM card operations.

1. Get the PhoneFactory and RemoteSimManager instances

```
#include <telux/tel/PhoneFactory.hpp>

using namespace telux::common;
using namespace telux::tel;

PhoneFactory &phoneFactory = PhoneFactory::getInstance();
std::shared_ptr<IRemoteSimManager> remoteSimMgr =
    phoneFactory.getRemoteSimManager(DEFAULT_SLOT_ID);
```

2. Instantiate and register RemoteSimListener

```
std::shared_ptr<IRemoteSimListener> listener = std::make_shared<RemoteSimListener>();
remoteSimMgr.registerListener(listener);
```

2.1 Implementation of IRemoteSimListener interface for receiving Remote SIM notifications

```
class RemoteSimListener : public IRemoteSimListener {
public:
    void onApduTransfer(const unsigned int id, const std::vector<uint8_t> &apdu) override {
        // Send APDU to SIM card
    }
    void onCardConnect() override {
        // Connect to SIM card and request AtR
    }
    void onCardDisconnect() override {
        // Disconnect from SIM card
    }
    void onCardPowerUp() override {
        // Power up SIM card and request AtR
    }
    void onCardPowerDown() override {
        // Power down SIM card
    }
    void onCardReset() override {
        // Reset SIM card
    }
    void onServiceStatusChange(ServiceStatus status) {
        // Handle case where modem goes down or comes up
    }
};
```

2.1 Implementation of event callback for asynchronous requests

```
void eventCallback(ErrorCode errorCode) {
    std::cout << "Received event response with errorcode " << static_cast<int>(errorCode)
        << std::endl;
}
```

3. Wait for Remote SIM subsystem initialization

```
int timeoutSec = 5;
if (!(remoteSimMgr->isSubsystemReady())) {
    auto f = remoteSimMgr->onSubsystemReady();
    if (f.wait_for(std::chrono::seconds(timeoutSec)) != std::future_status::ready) {
        std::cout << "Remote SIM subsystem did not initialize!" << std::endl;
    }
}
```

4. Send connection available event request

When the remote card is available and ready, make it available to the modem by sending a connection available request.

```
if (remoteSimMgr->sendConnectionAvailable(eventCallback) != Status::SUCCESS) {
    std::cout << "Failed to send connection available request!" << std::endl;
}
```

5. Send card reset request after receiving onCardConnect() notification from listener

You will receive an onCardConnect notification on the listener when the modem accepts the connection.

```
// After connecting to SIM card, requesting AtR, and receiving response with AtR bytes

if (remoteSimMgr->sendCardReset(atr, eventCallback) != Status::SUCCESS) {
    std::cout << "Failed to send card reset request!" << std::endl;
}
```

6. Send response APDU after receiving onTransmitApdu() notification from listener

```
// After sending command APDU to SIM and receiving the response

if (remoteSimMgr->sendApdu(id, apdu, true, apdu.size(), 0, eventCallback) != Status::SUCCESS) {
    std::cout << "Failed to send response APDU!" << std::endl;
}
```

7. Send connection unavailable request before exiting

When the card becomes unavailable (or before you exit), tear down the connection with the modem.

```
if (remoteSimMgr->sendConnectionUnavailable() != Status::SUCCESS) {
    std::cout << "Failed to send connection unavailable request!" << std::endl;
}
```

5.9.11 Using Remote SIM Reference Apps

Using Remote SIM Reference Apps

This section describes how to use the provided Remote SIM reference apps – remote-sim-daemon and sap-card-provider. The remote-sim-daemon app will run on the device without a SIM, while the sap-card-provider app will run on the device with a SIM. The two apps will communicate over a standard IP Ethernet connection, providing the WWAN capabilities of the remote SIM card to the device without a SIM inserted. Both devices are required to have support for the Telematics SDK and to be connected to each other via Ethernet.

Required Items

It is assumed that both devices are configured to enable the necessary features to support Remote SIM capabilities.

1. Set Up the Ethernet connection

First, connectivity between the two devices needs to be established.

1.1 Disable Automatic Configuration IP Address

Depending on the device type, it may be necessary to first disable the auto-config IP (169.254.x.x) address on both devices.

```
1 brctl delif bridge0 eth0
```

1.2 Configure the IP Address on Both Devices

```
1 ifconfig eth0 192.168.1.2
```

The device with a SIM can use 192.168.1.3.

NOTE: Depending on the device, it may be necessary to execute steps 1.1 and 1.2 again whenever either device restarts or is disconnected, due to auto-config settings.

2. Run the Remote SIM Daemon in the Background on the Device Without a SIM

```
1 remote-sim-daemon &
```

The -d and -s flags can also be used for debugging purposes (use -h for usage instructions).

3. Run the Sap Card Provider on the Device with a SIM

```
1 sap-card-provider -i 192.168.1.2
```

Use the -i flag to provide the IP address of the device running remote-sim-daemon. The -d and -s flags can also be used for debugging purposes (use -h for usage instructions).

5.9.12 Sending SMS

Sending SMS

Please follow below steps to send an SMS to any mobile number.

1. Get the PhoneFactory and PhoneManager instances.

```
auto &phoneFactory = PhoneFactory::getInstance();
auto phoneManager = phoneFactory.getPhoneManager();
```

2. Check if telephony subsystem is ready

```
bool subSystemsStatus = phoneManager->isSubsystemReady();
```

2.1 If telephony subsystem is not ready, wait for it to be ready

Telephony subsystems is to make sure that device is ready for services like Phone, SMS and others. if subsystems were not ready, wait for unconditionally.

```
if(!subSystemsStatus) {
   std::future<bool> f = phoneManager->onSubsystemReady();
   subSystemsStatus = f.get();
}
```

3. Instantiate SMS sent and delivery callback

```
auto smsSentCb = std::make_shared<SmsCallback>();
auto smsDeliveryCb = std::make_shared<SmsDeliveryCallback>();
```

3.1 Implement ICommandResponseCallback interface to know SMS sent and Delivery status

```
class SmsCallback : public ICommandResponseCallback {
public:
   void commandResponse(ErrorCode error) override;
};

void SmsCallback::commandResponse(ErrorCode error) {
   std::cout << "onSmsSent callback" << std::endl;
}

class SmsDeliveryCallback : public ICommandResponseCallback {
public:
   void commandResponse(ErrorCode error) override;
};

void SmsDeliveryCallback::commandResponse(ErrorCode error) {
   std::cout << "SMS Delivery callback" << std::endl;
}
```

4. Get default SMS Manager instance

```
std::shared_ptr<ISmsManager> smsManager = phoneFactory.getSmsManager();
```

5. Send an SMS using ISmsManager by passing the text and receiver number along with required callback

```
if(smsManager) {
   std::string receiverAddress("+18989531755");
   std::string message("TEST message");
   smsManager->sendSms(message, receiverAddress, smsSentCb, smsDeliveryCb);
}
```

6. Receive responses for sendSms request

5.9.13 Listening to Incoming SMS

Listening to Incoming SMS

Please follow below steps to listen for incoming SMS

1. Implement ISmsListener interface to receive incoming SMS

```
class MySmsListener : public ISmsListener {
public:
   void onIncomingSms(int phoneId, std::shared_ptr<SmsMessage> message) override;
};
```

```
void MySmsListener::onIncomingSms(int phoneId, std::shared_ptr<SmsMessage> smsMsg) {
   std::cout << "MySmsListener::onIncomingSms from PhoneId : " << phoneId << std::endl;
   std::cout << "smsReceived: From : " << smsMsg->toString() << std::endl;
}
```

2. Get the PhoneFactory and PhoneManager instances

```
auto &phoneFactory = PhoneFactory::getInstance();
auto phoneManager = phoneFactory.getPhoneManager();
```

3. Check if telephony subsystem is ready

```
bool subSystemStatus = phoneManager->isSubsystemReady();
```

4. Exit the application, if SDK is unable to initialize telephony subsystems

```
if(subSystemStatus) {
   std::cout << " *** Subsystem Ready *** " << std::endl;
} else {
   std::cout << " *** ERROR - Unable to initialize telephony subsystem" << std::endl;
   return 1;
}
```

5. Instantiate global ISmsListener

```
auto myPhoneListener = std::make_shared<MyPhoneListener>();
```

6. Get default SMS Manager instance

```
std::shared_ptr<ISmsManager> smsMgr = phoneFactory.getSmsManager();
```

7. Register for incoming SMS

```
if(smsMgr) {
   smsMgr->registerListener(mySmsListener);
}
```

8. Wait for incoming SMS

```
std::cout << " *** wait for MyPhoneListener::onIncomingSms() to be triggered*** " << std::endl;
std::cout << " *** Press enter to exit the application *** " << std::endl;
std::string input;
std::getline(std::cin, input);
return 0;
```

5.9.14 rsp

The list of sample apps related to Remote SIM Provisioning:

- Using Remote SIM Provisioning API
- Using Remote SIM Provisioning Reference Service

5.9.14.1 Using Remote SIM Provisioning API

Using Remote SIM Provisioning API

This section demonstrates how to use the Remote SIM Provisioning API for performing SIM profile management operations on the eUICC such as add profile, enable/disable profile, delete profile, query profile list, configure server address and perform memory reset.

1. Get phone factory, SIM profile manager and Card manager instance

```
auto &phoneFactory = telux::tel::PhoneFactory::getInstance();
auto simProfileManager = phoneFactory.getSimProfileManager();
auto cardManager = phoneFactory.getCardManager();
```

2. Check if SIM profile subsystem is ready

```
bool subSystemStatus = simProfileManager->isSubsystemReady();
```

2.1 If SIM profile manager subsystem is not ready, wait for it to be ready

```
if(!subSystemsStatus) {
   std::cout << "SIM profile manager subsystem is not ready" << std::endl;
   std::cout << "wait unconditionally for it to be ready " << std::endl;
   std::future<bool> f = simProfileManager->onSubsystemReady();
   subSystemsStatus = f.get();
}
```

3. Check if card subsystem is ready

```
bool subSystemStatus = cardManager->isSubsystemReady();
```

3.1 If card manager subsystem is not ready, wait for it to be ready

```
if(!subSystemsStatus) {
   std::cout << "Card manager subsystem is not ready" << std::endl;
   std::cout << "wait unconditionally for it to be ready " << std::endl;
   std::future<bool> f = cardManager->onSubsystemReady();
   subSystemsStatus = f.get();
}
```

4. Exit the application, if SDK is unable to initialize SIM profile manager and card manager subsystem

```
if(subSystemsStatus) {
   std::cout << " *** SIM profile manager subsystem and Card manager ready *** " << std::endl;
} else {
   std::cout << " *** ERROR - Unable to initialize SIM profile manager/Card manager subsystem"
             << std::endl;
}
```

5. Instantiate and register RspListener

```
std::shared_ptr<ISimProfileListener> listener = std::make_shared<RspListener>();
simProfileManager.registerListener(listener);
```

5.1 Implementation of ISimProfileListener interface for receiving Remote SIM provisioning notifications

```
class RspListener : public telux::tel::ISimProfileListener {
public:
    void onDownloadStatus(SlotId slotId, telux::tel::DownloadStatus status,
        telux::tel::DownloadErrorCause cause) override {
         // Profile download and installation status when add profile operation performed.
    }
    void onUserDisplayInfo(SlotId slotId, bool userConsentRequired,
        telux::tel::PolicyRuleMask mask) override  {
         // Based on profile policy rules received client can decide to provide user consent
         // for download and installation of profile by calling
         // ISimProfileManager::provideUserConsent
    }
    void onConfirmationCodeRequired(SlotId slotId, std::string profileName) override {
        // Provide confirmation code for the download and installation of profile by calling
        // ISimProfileManager::provideConfirmationCode
    }
```

6. Request EID of the eUICC

```
auto respCb = [&](std::string eid, telux::common::ErrorCode errorCode)
   { eidCallback(eid, errorCode); };

// Implement a callback function to get response for the EID request
void eidCallback(std::string eid, telux::common::ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "requestEid failed with error" << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "requestEid succeeded." << std::endl;
}
// Request EID of the eUICC.
auto card = cardManager->getCard(SlotId::DEFAULT_SLOT_ID, &status);
status = card->requestEid(respCb);
```

7. Add profile on the eUICC

```
auto respCb = [&](telux::common::ErrorCode errorCode) { addProfileCallback(errorCode); };

// Implement a callback function to get response for the add profile request
void addProfileCallback(telux::common::ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "addProfile failed with error" << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "addProfile succeeded." << std::endl;
}
// Add/Download profile on the eUICC.
status = simProfileManager->addProfile(SlotId::DEFAULT_SLOT_ID, activationCode,
    confirmationCode, isUserConsentSupported, respCb);
```

7.1 If user consent is required for downloading the profile, the registered listener of client will be notified by invoking onUserDisplayInfo API.

Client is expected to invoke ISimProfileManager::provideUserConsent API in order to proceed further for downloading the profile.

```
void onUserDisplayInfo(SlotId slotId, bool userConsentRequired,
    telux::tel::PolicyRuleMask mask) override  {
    // Based on user info received client can decide to provide user consent for download and
    // installation of profile by calling ISimProfileManager::provideUserConsent
}
```

7.2 If confirmation code is required for downloading the profile, the registered listener of client will be notified by invoking onConfirmationCodeRequired API.

Client is expected to invoke ISimProfileManager::provideConfirmationCode API in order to proceed further for downloading the profile.

```
void onConfirmationCodeRequired(SlotId slotId, std::string profileName) override {
    // Provide confirmation code for the download and installation of profile by calling
    // ISimProfileManager::provideConfirmationCode
}
```

7.3 When the download of profile completes or fails, the client is notified about download status.

```
void onDownloadStatus(SlotId slotId, telux::tel::DownloadStatus status,
    telux::tel::DownloadErrorCause cause) override {
    // Profile download and installation status is recieved here whenever add profile operation
    // is performed.
}
```

8. Delete profile on the eUICC

```
auto respCb = [&](telux::common::ErrorCode errorCode) { deleteProfileCallback(errorCode); };

// Implement a callback function to get response for the delete profile request
void deleteProfileCallback(telux::common::ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "deleteProfile failed with error" << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "deleteProfile succeeded." << std::endl;
}
// Delete profile on the eUICC.
status = simProfileManager->deleteProfile(SlotId::DEFAULT_SLOT_ID, profileId,
    respCb);
```

9. Request profile list on the eUICC

```
auto respCb = [&](const std::vector<std::shared_ptr<telux::tel::SimProfile>> &profiles,
    telux::common::ErrorCode errorCode) { profileListCallback(profiles, errorCode); };

// Implement a callback function to get response for the request profile list
void profileListCallback(const std::vector<std::shared_ptr<telux::tel::SimProfile>> &profiles,
    telux::common::ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "profileList failed with error" << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "profileList succeeded." << std::endl;
}
// Get profile list on the eUICC.
status = simProfileManager->requestProfileList(SlotId::DEFAULT_SLOT_ID, respCb);
```

10. Enable/disable profile on the eUICC

```
auto respCb = [&](telux::common::ErrorCode errorCode) { setProfileCallback(errorCode); };

// Implement a callback function to get response for the set profile request
void setProfileCallback(telux::common::ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "setProfile failed with error" << static_cast<int>(error) << std::endl;
        return;
    }
    std::cout << "setProfile succeeded." << std::endl;
}
// Enable/disable profile on the eUICC.
```

```
status = simProfileManager->setProfile(SlotId::DEFAULT_SLOT_ID, profileId, enable,
    respCb);
```

11. Update Nickname of the profile

```
auto respCb = [&](telux::common::ErrorCode errorCode) { updateNicknameCallback(errorCode); };

// Implement a callback function to get response for update nickname request
void updateNicknameCallback(telux::common::ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "updateNickname failed with error" << static_cast<int>(error) <<
            std::endl;
        return;
    }
    std::cout << "updateNickname succeeded." << std::endl;
}
// Update Nickname of the profile
status = simProfileManager->updateNickName(SlotId::DEFAULT_SLOT_ID, profileId, nickname,
    respCb);
```

12. Set SMDP+ server address on the eUICC

```
auto respCb = [&](telux::common::ErrorCode errorCode) { setServerAddressCallback(errorCode); };

// Implement a callback function to get response for set server addresss request
void setServerAddressCallback(telux::common::ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "setServerAddress failed with error" << static_cast<int>(error) <<
            std::endl;
        return;
    }
    std::cout << "setServerAddress succeeded." << std::endl;
}
// Set SMDP server address on the eUICC
status = simProfileManager->setServerAddress(SlotId::DEFAULT_SLOT_ID, smdpAddress,
    respCb);
```

13. Get SMDP+ and SMDS server address from the eUICC

```
auto respCb = [&](std::string smdpAddress,
    std::string smdsAddress, telux::common::ErrorCode errorCode) {
        requestServerAddressCallback(smdpAddress, smdsAddress, errorCode); };

// Implement a callback function to get response for the get server addresss request
void requestServerAddressCallback(std::string smdpAddress,
    std::string smdsAddress, telux::common::ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "requestServerAddress failed with error" << static_cast<int>(error) <<
            std::endl;
        return;
    }
    std::cout << "requestServerAddress succeeded." << std::endl;
}
// Get SMDP+ and SMDS server address on the eUICC
status = simProfileManager->requestServerAddress(SlotId::DEFAULT_SLOT_ID,
    respCb);
```

14. Memory reset on the eUICC

```
auto respCb = [&](telux::common::ErrorCode errorCode) { memoryResetCallback(errorCode); };

// Implement a callback function to get response for the memory reset request
void memoryResetCallback(telux::common::ErrorCode error) {
    if (error != ErrorCode::SUCCESS) {
        std::cout << "memoryReset failed with error" << static_cast<int>(error) << std::endl;
        return;
    }
```

```
    std::cout << "memoryReset succeeded." << std::endl;
}
// Memory reset on the eUICC
status = simProfileManager->memoryReset(SlotId::DEFAULT_SLOT_ID, resetmask,
    respCb);
```

5.9.14.2 Using Remote SIM Provisioning Reference Service

Using Remote SIM Provisioning Reference Service

SIM profile management operations such as add profile, delete profile, enable/disable profile requires HTTP interaction with the cloud to synchronize the profile/s on the eUICC with the SMDP+/SMDS server. When modem LPA/eUICC needs to reach SMDP+/SMDS server on the cloud for HTTP transaction, the HTTP request is sent to Remote SIM provisioning service i.e RSP HTTP daemon running on Application Processor. The RSP HTTP Daemon performs these HTTP transactions on behalf of modem with SMDP+/SMDS server running on the cloud. The HTTP response from cloud is sent back to modem LPA/eUICC to take appropriate action. Make sure there is internet connectivity available for performing HTTP transaction either using WLAN, WWAN or ethernet.

This section describes how to use the provided Remote SIM provisioning reference service – rsp_httpd

1. Run the Remote SIM HTTP Daemon on the device

```
1 rsp_httpd
```

Use -h option to check for all the options supported and usage instructions.

5.10 data

The List of sample apps related to data:

- Start/Stop data call
- Get data profile
- Get and Set data filter mode
- Add data filter
- Remove data filter mode
- Enable/Disable Firewall
- Create Firewall DMZ
- Add Firewall Entry
- Add a software bridge and Enable software bridge management
- Remove a software bridge and Disable software bridge management
- Create Vlan And Bind It To PDN
- Create Static NAT Entry
- Enable L2TP and Add Tunnel
- Enable/Disable Socks

- Get Dedicated Radio Bearer Status and Indication
- Get Service Status and Indication
- Get Roaming Status and Indication

5.10.1 Start/Stop data call

Cellular Data Call - Start/Stop

Please follow below steps to start or stop cellular data call

1. Implement initialization callback and get the DataFactory instance

Optionally initialization callback can be provided with get manager instance. Data factory will call callback when manager initialization is complete.

```
auto initCb = [&](telux::common::ServiceStatus status) {
   std::lock_guard<std::mutex> lock(mtx);
   status_ = status;
   initCv.notify_all();
};
auto &dataFactory = DataFactory::getInstance();
```

2. Get the DataConnectionManager instances

```
std::unique_lock<std::mutex> lck(mtx);
dataConnMgr = dataFactory.getDataConnectionManager(slotId, initCb);
```

3. Wait for DataConnectionManager initialization to be complete

```
initCv.wait(lck);
```

3.1 Check data connection manager initialization state

If DataConnectionManager initialization failed, new initialization attempt can be accomplished by calling step 2. If DataConnectionManager initialization succeed, proceed to step 4

```
if (status_ == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   // Go to step 4
}
else {
   //Go to step 2 for another initialization attempt
}
```

4. Implement DataCallResponseCb callback for startDatacall

```
void startDataCallResponseCallBack(const std::shared_ptr<telux::data::IDataCall> &dataCall,
                                   telux::common::ErrorCode error) {
   std::cout<< "Received callback for startDataCall" << std::endl;
   if(error == telux::common::ErrorCode::SUCCESS) {
      std::cout<< "Request sent successfully" << std::endl;
   } else {
      std::cout<< "Request failed with errorCode: " << static_cast<int>(error) << std::endl;
   }
}
```

5. Send a start data call request with profile ID, IpFamily type along with required callback function

```
dataConnectionManager->startDataCall(profileId, telux::data::IpFamilyType::IPV4V6,
                                     startDataCallResponseCallBack);
```

6. Response callback will be called for the startDataCall response

7. Implement DataCallResponseCb callback for stopDatacall

```
void stopDataCallResponseCallBack(const std::shared_ptr<telux::data::IDataCall> &dataCall,
                                  telux::common::ErrorCode error) {
std::cout << "Received callback for stopDataCall" << std::endl;
if(error == telux::common::ErrorCode::SUCCESS) {
   std::cout << "Request sent successfully" << std::endl;
} else {
   std::cout << "Request failed with errorCode: " << static_cast<int>(error) << std::endl;
}
```

8. Send a stop data call request with profile ID, IpFamily type along with required callback function

```
dataConnectionManager->stopDataCall(profileId, telux::data::IpFamilyType::IPV4V6,
                                    stopDataCallResponseCallBack);
```

9. Response callback will be called for the stopDataCall response

5.10.2 Get data profile

How to get data profile list

Please follow below steps to request list of available modem profiles

1. Implement initialization callback and get the DataFactory instance

Optionally initialization callback can be provided with get manager instance. Data factory will call callback when manager initialization is complete.

```
auto initCb = [&](telux::common::ServiceStatus status) {
   std::lock_guard<std::mutex> lock(mtx);
   status_ = status;
   initCv.notify_all();
};
auto &dataFactory = DataFactory::getInstance();
```

2. Get DataProfileManager instances

std::unique_lock<std::mutex> lck(mtx); auto dataProfileMgr = dataFactory.getDataProfileManager(slotId, initCb);

3. Wait for DataProfileManager initialization to be complete

initCv.wait(lck);

3.1 Check DataProfileManager initialization state

If DataProfileManager initialization failed, new initialization attempt can be accomplished by calling step 2. If initialization succeed, proceed to step 4

```
if (status == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
```

```
   // Go to step 4
}
else {
   // Go to step 2 for another initialization attempt
}
```

4. Instantiate requestProfileList callback

```
auto dataProfileListCb_ = std::make_shared<DataProfileListCallback>();
```

4.1 Implement IDataProfileListCallback interface to know status of requestProfileList

```
class DataProfileListCallback : public telux::common::IDataProfileListCallback {

  virtual void onProfileListResponse(const std::vector<std::shared_ptr<DataProfile>> &profiles,
                                     telux::common::ErrorCode error) override {

   std::cout<<"Length of available profiles are "<<profiles.size()<<std::endl;
  }
};
```

5. Send a requestProfileList along with required callback function

```
telux::common::Status status = dataProfileMgr->requestProfileList(dataProfileListCb_);
```

6. Receive DataProfileListCallback responses for requestProfileList request

5.10.3 Get and Set data filter mode

How to get and set data filter mode

Please follow below steps to get and set data filter mode

1. Get the DataFactory, DataConnectionManager and DataFilterManager instances

```
auto &dataFactory = telux::data::DataFactory::getInstance();
auto dataConnMgr_ = dataFactory.getDataConnectionManager();
auto dataFilterMgr_ = dataFactory.getDataFilterManager();
```

2. Wait for the Data Connection Manager and Data Filter Manager sub system initialization

```
bool dataConnectionSubSystemStatus = dataConnMgr_->isSubsystemReady();
if (!dataConnectionSubSystemStatus) {
    std::cout << "Data Connection Manager subsystem is not ready, Please wait" << std::endl;
    std::future<bool> f = dataConnMgr_->onSubsystemReady();
    // Wait unconditionally for data manager subsystem to be ready
    dataConnectionSubSystemStatus = f.get();
}
// Exit the application, if SDK is unable to initialize data manager subsystems
if (!dataConnectionSubSystemStatus) {
    std::cout << "ERROR - Unable to initialize subSystem" << std::endl;
    return EXIT_FAILURE;
}
bool dataFilterSubSystemStatus = dataFilterMgr_->isReady();
if (!dataFilterSubSystemStatus) {
    std::cout << "Data Filter Manager subsystem is not ready, Please wait" << std::endl;
    std::future<bool> f = dataFilterMgr_->onReady();
    // Wait unconditionally for data filter subsystem to be ready
    dataFilterSubSystemStatus = f.get();
}
// Exit the application, if SDK is unable to initialize data manager subsystems
if (!dataFilterSubSystemStatus) {
    std::cout << "ERROR - Unable to initialize subSystem" << std::endl;
    return EXIT_FAILURE;
```

```
}
```

3. Set data filter mode to enable

```
std::promise<bool> p;
int profileId = 2;
telux::data::IpFamilyType ipFamilyType = telux::data::IpFamilyType::IPV4;

telux::data::DataRestrictMode enableMode;
enableMode.filterAutoExit = telux::data::DataRestrictModeType::DISABLE;
enableMode.filterMode = telux::data::DataRestrictModeType::ENABLE;

auto status = dataFilterMgr_->setDataRestrictMode(enableMode,
         [&p](telux::common::ErrorCode error) {
        if (error == telux::common::ErrorCode::SUCCESS) {
            p.set_value(true);
        } else {
            std::cout << "Failed to set data filter mode" << std::endl;
            p.set_value(false);
        }
    }, profileId, ipFamilyType);
if(status == telux::common::Status::SUCCESS) {
    std::cout << "Set data filter mode Request sent" << std::endl;
} else {
    std::cout << "Set data filter mode Request failed" << std::endl;
}

if (p.get_future().get()) {
         std::cout << "Set data filter mode succeeded." << std::endl;
}
```

4. Get data filter mode

```
std::promise<bool> p;
std::string interfaceName = "rmnet_data0";

auto status = dataFilterMgr_->requestDataRestrictMode(interfaceName, configType_,
         [&p](telux::data::DataRestrictMode mode, telux::common::ErrorCode error) {
        if (error == telux::common::ErrorCode::SUCCESS) {
            p.set_value(true);
            if (mode.filterMode == DataRestrictModeType::DISABLE) {
                std::cout << " DataRestrictMode Disabled" << std::endl;
            } else if (mode.filterMode == DataRestrictModeType::ENABLE) {
                std::cout << " DataRestrictMode Enabled" << std::endl;
            } else {
                std::cout << " Invalid DataRestrictMode" << std::endl;
            }
        } else {
            std::cout << "Failed to get data filter mode" << std::endl;
            p.set_value(false);
        }
    });
if(status == telux::common::Status::SUCCESS) {
    std::cout << "Get data filter mode Request sent" << std::endl;
} else {
    std::cout << "Get data filter mode Request failed" << std::endl;
}

if (p.get_future().get()) {
         std::cout << "Get data filter mode succeeded." << std::endl;
}
```

5.10.4 Add data filter

How to add data filter

Please follow below steps to add data filter

1. Get the DataFactory, DataConnectionManager and DataFilterManager instances

```
auto &dataFactory = telux::data::DataFactory::getInstance();
auto dataConnMgr_ = dataFactory.getDataConnectionManager();
auto dataFilterMgr_ = dataFactory.getDataFilterManager();
```

2. Wait for the Data Connection Manager and Data Filter Manager sub system initialization

```
bool dataConnectionSubSystemStatus = dataConnMgr_->isSubsystemReady();
if (!dataConnectionSubSystemStatus) {
    std::cout << "Data Connection Manager subsystem is not ready, Please wait" << std::endl;
    std::future<bool> f = dataConnMgr_->onSubsystemReady();
    // Wait unconditionally for data manager subsystem to be ready
    dataConnectionSubSystemStatus = f.get();
}
// Exit the application, if SDK is unable to initialize data manager subsystems
if (!dataConnectionSubSystemStatus) {
    std::cout << "ERROR - Unable to initialize subSystem" << std::endl;
    return EXIT_FAILURE;
}
bool dataFilterSubSystemStatus = dataFilterMgr_->isReady();
if (!dataFilterSubSystemStatus) {
    std::cout << "Data Filter Manager subsystem is not ready, Please wait" << std::endl;
    std::future<bool> f = dataFilterMgr_->onReady();
    // Wait unconditionally for data filter subsystem to be ready
    dataFilterSubSystemStatus = f.get();
}
// Exit the application, if SDK is unable to initialize data manager subsystems
if (!dataFilterSubSystemStatus) {
    std::cout << "ERROR - Unable to initialize subSystem" << std::endl;
    return EXIT_FAILURE;
}
```

3. Set data filter mode to enable

```
std::promise<bool> p;
int profileId = 2;
telux::data::IpFamilyType ipFamilyType = telux::data::IpFamilyType::IPV4;

telux::data::DataRestrictMode enableMode;
enableMode.filterAutoExit = telux::data::DataRestrictModeType::DISABLE;
enableMode.filterMode = telux::data::DataRestrictModeType::ENABLE;

auto status = dataFilterMgr_->setDataRestrictMode(enableMode,
         [&p](telux::common::ErrorCode error) {
        if (error == telux::common::ErrorCode::SUCCESS) {
            p.set_value(true);
        } else {
            std::cout << "Failed to set data filter mode" << std::endl;
            p.set_value(false);
        }
    }, profileId, ipFamilyType);
if(status == telux::common::Status::SUCCESS) {
    std::cout << "Set data filter mode Request sent" << std::endl;
} else {
    std::cout << "Set data filter mode Request failed" << std::endl;
}

if (p.get_future().get()) {
         std::cout << "Set data filter mode succeeded." << std::endl;
}
```

4. Add data filter

```
std::promise<bool> p;
std::string ipAddr = std::string("168.128.91.1");
int port = 8888;
telux::data::IPv4Info ipv4Info_ = {};
ipv4Info_.srcAddr = ipAddr;

telux::data::PortInfo srcPort;
```

```
srcPort.port = port;
srcPort.range = 0;
telux::data::UdpInfo udpInfo_ = {};
udpInfo_.src = srcPort;

// IpProtocol for UDP
int PROTO_UDP =  17;
// create a filter of UDP type, and set source IP and port.
std::shared_ptr<telux::data::IIpFilter> dataFilter =
dataFactory.getNewIpFilter(PROTO_UDP);
dataFilter->setIPv4Info(ipv4Info_);

auto udpRestrictFilter = std::dynamic_pointer_cast<telux::data::IUdpFilter>(dataFilter);
udpRestrictFilter->setUdpInfo(udpInfo_);

auto status = dataFilterMgr_->addDataRestrictFilter(dataFilter,
         [&p](telux::common::ErrorCode error) {
        if (error == telux::common::ErrorCode::SUCCESS) {
            p.set_value(true);
        } else {
            std::cout << "Failed to add data filter" << std::endl;
            p.set_value(false);
        }
    }, profileId, ipFamilyType);
if(status == telux::common::Status::SUCCESS) {
    std::cout << "Add data filter Request sent" << std::endl;
} else {
    std::cout << "Add data filter Request failed" << std::endl;
}

if (p.get_future().get()) {
         std::cout << "Add data filter succeeded." << std::endl;
}
```

5.10.5 Remove data filter mode

How to remove data filter mode

Please follow below steps to remove all data filter mode

1. Get the DataFactory, DataConnectionManager and DataFilterManager instances

```
auto &dataFactory = telux::data::DataFactory::getInstance();
auto dataConnMgr_ = dataFactory.getDataConnectionManager();
auto dataFilterMgr_ = dataFactory.getDataFilterManager();
```

2. Wait for the Data Connection Manager and Data Filter Manager sub system initialization

```
bool dataConnectionSubSystemStatus = dataConnMgr_->isSubsystemReady();
if (!dataConnectionSubSystemStatus) {
    std::cout << "Data Connection Manager subsystem is not ready, Please wait" << std::endl;
    std::future<bool> f = dataConnMgr_->onSubsystemReady();
    // Wait unconditionally for data manager subsystem to be ready
    dataConnectionSubSystemStatus = f.get();
}
// Exit the application, if SDK is unable to initialize data manager subsystems
if (!dataConnectionSubSystemStatus) {
    std::cout << "ERROR - Unable to initialize subSystem" << std::endl;
    return EXIT_FAILURE;
}
bool dataFilterSubSystemStatus = dataFilterMgr_->isReady();
if (!dataFilterSubSystemStatus) {
    std::cout << "Data Filter Manager subsystem is not ready, Please wait" << std::endl;
    std::future<bool> f = dataFilterMgr_->onReady();
    // Wait unconditionally for data filter subsystem to be ready
    dataFilterSubSystemStatus = f.get();
}
// Exit the application, if SDK is unable to initialize data manager subsystems
```

```
if (!dataFilterSubSystemStatus) {
    std::cout << "ERROR - Unable to initialize subSystem" << std::endl;
    return EXIT_FAILURE;
}
```

3. Remove all data filter

```
std::promise<bool> p;
int profileId = 2;
telux::data::IpFamilyType ipFamilyType = telux::data::IpFamilyType::IPV4;

auto status = dataFilterMgr_->removeAllDataRestrictFilters(
         [&p](telux::common::ErrorCode error) {
        if (error == telux::common::ErrorCode::SUCCESS) {
            p.set_value(true);
        } else {
            std::cout << "Failed to remove all filter" << std::endl;
            p.set_value(false);
        }
    }, profileId, ipFamilyType);
if(status == telux::common::Status::SUCCESS) {
    std::cout << "Remove all data filter Request sent" << std::endl;
} else {
    std::cout << "Remove all data filter Request failed" << std::endl;
}

if (p.get_future().get()) {
         std::cout << "Remove all data filter succeeded." << std::endl;
}
```

5.10.6 Enable/Disable Firewall

Enable/Disable Firewall

Please follow below steps to Enable/Disable Firewall

1. Implement initialization callback and get the DataFactory instances

Optionally initialization callback can be provided with get manager instance. Data factory will call callback when manager initialization is complete.

```
auto initCb = [&](telux::common::ServiceStatus status) {
   std::lock_guard<std::mutex> lock(mtx);
   status_ = status;
   initCv.notify_all();
};
auto &dataFactory = telux::data::DataFactory::getInstance();
```

2. Get the FirewallManager instances

```
std::unique_lock<std::mutex> lck(mtx);
auto dataFwMgr  = dataFactory.getFirewallManager(opType, initCb);
```

3. Wait for FirewallManager initialization to be complete

```
initCv.wait(lck);
```

3.1 Check FirewallManager initialization state

If FirewallManager initialization failed, new initialization attempt can be accomplished by calling step 2. If FirewallManager initialization succeed, proceed to step 4

```
if (status_ == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   // Go to step 4
}
else {
   //Go to step 2 for another initialization attempt
}
```

4. Implement callback for setting firewall

```
auto respCb = [](telux::common::ErrorCode error) {
   std::cout << std::endl << std::endl;
   std::cout << "CALLBACK: "
               << "setFirewall Response"
               << (error == telux::common::ErrorCode::SUCCESS ? " is successful" : " failed");
};
```

5. set firewall mode based on profileId, enable/disable and allow/drop packets

```
dataFwMgr->setFirewall(profileId,fwEnable, allowPackets, respCb);
```

6. Response callback will be called for the setFirewall response

5.10.7 Create Firewall DMZ

Create Firewall DMZ

Please follow below steps to create firewall DMZ

1. Implement initialization callback and get the DataFactory instances

Optionally initialization callback can be provided with get manager instance. Data factory will call callback when manager initialization is complete.

```
auto initCb = [&](telux::common::ServiceStatus status) {
   std::lock_guard<std::mutex> lock(mtx);
   status_ = status;
   initCv.notify_all();
};
auto &dataFactory = telux::data::DataFactory::getInstance();
```

2. Get the FirewallManager instances

```
std::unique_lock<std::mutex> lck(mtx);
auto dataFwMgr  = dataFactory.getFirewallManager(opType, initCb);
```

3. Wait for FirewallManager initialization to be complete

```
initCv.wait(lck);
```

3.1 Check FirewallManager initialization state

If FirewallManager initialization failed, new initialization attempt can be accomplished by calling step 2. If FirewallManager initialization succeed, proceed to step 4

```
if (status_ == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   // Go to step 4
}
else {
   //Go to step 2 for another initialization attempt
}
```

4. Implement callback for create DMZ

```
auto respCb = [](telux::common::ErrorCode error) {
   std::cout << std::endl << std::endl;
   std::cout << "CALLBACK: "
             << "addDmz Response"
             << (error == telux::common::ErrorCode::SUCCESS ? " is successful" : " failed");
};
```

5. Create DMZ based on profile id and local ip address

```
dataFwMgr->enableDmz(profileId,ipAddr, respCb);
```

6. Response callback will be called for the addDmz response

5.10.8 Add Firewall Entry

Add Firewall Entry

Please follow below steps to create and add Firewall Entry

1. Implement initialization callback and get the DataFactory instances

Optionally initialization callback can be provided with get manager instance. Data factory will call callback when manager initialization is complete.

```
auto initCb = [&](telux::common::ServiceStatus status) {
   std::lock_guard<std::mutex> lock(mtx);
   status_ = status;
   initCv.notify_all();
};
auto &dataFactory = telux::data::DataFactory::getInstance();
```

2. Get the FirewallManager instances

```
std::unique_lock<std::mutex> lck(mtx);
auto dataFwMgr  = dataFactory.getFirewallManager(opType, initCb);
```

3. Wait for FirewallManager initialization to be complete

```
initCv.wait(lck);
```

3.1 Check FirewallManager initialization state

If FirewallManager initialization failed, new initialization attempt can be accomplished by calling step 2. If FirewallManager initialization succeed, proceed to step 4

```
if (status_ == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   // Go to step 4
}
else {
   //Go to step 2 for another initialization attempt
}
```

4. Get firewall Entry instance

```
std::shared_ptr<telux::data::net::IFirewallEntry> fwEntry
   = dataFactory.getNewFirewallEntry(proto, fwDir, ipFamType);
```

5. Get pointer to Ip Filter

```
std::shared_ptr<telux::data::IIpFilter> ipFilter = fwEntry->getIProtocolFilter();
```

6. Populate Ip Filter based on Ip Family type

```
switch (ipFamType) {
   case telux::data::IpFamilyType::IPV4: {
         telux::data::IPv4Info info;
         info.srcAddr = srcAddr;
         info.destAddr = destAddr;
         info.srcSubnetMask = configParser->getValue(std::string("IPV4_SRC_SUBNET_MASK"));
         info.destSubnetMask = configParser->getValue(std::string("IPV4_DEST_SUBNET_MASK"));
         info.value = (uint8_t)std::atoi(
            configParser->getValue(std::string("IPV4_SERVICE_TYPE")).c_str());
         info.mask = (uint8_t)std::atoi(
            configParser->getValue(std::string("IPV4_SERVICE_TYPE_MASK")).c_str());
         info.nextProtoId = proto;
         ipFilter->setIPv4Info(info);
   } break;
   case telux::data::IpFamilyType::IPV6: {
         telux::data::IPv6Info info;
         info.srcAddr = srcAddr;
         info.destAddr = destAddr;
         info.nextProtoId = proto;
         info.val = (uint8_t)std::atoi(
            configParser->getValue(std::string("IPV6_TRAFFIC_CLASS")).c_str());
         info.mask = (uint8_t)std::atoi(
            configParser->getValue(std::string("IPV6_TRAFFIC_CLASS_MASK")).c_str());
         info.flowLabel = (uint32_t)std::atoi(
            configParser->getValue(std::string("IPV6_FLOW_LABEL")).c_str());
         ipFilter->setIPv6Info(info);
   } break;
   default: {
      std::cout <<"Error: Unrecognized Ip Family used .. exiting app" <<std::endl;
      return 1;
   } break;
}
```

7. Populate Protocol information

```
switch (proto) {
   case 6: {   // TCP
      telux::data::TcpInfo tcpInfo;
      tcpInfo.src.port = (uint16_t)protSrcPort;
      tcpInfo.src.range = (uint16_t)protSrcRange;
      tcpInfo.dest.port = (uint16_t)protDestPort;
      tcpInfo.dest.range = (uint16_t)protDestRange;
      auto tcpFilter = std::dynamic_pointer_cast<telux::data::ITcpFilter>(ipFilter);
```

```
      if(tcpFilter) {
            tcpFilter->setTcpInfo(tcpInfo);
      }
   } break;
   case 17: {  //UDP
      telux::data::UdpInfo info;
      info.src.port = (uint16_t)protSrcPort;
      info.src.range = (uint16_t)protSrcRange;
      info.dest.port = (uint16_t)protDestPort;
      info.dest.range = (uint16_t)protDestRange;
      auto udpFilter = std::dynamic_pointer_cast<telux::data::IUdpFilter>(ipFilter);
      if(udpFilter) {
            udpFilter->setUdpInfo(info);
      }
   } break;
   default: {
   } break;
}
```

8. Instantiate add firewall entry callback instance - this is optional

```
auto respCb = [](telux::common::ErrorCode error) {
   std::cout << std::endl << std::endl;
   std::cout << "CALLBACK: "
            << "addFirewallEntry Response"
            << (error == telux::common::ErrorCode::SUCCESS ? " is successful" : " failed")
            << ". ErrorCode: " << static_cast<int>(error) << std::endl;
            promise.set_value(1);
};

std::future<int> future = promise.get_future();
dataFwMgr->addFirewallEntry(profileId, fwEntry, respCb);
```

5.10.9 Add a software bridge and Enable software bridge management

Add a software bridge and Enable software bridge management

Please follow below steps to add a software bridge and enable software bridge management

1. Implement initialization callback Get the DataFactory instances

Optionally initialization callback can be provided with get manager instance. Data factory will call callback when manager initialization is complete.

```
auto initCb = [&](telux::common::ServiceStatus status) {
   std::lock_guard<std::mutex> lock(mtx);
   status_ = status;
   initCv.notify_all();
};
 auto &dataFactory = telux::data::DataFactory::getInstance();
```

2. Get the BridgeManager instances

std::unique_lock<std::mutex> lck(mtx); auto dataBridgeMgr = dataFactory.getBridgeManager(initCb);

3. Wait for BridgeManager initialization to be complete

initCv.wait(lck);

3.1 Check BridgeManager initialization state

BridgeManager needs to be ready to remove a software bridge and disable the software bridge management in the system. If BridgeManager initialization failed, new initialization attempt can be accomplished by calling step 2. If BridgeManager initialization succeed, proceed to step 4

```
if (status_ == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   // Go to step 4
}
else {
   //Go to step 2 for another initialization attempt
}
```

4. Implement callbacks for adding a software bridge and enabling the software bridge management

```
auto respCbAdd = [](telux::common::ErrorCode error) {
    std::cout << "CALLBACK: "
              << "Add software bridge request"
              << (error == telux::common::ErrorCode::SUCCESS ? " is successful" : " failed")
              << ". ErrorCode: " << static_cast<int>(error)
              << ", description: " << Utils::getErrorCodeAsString(error) << std::endl;
};
auto respCbEnable = [](telux::common::ErrorCode error) {
    std::cout << "CALLBACK: "
              << "Enable software bridge management request is"
              << (error == telux::common::ErrorCode::SUCCESS ? " successful" : " failed")
              << ". ErrorCode: " << static_cast<int>(error)
              << ", description: " << Utils::getErrorCodeAsString(error) << std::endl;
};
```

5. Add software bridge based on the interface name, interface type and bandwidth required

```
telux::data::net::BridgeInfo config;
config.ifaceName = infName;
config.ifaceType = infType;
config.bandwidth = bridgeBw;
dataBridgeMgr->addBridge(config, respCbAdd);
```

6. Response callback will be invoked with the addBridge response

7. Enable the software bridge management if not enabled already

```
bool enable = true;
dataBridgeMgr->enableBridge(enable, respCbEnable);
```

8. Response callback will be invoked with the enableBridge response

5.10.10 Remove a software bridge and Disable software bridge management

Remove a software bridge and Disable software bridge management

Please follow below steps to remove a software bridge and and disable software bridge management

1. Implement initialization callback Get the DataFactory instances

Optionally initialization callback can be provided with get manager instance. Data factory will call callback when manager initialization is complete.

```
auto initCb = [&](telux::common::ServiceStatus status) {
   std::lock_guard<std::mutex> lock(mtx);
   status_ = status;
   initCv.notify_all();
};
 auto &dataFactory = telux::data::DataFactory::getInstance();
```

2. Get the BridgeManager instances

std::unique_lock<std::mutex> lck(mtx); auto dataBridgeMgr = dataFactory.getBridgeManager(initCb);

3. Wait for BridgeManager initialization to be complete

initCv.wait(lck);

3.1 Check BridgeManager initialization state

BridgeManager needs to be ready to remove a software bridge and disable the software bridge management in the system. If BridgeManager initialization failed, new initialization attempt can be accomplished by calling step 2. If BridgeManager initialization succeed, proceed to step 4

```
if (status_ == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   // Go to step 4
}
else {
   //Go to step 2 for another initialization attempt
}
```

4. Implement callbacks to get, remove software bridge and disable the software bridge management

```
auto respCbGet = [](const std::vector<BridgeInfo> &configs, telux::common::ErrorCode error) {
    std::cout << "CALLBACK: "
              << "Get software bridge info request"
              << (error == telux::common::ErrorCode::SUCCESS ? " is successful" : " failed")
              << ". ErrorCode: " << static_cast<int>(error)
              << ", description: " << Utils::getErrorCodeAsString(error) << std::endl;
    for (auto c : configs) {
        std::cout << "Iface name: " << c.ifaceName << ", ifaceType: " << (int)c.ifaceType
                  << ", bandwidth: " << c.bandwidth << std::endl;
    }
};
auto respCbRemove = [](telux::common::ErrorCode error) {
    std::cout << "CALLBACK: "
              << "Remove software bridge request"
              << (error == telux::common::ErrorCode::SUCCESS ? " is successful" : " failed")
              << ". ErrorCode: " << static_cast<int>(error)
              << ", description: " << Utils::getErrorCodeAsString(error) << std::endl;
};
auto respCbDisable = [](telux::common::ErrorCode error) {
    std::cout << "CALLBACK: "
              << "Disable software bridge management request is"
              << (error == telux::common::ErrorCode::SUCCESS ? " successful" : " failed")
              << ". ErrorCode: " << static_cast<int>(error)
              << ", description: " << Utils::getErrorCodeAsString(error) << std::endl;
};
```

5. Get the list of software bridges configured in the system

```
dataBridgeMgr->requestBridgeInfo(respCbGet);
```

6. Response callback will be invoked with the list of software bridges

7. Remove the software bridge from the configured bridges, based on the interface name

```
dataBridgeMgr->removeBridge(ifaceName, respCbRemove);
```

8. Response callback will be invoked with the removeBridge response

9. Disable the software bridge management if required

```
bool enable = false;
dataBridgeMgr->enableBridge(enable, respCbDisable);
```

10. Response callback will be invoked with the enableBridge response

5.10.11 Create Vlan And Bind It To PDN

Create Vlan And Bind It To PDN

Please follow below steps to create Vlan and bind it to PDN

1. Implement initialization callback and get the DataFactory instance

Optionally initialization callback can be provided with get manager instance. Data factory will call callback when manager initialization is complete.

```
auto initCb = [&](telux::common::ServiceStatus status) {
   std::lock_guard<std::mutex> lock(mtx);
   status_ = status;
   initCv.notify_all();
};
 auto &dataFactory = telux::data::DataFactory::getInstance();
```

2. Get the VlanManager instances

```
std::unique_lock<std::mutex> lck(mtx);
auto dataVlanMgr  = dataFactory.getVlanManager(opType, initCb);
```

3. Wait for VlanManager initialization to be complete

```
initCv.wait(lck);
```

3.1 Check VlanManager initialization state

If VlanManager initialization failed, new initialization attempt can be accomplished by calling step 2. If VlanManager initialization succeed, proceed to step 4

```
if (status_ == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   // Go to step 4
}
else {
   //Go to step 2 for another initialization attempt
}
```

4. Implement callback for create Vlan

```
auto respCbCreate = [](bool isAccelerated, telux::common::ErrorCode error) {
   std::cout << std::endl << std::endl;
   std::cout << "CALLBACK: "
             << "createVlan Response"
             << (error == telux::common::ErrorCode::SUCCESS ? " is successful" : " failed")
             << ". ErrorCode: " << static_cast<int>(error);
   std::cout << " Acceleration " << (isAccelerated ? "is allowed" : "is not allowed") << "\n";
};
```

5. Create Vlan based on interface type, acceleration, and assigned id

```
telux::data::VlanConfig config;
config.iface = infType;
config.vlanId = vlanId;
config.isAccelerated = isAccelerated;
dataVlanMgr->createVlan(config, respCbCreate);
```

6. Response callback will be called for the createVlan response

7. Implement callback for bindWithprofile reponse

```
auto respCbBind = [](telux::common::ErrorCode error) {
   std::cout << std::endl << std::endl;
   std::cout << "CALLBACK: "
             << "bindWithProfile Response"
             << (error == telux::common::ErrorCode::SUCCESS ? " is successful" : " failed")
             << ". ErrorCode: " << static_cast<int>(error) << std::endl;
};
```

8. Bind created Vlan with user provided profile id

```
dataVlanMgr->bindWithProfile(profileId, vlanId, respCbBind);
```

9. Response callback will be called for the bindWithProfile response

5.10.12 Create Static NAT Entry

Create Static NAT Entry

Please follow below steps to create static NAT entry

1. Implement initialization callback and get the DataFactory instance

Optionally initialization callback can be provided with get manager instance. Data factory will call callback when manager initialization is complete.

```
auto initCb = [&](telux::common::ServiceStatus status) {
   std::lock_guard<std::mutex> lock(mtx);
   status_ = status;
   initCv.notify_all();
};
 auto &dataFactory = telux::data::DataFactory::getInstance();
```

2. Get the NatManager instances

```
std::unique_lock<std::mutex> lck(mtx);
auto dataSnatMgr  = dataFactory.getNatManager(opType);
```

3. Wait for NatManager initialization to be complete

```
initCv.wait(lck);
```

3.1 Check NatManager initialization state

If NatManager initialization failed, new initialization attempt can be accomplished by calling step 2. If NatManager initialization succeed, proceed to step 4

```
if (status_ == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   // Go to step 4
}
else {
   //Go to step 2 for another initialization attempt
}
```

4. Implement callback for create Snat entry

```
auto respCb = [](telux::common::ErrorCode error) {
   std::cout << std::endl << std::endl;
   std::cout << "CALLBACK: "
             << "addStaticNatEntry"
             << (error == telux::common::ErrorCode::SUCCESS ? " is successful" : " failed");
};
```

5. Create Snat entry based on profile id, local ip, local port, global port, and protocol

```
natConfig.addr = ipAddr;
natConfig.port = (uint16_t)localIpPort;
natConfig.globalPort = (uint16_t)globalIpPort;
natConfig.proto = (uint8_t)proto;
dataSnatMgr->addStaticNatEntry(profileId, natConfig, respCb);
```

6. Response callback will be called for the addStaticNatEntry response

5.10.13 Enable L2TP and Add Tunnel

Enable L2TP and Add Tunnel

Please follow below steps to enable L2TP and Tunnel

1. Implement initialization callback and get get the DataFactory instance

Optionally initialization callback can be provided with get manager instance. Data factory will call callback when manager initialization is complete.

```
auto initCb = [&](telux::common::ServiceStatus status) {
   std::lock_guard<std::mutex> lock(mtx);
   status_ = status;
   initCv.notify_all();
};
auto &dataFactory = telux::data::DataFactory::getInstance();
```

2. Get the L2tpManager instances

```
std::unique_lock<std::mutex> lck(mtx);
auto dataL2tpMgr  = dataFactory.getL2tpManager(initCb);
```

3. Wait for L2tpManager initialization to be complete

```
initCv.wait(lck);
```

3.1 Check L2tpManager initialization state

If L2tpManager initialization failed, new initialization attempt can be accomplished by calling step 2. If L2tpManager initialization succeed, proceed to step 4

```
if (status_ == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   // Go to step 4
}
else {
   //Go to step 2 for another initialization attempt
}
```

4. Instantiate setConfig callback instance - this is optional

```
auto setConfigCb = [&setConfigPass, &promise](telux::common::ErrorCode error) {
std::cout << std::endl << std::endl;
std::cout << "CALLBACK: "
          << "setConfig Response"
          << (error == telux::common::ErrorCode::SUCCESS ? " is successful" : " failed");
};
```

5. Set L2TP Configuration

```
bool enable = true;        //Enable L2TP
bool enableMss = true;     // Enable MSS Clamping
bool enableMtu = true;     // Enable custom size MTU
int mtuSize = 0;           // Set MTU size to default 1422 bytes, otherwise set desired mtu size
dataL2tpMgr->setConfig(enable, enableMss, enableMtu, setConfigCb, mtuSize);
```

6. Configure L2TP Tunnel and Session

```
std::cout << "L2TP Set Configuration succeeded ... Adding Tunnel" << std::endl;
telux::data::net::L2tpTunnelConfig l2tpTunnelConfig;
l2tpTunnelConfig.locIface = "eth0.1"; //Set interface name to eth0.x where x is vlan id
l2tpTunnelConfig.prot = static_cast<telux::data::net::L2tpProtocol>(2); //Set protocol to UDP
l2tpTunnelConfig.locId = 1;  //Set local tunnel id
l2tpTunnelConfig.peerId = 1; //Set peer tunnel id
l2tpTunnelConfig.localUdpPort = 500;   //Set local UDP port if UDP protocol is selected above
l2tpTunnelConfig.peerUdpPort = 100;    //Set peer UDP port if UDP protocol is selected above
l2tpTunnelConfig.ipType =  static_cast<telux::data::IpFamilyType>(6); //Set Ip family type
l2tpTunnelConfig.peerIpv6Addr =  "fe80::b044::c0ff::fec4";  // Set peer Ip address
telux::data::net::L2tpSessionConfig l2tpSessionConfig;
l2tpSessionConfig.locId = 1;   //Set local session id
l2tpSessionConfig.peerId = 1;  //Set peer session id
l2tpTunnelConfig.sessionConfig.emplace_back(l2tpSessionConfig); // Add session to tunnel config
```

7. Instantiate addTunnel callback instance - this is optional

```
auto addTunnelCb = [&setConfigPass, &promise](telux::common::ErrorCode error) {
   std::cout << std::endl << std::endl;
   std::cout << "CALLBACK: "
             << "addTunnel Response"
             << (error == telux::common::ErrorCode::SUCCESS ? " is successful" : " failed")
             << ". ErrorCode: " << static_cast<int>(error) << "\n";
};
```

8. addTunnel to L2TP

```
dataL2tpMgr->addTunnel(l2tpTunnelConfig, addTunnelCb);
```

5.10.14 Enable/Disable Socks

Enable/Disable Socks

Please follow below steps to Enable/Disable Socks

1.Implement initialization callback and get the DataFactory instance

Optionally initialization callback can be provided with get manager instance. Data factory will call callback when manager initialization is complete.

```
auto initCb = [&](telux::common::ServiceStatus status) {
   std::lock_guard<std::mutex> lock(mtx);
   status_ = status;
   initCv.notify_all();
};
auto &dataFactory = telux::data::DataFactory::getInstance();
```

2. Get the SocksManager instances

std::unique_lock<std::mutex> lck(mtx); auto dataSocksMgr = dataFactory.getSocksManager(opType, initCb);

3. Wait for DataConnectionManager initialization to be complete

```
initCv.wait(lck);
```

3.1 Check SocksManager initialization state

If SocksManager initialization failed, new initialization attempt can be accomplished by calling step 2. If SocksManager initialization succeed, proceed to step 4

```
if (status_ == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   // Go to step 4
}
else {
   //Go to step 2 for another initialization attempt
}
```

4. Instantiate enable Socks callback instance - this is optional

```
auto respCb = [](telux::common::ErrorCode error) {
   std::cout << std::endl << std::endl;
   std::cout << "CALLBACK: "
             << "enableSocks Response"
             << (error == telux::common::ErrorCode::SUCCESS ? " is successful" : " failed")
             << ". ErrorCode: " << static_cast<int>(error) << "\n";
   promise.set_value(1);
};
```

5. enable/disable Socks

```
dataSocksMgr->enableSocks(enable, respCb);
```

6. Response callback will be called for the setFirewall response

5.10.15 Get Dedicated Radio Bearer Status and Indication

How to get dedicated radio bearer status and indications

Please follow below steps to get dedicated radio bearer status and indication

1. Implement IServingSystemListener listener class

```
class ServingSystemListener : public telux::data::IServingSystemListener {
public:
ServingSystemListener(SlotId slotId) : slotId_(slotId) {}
void onDrbStatusChanged(telux::data::DrbStatus status) override {
    std::cout << "\n onDrbStatusChanged on SlotId: "
                << static_cast<int>(slotId_) << std::endl;
    switch(status) {
        case telux::data::DrbStatus::ACTIVE:
            std::cout << "Current Drb Status is Active" << std::endl;
            break;
        case telux::data::DrbStatus::DORMANT:
            std::cout << "Current Drb Status is Dormant" << std::endl;
            break;
        case telux::data::DrbStatus::UNKNOWN:
            std::cout << "Current Drb Status is Unknown" << std::endl;
            break;
        default:
            std::cout << "Error: Unexpected Drb Status is reported" << std::endl;
            break;
    }
}

private:
SlotId slotId_;
};
```

2. Instantiate initialization callback - this is optional

```
auto initCb = [&](telux::common::ServiceStatus status) {
    subSystemStatus = status;
    subSystemStatusUpdated = true;
    cv_.notify_all();
};
```

3. Get the DataFactory and data Serving System Manager instance

```
auto &dataFactory = telux::data::DataFactory::getInstance();
do {
    subSystemStatusUpdated = false;
    std::unique_lock<std::mutex> lck(mtx_);
    dataServingSystemMgr = dataFactory.getServingSystemManager(slotId, initCb);
```

4. Check if data Serving System manager is ready

```
    if (dataServingSystemMgr) {
    std::cout << "\n\nInitializing Data Serving System manager subsystem on slot " <<
            slotId << ", Please wait ..." << std::endl;
    cv_.wait(lck, [&]{return subSystemStatusUpdated;});
    subSystemStatus = dataServingSystemMgr->getServiceStatus();
    }
    if (subSystemStatus == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << " *** DATA Serving System is Ready *** " << std::endl;
    break;
    }
    else {
    std::cout << " *** Unable to initialize data Serving System *** " << std::endl;
    }
```

```
} while (1);
```

5. Register for Serving System listener

```
dataServingSystemMgr->registerListener(dataListener);
```

6. Get dedicated radio bearer Status

```
telux::data::DrbStatus drbStatus = dataServingSystemMgr->getDrbStatus();
switch(drbStatus) {
    case telux::data::DrbStatus::ACTIVE:
    std::cout << "Current Drb Status is Active" << std::endl;
    break;
    case telux::data::DrbStatus::DORMANT:
    std::cout << "Current Drb Status is Dormant" << std::endl;
    break;
    case telux::data::DrbStatus::UNKNOWN:
    std::cout << "Current Drb Status is Unknown" << std::endl;
    break;
    default:
    std::cout << "Error: Unexpected Drb Status is reported" << std::endl;
    break;
}
```

7. Wait for dedicated radio bearer notifications

5.10.16 Get Service Status and Indication

How to get service status and indications

Please follow below steps to request service status and indication

1. Implement IServingSystemListener listener class

```
class ServingSystemListener : public telux::data::IServingSystemListener {
public:
ServingSystemListener(SlotId slotId) : slotId_(slotId) {}
void onServiceStateChanged(telux::data::ServiceStatus status) {
    std::cout << "\n onServiceStateChanged on SlotId: " << static_cast<int>(slotId_);
    if(status.serviceState == telux::data::DataServiceState::OUT_OF_SERVICE) {
        std::cout << "Current Status is Out Of Service" << std::endl;
    } else {
        std::cout << "Current Status is In Service" << std::endl;
    }
}
private:
SlotId slotId_;
};
```

2. Instantiate initialization callback - this is optional

```
auto initCb = [&](telux::common::ServiceStatus status) {
    subSystemStatus = status;
    subSystemStatusUpdated = true;
    cv_.notify_all();
};
```

3. Get the DataFactory and data Serving System Manager instance

```
auto &dataFactory = telux::data::DataFactory::getInstance();
do {
    subSystemStatusUpdated = false;
    std::unique_lock<std::mutex> lck(mtx_);
    dataServingSystemMgr = dataFactory.getServingSystemManager(slotId, initCb);
```

4. Check if data serving system manager is ready

```
    if (dataServingSystemMgr) {
        std::cout << "\n\nInitializing Data Serving System manager subsystem on slot " <<
            slotId << ", Please wait ..." << std::endl;
        cv_.wait(lck, [&]{return subSystemStatusUpdated;});
        subSystemStatus = dataServingSystemMgr->getServiceStatus();
    }
    if (subSystemStatus == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
        std::cout << " *** DATA Serving System is Ready *** " << std::endl;
        break;
    }
    else {
        std::cout << " *** Unable to initialize data Serving System *** " << std::endl;
    }
} while (1);
```

5. Register for Serving System listener

```
dataServingSystemMgr->registerListener(dataListener);
```

6. Get current Service Status

```
// Callback
auto respCb = [&slotId](
            telux::data::ServiceStatus serviceStatus, telux::common::ErrorCode error) {
    std::cout << std::endl << std::endl;
    std::cout << "CALLBACK: "
                << "requestServiceStatus Response on slotid " << static_cast<int>(slotId);
    if(error == telux::common::ErrorCode::SUCCESS) {
        std::cout << " is successful" << std::endl;
        logServiceStatusDetails(serviceStatus);
    }
    else {
        std::cout << " failed"
                  << ". ErrorCode: " << static_cast<int>(error) << std::endl;
    }
};
dataServingSystemMgr->requestServiceStatus(respCb);
```

7. Wait for request response and notifications

5.10.17 Get Roaming Status and Indication

How to get roaming status and indications

Please follow below steps to request roaming status and indication

1. Implement IServingSystemListener listener class

```
class ServingSystemListener : public telux::data::IServingSystemListener {
public:
ServingSystemListener(SlotId slotId) : slotId_(slotId) {}
void onRoamingStatusChanged(telux::data::RoamingStatus status) {
    std::cout << "\n onRoamingStatusChanged on SlotId: "
```

```
                << static_cast<int>(slotId_) << std::endl;
    bool isRoaming = status.isRoaming;
    if(isRoaming) {
        std::cout << "System is in Roaming State" << std::endl;
    } else {
        std::cout << "System is not in Roaming State" << std::endl;
    }
}
private:
SlotId slotId_;
};
```

2. Instantiate initialization callback - this is optional

```
auto initCb = [&](telux::common::ServiceStatus status) {
    subSystemStatus = status;
    subSystemStatusUpdated = true;
    cv_.notify_all();
};
```

3. Get the DataFactory and data Serving System Manager instance

```
auto &dataFactory = telux::data::DataFactory::getInstance();
do {
    subSystemStatusUpdated = false;
    std::unique_lock<std::mutex> lck(mtx_);
    dataServingSystemMgr = dataFactory.getServingSystemManager(slotId, initCb);
```

4. Check if data Serving System manager is ready

```
    if (dataServingSystemMgr) {
        std::cout << "\n\nInitializing Data Serving System manager subsystem on slot " <<
            slotId << ", Please wait ..." << std::endl;
        cv_.wait(lck, [&]{return subSystemStatusUpdated;});
        subSystemStatus = dataServingSystemMgr->getServiceStatus();
    }
    if (subSystemStatus == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
        std::cout << " *** DATA Serving System is Ready *** " << std::endl;
        break;
    }
    else {
        std::cout << " *** Unable to initialize data Serving System *** " << std::endl;
    }
} while (1);
```

5. Register for Serving System listener

```
dataServingSystemMgr->registerListener(dataListener);
```

6. Get current Service Status

```
// Callback
auto respCb = [&slotId](
                telux::data::RoamingStatus roamingStatus, telux::common::ErrorCode error) {
   std::cout << std::endl << std::endl;
   std::cout << "CALLBACK: "
                << "requestRoamingStatus Response on slotid " << static_cast<int>(slotId);
   if(error == telux::common::ErrorCode::SUCCESS) {
      std::cout << " is successful" << std::endl;
      logRoamingStatusDetails(roamingStatus);
   }
   else {
      std::cout << " failed"
                << ". ErrorCode: " << static_cast<int>(error) << std::endl;
   }
};
```

```
dataServingSystemMgr->requestRoamingStatus(respCb);
```

7. Wait for request response and notifications

5.11 loc

The List of sample apps related to location:

- Using Location Service APIs
- Using Location Configurator APIs

5.11.1 Using Location Service APIs

Using Location Service APIs

Please follow below steps to get Location, Satellite Vehicle (SV) and Jammer Info reports

1. Implement a command response function

```
void CmdResponse(ErrorCode error) {
    if (error == ErrorCode::SUCCESS) {
        std::cout << " Command executed successfully" << std::endl;
    }
    else {
        std::cout << " Command failed\n errorCode: " << static_cast<int>(error) << std::endl;
    }
}
```

2. Implement ILocationListener interface

```
class MyLocationListener : public ILocationListener {
    public:
        void onBasicLocationUpdate(const std::shared_ptr<ILocationInfoBase> &locationInfo)
          override;
        void onDetailedLocationUpdate(const std::shared_ptr<ILocationInfoEx> &locationInfo)
          override;
        void onGnssSVInfo(const std::shared_ptr<IGnssSVInfo> &gnssSVInfo) override;
        void onGnssSignalInfo(const std::shared_ptr<IGnssSignalInfo> &gnssDatainfo) override;
};
```

3. Get the LocationFactory instance

```
auto &locationFactory = LocationFactory::getInstance();
```

4. Get LocationManager instance

```
std::promise<ServiceStatus> prom = std::promise<ServiceStatus>();
auto locationManager_ = locationFactory.getLocationManager([&](ServiceStatus status) {
    prom.set_value(status);
});
```

5. Wait for the location subsystem initialization

```
ServiceStatus managerStatus = locationManager_->getServiceStatus();
if (managerStatus != ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "Location subsystem is not ready, Please wait!!!... "<< std::endl;
    managerStatus = prom.get_future().get();
}
```

6. Exit the application, if SDK is unable to initialize location subsystems

```
if (managerStatus == ServiceStatus::SERVICE_AVAILABLE) {
    std::cout<< "Subsystem is ready" << std::endl;
```

```
} else {
    std::cout << " *** ERROR - Unable to initialize Location subsystem"<< std::endl;
    return -1;
}
```

7. Instantiate MyLocationListener

```
auto myLocationListener = std::make_shared<MyLocationListener>();
```

8. Register for Location, SV and Jammer info updates

```
locationManager_->registerListenerEx(myLocationListener);
```

9. Start Location reports with Detailed information

```
uint32_t minIntervalInput = 2000; // Default is 1000 milli seconds.
locationManager_->startDetailedReports(minIntervalInput, CmdResponse);
```

10. Command response callback is invoked with error code indicating SUCCESS or FAILURE of the operation.

11. Wait for Location fix, SV and Jammer info

```
void MyLocationListener::onDetailedLocationUpdate(
  const std::shared_ptr<telux::loc::ILocationInfoEx> &locationInfo) {
    std::cout << "New detailed Location info received" << std::endl;
}

void MyLocationListener::onGnssSVInfo(
  const std::shared_ptr<telux::loc::IGnssSVInfo> &gnssSVInfo) {
    std::cout << "Gnss Satellite Vehicle info received" << std::endl;
}

void MyLocationListener::onGnssSignalInfo(
  const std::shared_ptr<telux::loc::IGnssSignalInfo> &gnssDatainfo) {
    std::cout << "Gnss Signal info received" << std::endl;
}
```

12. Observe that changes in the configuration parameters have corresponding effect on how the Location, Satellite Vehicle (SV) and Jammer reports are received. The configuration will be same across all the location client applications and the least value of the parameter from all client applications will take effect finally.

5.11.2 Using Location Configurator APIs

Using Location Configurator APIs

Please follow below steps to use Configurator APIs

1. Implement a command response function

```
void CmdResponse(ErrorCode error) {
    if (error == ErrorCode::SUCCESS) {
        std::cout << " Command executed successfully" << std::endl;
    }
    else {
        std::cout << " Command failed\n errorCode: " << static_cast<int>(error) << std::endl;
    }
}
```

2. Get the LocationFactory instance

```
auto &locationFactory = LocationFactory::getInstance();
```

3. Get LocationConfigurator instance

```
std::promise<ServiceStatus> prom = std::promise<ServiceStatus>();
auto locConfigurator_ = locationFactory.getLocationConfigurator([&](ServiceStatus status) {
    prom.set_value(status);
});
```

4. Wait for the Location Config. initialization

```
ServiceStatus managerStatus = locConfigurator_->getServiceStatus();
 if (managerStatus != ServiceStatus::SERVICE_AVAILABLE) {
     std::cout << "Location Config. is not ready, Please wait!!!... "<< std::endl;
     managerStatus = prom.get_future().get();
 }
```

5. Exit the application, if SDK is unable to initialize Location Config.

```
if (managerStatus == ServiceStatus::SERVICE_AVAILABLE) {
     std::cout<< "Location Config. is ready" << std::endl;
 } else {
     std::cout << " *** ERROR - Unable to initialize Location Config."<< std::endl;
     return -1;
 }
```

6. Enable/Disable Constraint Tunc API

```
locConfigurator_->configureCTunc(enable, CmdResponse, optThreshold, optPower);
```

7. Command response callback is invoked with error code indicating SUCCESS or FAILURE of the operation.

5.12 config

The List of sample apps related to modem config:

- How to load and activate modem config file
- How to Get and Set Auto Config Selection Mode
- How to deactivate and delete modem config file

5.12.1 How to load and activate modem config file

How to load and activate modem config file

Please follow below steps to load and activate a modem config file

1. Get the ConfigFactory and ModemConfigManager instances

```
auto &configFactory = telux::config::ConfigFactory::getInstance();
auto modemConfigManager = configFactory.getModemConfigManager();
```

2. Wait for the Modem Config sub system initialization

```
bool subSystemStatus = modemConfigManager->isSubsystemReady();
if (!subSystemStatus) {
    std::cout << "Modem Config subsystem is not ready, Please wait" << std::endl;
    std::future<bool> f = modemConfigManager_->onSubsystemReady();
    // Wait unconditionally for modem config subsystem to be ready
    subSystemStatus = f.get();
}
// Exit the application, if SDK is unable to initialize modem config subsystems
```

```
if (!subSystemStatus) {
    std::cout << "ERROR - Unable to initialize subSystem" << std::endl;
    return EXIT_FAILURE;
}
```

3. Load a modem config file from file path

```
std::promise<bool> p;
auto status = modemConfigManager_->loadConfigFile(filePath, configType_,
         [&p](telux::common::ErrorCode error) {
        if (error == telux::common::ErrorCode::SUCCESS) {
            p.set_value(true);
        } else {
            std::cout << "Failed to load config" << std::endl;
            p.set_value(false);
        }
    });
if(status == telux::common::Status::SUCCESS) {
    std::cout << "Load config Request sent" << std::endl;
} else {
    std::cout << "Load config Request failed" << std::endl;
}

if (p.get_future().get()) {
         std::cout << "Load config succeeded." << std::endl;
}
```

4. Request Config List from modem's storage.

```
// configList_ is member variable to store config list.
std::promise<bool> p;
telux::common::Status status = modemConfigManager_->requestConfigList(
    [&p, this](std::vector<telux::config::ConfigInfo> configList,
                                    telux::common::ErrorCode errCode) {
    if (errCode == telux::common::ErrorCode::SUCCESS) {
        configList_ = configList;
        p.set_value(true);
    } else {
        std::cout << "Failed to get config list" << std::endl;
        p.set_value(false);
    }
});
if (status == telux::common::Status::SUCCESS) {
    std::cout << "Get Config List Request sent" << std::endl;
} else {
    std::cout << "Get Config List Request failed" << std::endl;
}

if (p.get_future().get()) {
    std::cout << "Config List Updated !!" << std::endl;
}
```

5. Activate modem config File.

```
// configNo denotes index of the modem config file in in config List.
ConfigId configId;
configId = configList_[configNo].id;
telux::common::Status status = modemConfigManager_->activateConfig(configType_, configId,
        slotId_, [&p](telux::common::ErrorCode error) {
        if (error == telux::common::ErrorCode::SUCCESS) {
            p.set_value(true);
        } else {
            std::cout << "Failed to activate config" << std::endl;
            p.set_value(false);
        }
    });
if (status == telux::common::Status::SUCCESS) {
    std::cout << "Activate Request sent " << std::endl;
} else {
    std::cout << "Activate Request failed" << std::endl;
```

```
}

if (p.get_future().get()) {
         std::cout << "config Activated !!" << std::endl;
}
```

5.12.2 How to Get and Set Auto Config Selection Mode

How to Get and Set Auto Config Selection Mode.

Please follow below steps to Get and Set Auto Config Selection Mode.

1. Get the ConfigFactory and ModemConfigManager instances

```
auto &configFactory = telux::config::ConfigFactory::getInstance();
auto modemConfigManager = configFactory.getModemConfigManager();
```

2. Wait for the Modem Config sub system initialization

```
bool subSystemStatus = modemConfigManager->isSubsystemReady();
if (!subSystemStatus) {
    std::cout << "Modem Config subsystem is not ready, Please wait" << std::endl;
    std::future<bool> f = modemConfigManager_->onSubsystemReady();
    // Wait unconditionally for modem config subsystem to be ready
    subSystemStatus = f.get();
}
// Exit the application, if SDK is unable to initialize modem config subsystems
if (!subSystemStatus) {
    std::cout << "ERROR - Unable to initialize subSystem" << std::endl;
    return EXIT_FAILURE;
}
```

3. Set Auto Config Selection Mode

```
std::promise<bool> p;
telux::config::AutoSelectionMode mode = telux::config::AutoSelectionMod::ENABLED;
telux::common::Status status = modemConfigManager_->setAutoSelectionMode(mode,
       slotId_, [&p](telux::common::ErrorCode error) {
        if (error == telux::common::ErrorCode::SUCCESS) {
            p.set_value(true);
        } else {
            std::cout << "Failed to set selection mode" << std::endl;
            p.set_value(false);
        }
    });
if (status == telux::common::Status::SUCCESS) {
    std::cout << "set selection mode Request sent" << std::endl;
} else {
    std::cout << "set selection mode Request failed" << std::endl;
}

if (p.get_future().get()) {
         std::cout << "set selection mode succeeded." << std::endl;
}
```

4. Get Auto Config Selection Mode

```
std::promise<bool> p;
telux::config::AutoSelectionMode selMode;
telux::common::Status status = modemConfigManager_->getAutoSelectionMode(
        [&p, &selMode](AutoSelectionMode selectionMode, telux::common::ErrorCode error) {
        if (error == telux::common::ErrorCode::SUCCESS) {
            selMode = selectionMode;
            p.set_value(true);
        } else {
            std::cout << "Failed to get selection mode" << std::endl;
```

```
            p.set_value(false);
        }
    }, slotId_);
if (status == telux::common::Status::SUCCESS) {
    std::cout << "Get selection mode Request sent" << std::endl;
} else {
    std::cout << "Get selection mode Request failed" << std::endl;
}

if (p.get_future().get()) {
    if (selMode == telux::config::AutoSelectionMode::DISABLED) {
        std::cout << "DISABLED" << std::endl;
    } else {
        std::cout << "ENABLED" << std::endl;
    }
}
```

5.12.3 How to deactivate and delete modem config file

How to deactivate and delete modem config file

Please follow below steps to deactivate and delete modem config file

1. Get the ConfigFactory and ModemConfigManager instances

```
auto &configFactory = telux::config::ConfigFactory::getInstance();
auto modemConfigManager = configFactory.getModemConfigManager();
```

2. Wait for the Modem Config sub system initialization.

```
bool subSystemStatus = modemConfigManager->isSubsystemReady();
if (!subSystemStatus) {
    std::cout << "Modem Config subsystem is not ready, Please wait" << std::endl;
    std::future<bool> f = modemConfigManager_->onSubsystemReady();
    // Wait unconditionally for modem config subsystem to be ready
    subSystemStatus = f.get();
}
// Exit the application, if SDK is unable to initialize modem config subsystems
if (!subSystemStatus) {
    std::cout << "ERROR - Unable to initialize subSystem" << std::endl;
    return EXIT_FAILURE;
}
```

3. Deactivate a modem config file.

```
std::promise<bool> p;
telux::common::Status status = modemConfigManager_->deactivateConfig(configType_,
       slotId_,[&p](telux::common::ErrorCode error) {
        if (error == telux::common::ErrorCode::SUCCESS) {
            p.set_value(true);
        } else {
            std::cout << "Failed to deactivate config" << std::endl;
            p.set_value(false);
        }
    });
if (status == telux::common::Status::SUCCESS) {
    std::cout << "Deactivate Request sent" << std::endl;
} else {
    std::cout << "Deactivate Request failed" << std::endl;
}

if (p.get_future().get()) {
         std::cout << "deactivate config succeeded." << std::endl;
}
```

4. Request Config List from modem's storage.

```
// configList_ is member variable to store config list.
```

```
std::promise<bool> p;
telux::common::Status status = modemConfigManager_->requestConfigList(
    [&p, this](std::vector<telux::config::ConfigInfo> configList,
                                    telux::common::ErrorCode errCode) {
    if (errCode == telux::common::ErrorCode::SUCCESS) {
        configList_ = configList;
        p.set_value(true);
    } else {
        std::cout << "Failed to get config list" << std::endl;
        p.set_value(false);
    }
});
if (status == telux::common::Status::SUCCESS) {
    std::cout << "Get Config List Request sent" << std::endl;
} else {
    std::cout << "Get Config List Request failed" << std::endl;
}

if (p.get_future().get()) {
    std::cout << "Config List Updated !!" << std::endl;
}
```

5. Delete a modem config file from modem's storage.

```
std::promise<bool> p;
// configNo denotes the index of config file in config list
configId = configList_[configNo].id;
telux::common::Status status = modemConfigManager_->deleteConfig(configType_,
      configId, [&p](telux::common::ErrorCode error) {
        if (error == telux::common::ErrorCode::SUCCESS) {
            p.set_value(true);
        } else {
            std::cout << "Failed to delete config" << std::endl;
            p.set_value(false);
        }
    });
if (status == telux::common::Status::SUCCESS) {
    std::cout << "Delete Request sent successfully" << std::endl;
} else {
    std::cout << "Delete Request failed" << std::endl;
}
if (p.get_future().get()) {
         std::cout << "Delete config succeeded." << std::endl;
}
```

5.13 power

The List of sample apps related to power:

- Using TCU Activity Manager APIs to get TCU activity state updates
- Using TCU Activity Manager APIs to set the TCU activity state in ACTIVE mode
- Using TCU Activity Manager APIs to set the TCU activity state in PASSIVE mode

5.13.1 Using TCU Activity Manager APIs to get TCU activity state updates

Using TCU Activity Manager APIs to get TCU activity state updates

The below steps need to be followed by applications to listen to TCU-activity state notifications, for performing any tasks before the state transition. These are valid in both ACTIVE and PASSIVE modes.

1. Implement ITcuActivityListener and IServiceStatusListener interface

```
class MyTcuActivityStateListener : public ITcuActivityListener,
                                   public IServiceStatusListener {
public:
    void onTcuActivityStateUpdate(TcuActivityState state) override;
    void onServiceStatusChange(ServiceStatus status) override;
};
```

2. Get the Power-Factory instance

```
auto &powerFactory = PowerFactory::getInstance();
```

3. Get TCU-activity manager instance with clientType as SLAVE

```
auto tcuActivityManager = powerFactory.getTcuActivityManager(ClientType::SLAVE);
```

4. Wait for the TCU-activity management services to be initialized and ready

```
bool isReady = tcuActivityManager->isReady();
if(!isReady) {
    std::cout << "TCU-activity management service is not ready" << std::endl;
    std::cout << "Waiting uncondotionally for it to be ready " << std::endl;
    std::future<bool> f = tcuActivityManager->onReady();
    isReady = f.get();
}
```

5. Exit the application, if SDK is unable to initialize TCU-activity management service

```
if(isReady) {
    std::cout << " *** TCU-activity management service is Ready *** " << std::endl;
} else {
    std::cout << " *** ERROR - Unable to initialize TCU-activity management service" << std::endl;
    return 1;
}
```

6. Instantiate MyTcuActivityStateListener

```
auto myTcuStateListener = std::make_shared<MyTcuActivityStateListener>();
```

7. Register for updates on TCU-activity state and its management service status

```
tcuActivityManager->registerListener(myTcuStateListener);
tcuActivityManager->registerServiceStateListener(myTcuStateListener);
```

8. Wait for the TCU-activity state updates

```
void MyTcuActivityStateListener::onTcuActivityStateUpdate(TcuActivityState state) {
    std::cout << std::endl << "********* TCU-activity state update *********" << std::endl;
    // Avoid long blocking calls when handling notifications
}
```

9. On SUSPEND/SHUTDOWN notification, save any required information and send one(despite multiple listeners) acknowledgement

```
tcuActivityManager->sendActivityStateAck(TcuActivityStateAck);
```

10. When the TCU-activity management service goes down, this API is invoked with status UNAVAILABLE. All TCU-activity state notifications will be stopped until the status becomes AVAILABLE again

```
void MyTcuActivityStateListener::onServiceStatusChange(ServiceStatus status) {
    std::cout << std::endl << "****** TCU-activity management service status update ******" << std::endl;
    // Avoid long blocking calls when handling notifications
}
```

5.13.2 Using TCU Activity Manager APIs to set the TCU activity state in ACTIVE mode

Using TCU Activity Manager APIs to set the TCU activity state in ACTIVE mode

The below steps need to be followed by the applications that control the TCU-activity state, to change the TCU-activity state in ACTIVE mode.

1. Implement a command response function

```
void commandResponse(ErrorCode error) {
    if (error == ErrorCode::SUCCESS) {
        std::cout << " Command executed successfully" << std::endl;
    } else {
        std::cout << " Command failed, errorCode: " << static_cast<int>(error) << std::endl;
    }
}
```

2. Implement ITcuActivityListener and IServiceStatusListener interface.

```
class MyTcuActivityStateListener : public ITcuActivityListener,
                                   public IServiceStatusListener {
public:
    void onTcuActivityStateUpdate(TcuActivityState state) override;
    void onServiceStatusChange(ServiceStatus status) override;
};
```

3. Get the Power-Factory instance

```
auto &powerFactory = PowerFactory::getInstance();
```

4. Get TCU-activity manager instance with ClientType as MASTER

```
auto tcuActivityManager = powerFactory.getTcuActivityManager(ClientType::MASTER);
```

5. Wait for the TCU-activity management services to be initialized and ready

```
bool isReady = tcuActivityManager->isReady();
if(!isReady) {
    std::cout << "TCU-activity management service is not ready" << std::endl;
    std::cout << "Waiting uncondotionally for it to be ready " << std::endl;
    std::future<bool> f = tcuActivityManager->onReady();
    isReady = f.get();
}
```

6. Exit the application, if SDK is unable to initialize TCU-activity management service

```
if(isReady) {
    std::cout << " *** TCU-activity management service is Ready *** " << std::endl;
} else {
    std::cout << " *** ERROR - Unable to initialize TCU-activity management service" << std::endl;
    return 1;
}
```

7. Instantiate MyTcuActivityStateListener

```
auto myTcuStateListener = std::make_shared<MyTcuActivityStateListener>();
```

8. Register for updates on TCU-activity state and its management service status

```
tcuActivityManager->registerListener(myTcuStateListener);
tcuActivityManager->registerServiceStateListener(myTcuStateListener);
```

9. Set the TCU-activity state

```
tcuActivityManager->setActivityState(state,&commandResponse);
```

10. Command response callback function is invoked with error code indicating whether the request was SUCCESS or FAILURE

11. This API on the listener is invoked to notify that the TCU-activity state is changing to the desired state

```
void MyTcuActivityStateListener::onTcuActivityStateUpdate(TcuActivityState state) {
    std::cout << std::endl << "********* TCU-activity state update *********" << std::endl;
    // Avoid long blocking calls when handling notifications
}
```

12. On SUSPEND/SHUTDOWN notification, save any required information and send one(despite multiple listeners) acknowledgement

```
tcuActivityManager->sendActivityStateAck(TcuActivityStateAck);
```

13. When the TCU-activity management service goes down, the below listener API is invoked with status UNAVAILABLE. Any further TCU-activity state commands and notifications will not be served until the status becomes AVAILABLE again

```
void MyTcuActivityStateListener::onServiceStatusChange(ServiceStatus status) {
    std::cout << std::endl << "****** TCU-activity management service status update ******" << std::endl;
    // Avoid long blocking calls when handling notifications
}
```

5.13.3 Using TCU Activity Manager APIs to set the TCU activity state in PASSIVE mode

Using TCU Activity Manager APIs to set the TCU activity state in PASSIVE mode

The below steps need to be followed by the applications that control the TCU-activity state, to change the TCU-activity state in PASSIVE mode.

1. Implement a command response function

```
void commandResponse(ErrorCode error) {
    if (error == ErrorCode::SUCCESS) {
        std::cout << " Command executed successfully" << std::endl;
    } else {
        std::cout << " Command failed, errorCode: " << static_cast<int>(error) << std::endl;
    }
}
```

2. Implement ITcuActivityListener and IServiceStatusListener interface.

```
class MyTcuActivityStateListener : public ITcuActivityListener,
                                   public IServiceStatusListener {
public:
    void onTcuActivityStateUpdate(TcuActivityState state) override;
```

```
    void onSlaveAckStatusUpdate(telux::common::Status status);
    void onServiceStatusChange(ServiceStatus status) override;
};
```

3. Get the Power-Factory instance

```
auto &powerFactory = PowerFactory::getInstance();
```

4. Get TCU-activity manager instance with ClientType as MASTER

```
auto tcuActivityManager = powerFactory.getTcuActivityManager(ClientType::MASTER);
```

5. Wait for the TCU-activity management services to be initialized and ready

```
bool isReady = tcuActivityManager->isReady();
if(!isReady) {
    std::cout << "TCU-activity management service is not ready" << std::endl;
    std::cout << "Waiting uncondotionally for it to be ready " << std::endl;
    std::future<bool> f = tcuActivityManager->onReady();
    isReady = f.get();
}
```

6. Exit the application, if SDK is unable to initialize TCU-activity management service

```
if(isReady) {
    std::cout << " *** TCU-activity management service is Ready *** " << std::endl;
} else {
    std::cout << " *** ERROR - Unable to initialize TCU-activity management service" << std::endl;
    return 1;
}
```

7. Instantiate MyTcuActivityStateListener

```
auto myTcuStateListener = std::make_shared<MyTcuActivityStateListener>();
```

8. Register for updates on TCU-activity state and its management service status

```
tcuActivityManager->registerListener(myTcuStateListener);
tcuActivityManager->registerServiceStateListener(myTcuStateListener);
```

9. Set the TCU-activity state

```
tcuActivityManager->setActivityState(state,&commandResponse);
```

10. Command response callback function is invoked with error code indicating whether the request was SUCCESS or FAILURE

11. The below listener API is invoked to notify that the TCU-activity state is changing to the desired state

```
void MyTcuActivityStateListener::onTcuActivityStateUpdate(TcuActivityState state) {
    std::cout << std::endl << "********* TCU-activity state update *********" << std::endl;
    // Avoid long blocking calls when handling notifications
}
```

12. On SUSPEND/SHUTDOWN notification, save any required information and send one(despite multiple listeners) acknowledgement

```
tcuActivityManager->sendActivityStateAck(TcuActivityStateAck);
```

13. The below listener API is invoked to notify the acknowledgement status of all the clients in the system

```
void MyTcuActivityStateListener::onSlaveAckStatusUpdate(telux::common::Status status) {
    if (status == telux::common::Status::SUCCESS) {
        std::cout << " All clients acknowledged successfully" << std::endl;
    } else if (status == telux::common::Status::EXPIRED) {
        std::cout << " Timeout occured while waiting for all acknowledgements" << std::endl;
    } else {
```

```
        std::cout << " Failed to receive all acknowledgements, status: " << static_cast<int>(status) <<
      std::endl;
    }
}
```

14. When the TCU-activity management service goes down, the below listener API is invoked with status UNAVAILABLE. Any further TCU-activity state commands and notifications will not be served until the status becomes AVAILABLE again

```
void MyTcuActivityStateListener::onServiceStatusChange(ServiceStatus status) {
    std::cout << std::endl << "****** TCU-activity management service status update ******" << std::endl;
    // Avoid long blocking calls when handling notifications
}
```

5.14 thermal

The List of sample apps related to thermal:

- Using Thermal Manager APIs
- Using Thermal Shutdown Manager APIs to get Thermal autoshutdown mode updates
- Using Thermal Shutdown Manager APIs to set Autoshutdown modes

5.14.1 Using Thermal Manager APIs

Using Thermal Manager APIs

Please follow below steps to get thermal zones and cooling devices

1. Get thermal factory and thermal manager instances

```
auto &thermalFactory = telux::therm::ThermalFactory::getInstance();
auto thermalMgr = thermalFactory.getThermalManager();
```

2. Send get thermal zones request using thermal manager object

```
if(thermalMgr != nullptr) {
   std::vector<std::shared_ptr<telux::therm::IThermalZone>> zoneInfo
      = thermalMgr->getThermalZones();
   if(zoneInfo.size() > 0) {
      for(auto index = 0; index < zoneInfo.size(); index++) {
         std::cout << "Thermal zone Id: " << zoneInfo(index)->getId() << "Description: "
                   << zoneInfo(index)->getDescription() << "Current temp: "
                   << zoneInfo(index)->getCurrentTemp() << std::endl;
         std::cout << std::endl;
      }
   } else {
      std::cout << "No thermal zones found!" << std::endl;
   }
} else {
   std::cout << "Thermal manager is nullptr" << std::endl;
}
```

3. Send get cooling devices request using thermal manager object

```
if(thermalMgr != nullptr) {
   std::vector<std::shared_ptr<telux::therm::ICoolingDevice>> coolingDevice
      = thermalMgr->getCoolingDevices();
   if(coolingDevice.size() > 0) {
      for(auto index = 0; index < coolingDevice.size(); index++) {
         std::cout << "Cooling device Id: " << coolingDevice(index)->getId()
                   << "Description: " << coolingDevice(index)->getDescription()
                   << "Max cooling level: " << coolingDevice(index)->getMaxCoolingLevel()
                   << "Current cooling level: " << coolingDevice(index)->getCurrentCoolingLevel()
                   << std::endl;
         std::cout << std::endl;
      }
   } else {
      std::cout << "No cooling devices found!" << std::endl;
   }
} else {
   std::cout << "Thermal manager is nullptr" << std::endl;
}
```

5.14.2 Using Thermal Shutdown Manager APIs to get Thermal autoshutdown mode updates

Using Thermal Shutdown Manager APIs to get Thermal autoshutdown mode updates

The below steps need to be followed by applications to listen to thermal auto-shutdown mode updates.

1. Implement IThermalShutdownListener interface

```
class MyThermalShutdownModeListener : public IThermalShutdownListener {
public:
    void onShutdownEnabled() override;
    void onShutdownDisabled() override;
    void onImminentShutdownEnablement(uint32_t imminentDuration) override;
    void onServiceStatusChange(ServiceStatus status) override;
};
```

2. Get thermal factory and thermal shutdown manager instances

```
auto &thermalFactory = telux::therm::ThermalFactory::getInstance();
auto thermShutdownMgr_ = thermalFactory.getThermalShutdownManager();
```

3. Check if thermal shutdown management service is ready.

```
bool subSystemsStatus = thermShutdownMgr_->isReady();
 if(subSystemsStatus) {
     std::cout << " Thermal-Shutdown management service is ready."<< std::endl;
 } else {
     std::cout << " Thermal-Shutdown management service is NOT ready."<< std::endl;
 }
```

4. If Thermal shutdown management service is not ready, wait for it to be ready

```
std::future<bool> f = thermShutdownMgr_->onSubsystemReady();
#if  //Timeout based wait
    if (f.wait_for(std::chrono::seconds(timeoutSec)) == std::future_status::timeout) {
        std::cout << "operation timed out." << std::endl;
    } else {
        subSystemsStatus = f.get();
        if (subSystemsStatus) {
            std::cout << " Thermal-Shutdown management service is ready." << std::endl;
```

```
        }
    }
#else //Unconditional wait
    subSystemsStatus = f.get();
    if (subSystemsStatus) {
        std::cout << " Thermal-Shutdown management service is ready." << std::endl;
    } else {
        std::cout << " Thermal-Shutdown management service is NOT ready." << std::endl;
        return -1;
    }
#endif
```

5. Instantiate MyThermalStateListener

```
auto myThermalModeListener = std::make_shared<MyThermalShutdownModeListener>();
```

6. Register for updates on thermal autoshutdown mode and its management service status

```
thermShutdownMgr_->registerListener(myThermalModeListener);
```

7. Wait for the Thermal auto shutdown mode updates

```
// Avoid long blocking calls when handling notifications
void MyThermalShutdownModeListener::onShutdownEnabled() {
    std::cout << std::endl << "**** Thermal auto shutdown mode enabled ****" << std::endl;
}
void MyThermalShutdownModeListener::onShutdownDisabled() {
    std::cout << std::endl << "**** Thermal auto shutdown mode disabled ****" << std::endl;
}
void MyThermalShutdownModeListener::onImminentShutdownEnablement(uint32_t imminentDuration) {
    std::cout << std::endl << "**** Thermal auto shutdown mode will be enabled in "
        << imminentDuration << " seconds ****" << std::endl;
}
```

8. When the Thermal shutdown management service goes down, this API is invoked with status UNAVAILABLE. All Thermal auto shutdown mode notifications will be stopped until the status becomes AVAILABLE again

```
void MyThermalShutdownModeListener::onServiceStatusChange(ServiceStatus status) {
    std::cout << std::endl << "**** Thermal-Shutdown management service status update ****" << std::endl;
    // Avoid long blocking calls when handling notifications
}
```

5.14.3 Using Thermal Shutdown Manager APIs to set Autoshutdown modes

Using Thermal Shutdown Manager APIs to set Autoshutdown modes

Please follow below steps to set thermal autoshutdown modes

1. Get thermal factory and thermal shutdown manager instances

```
auto &thermalFactory = telux::therm::ThermalFactory::getInstance();
auto thermShutdownMgr_ = thermalFactory.getThermalShutdownManager();
```

2. Check if thermal shutdown management service is ready.

```
bool subSystemsStatus = thermShutdownMgr_->isReady();
 if(subSystemsStatus) {
     std::cout << APP_NAME << " Thermal-Shutdown management service is ready."<< std::endl;
 } else {
```

```
    std::cout << APP_NAME << " Thermal-Shutdown management service is NOT ready."<< std::endl;
 }
```

3 If Thermal shutdown management service is not ready, wait for it to be ready

```
std::future<bool> f = thermShutdownMgr_->onSubsystemReady();
#if  //Timeout based wait
    if (f.wait_for(std::chrono::seconds(timeoutSec)) == std::future_status::timeout) {
        std::cout << "operation timed out." << std::endl;
    } else {
        subSystemsStatus = f.get();
        if (subSystemsStatus) {
            std::cout << " Thermal-Shutdown management service is ready." << std::endl;
        }
    }
#else //Unconditional wait
    subSystemsStatus = f.get();
    if (subSystemsStatus) {
        std::cout << " Thermal-Shutdown management service is ready." << std::endl;
    } else {
        std::cout << " Thermal-Shutdown management service is NOT ready" << std::endl;
        return -1;
    }
#endif
```

3. Query the current thermal auto shutdown mode

```
// Callback which provides response to query operation
void getStatusCallback(AutoShutdownMode mode)
{
    if(mode == AutoShutdownMode::ENABLE) {
        std::cout << " Current auto shutdown mode is: Enable" << std::endl;
    } else if(mode == AutoShutdownMode::DISABLE) {
        std::cout << " Current auto shutdown mode is: Disable" << std::endl;
    } else {
        std::cout << " *** ERROR - Failed to send get auto-shutdown mode " << std::endl;
    }
}
// Send get themal auto Shutdown mode command
auto status = thermShutdownMgr_->getAutoShutdownMode(getStatusCallback);
if(status != telux::common::Status::SUCCESS) {
    std::cout << "getShutdownMode command failed with error" << static_cast<int>(status) << std::endl;
} else {
    std::cout << "Request to query thermal shutdown status sent" << std::endl;
}
```

4. Set thermal auto shutdown mode

```
// Callback which provides response to set thermal auto shutdown mode command
void commandResponse(ErrorCode error)
{
    if(error == ErrorCode::SUCCESS) {
    std::cout << " sent successfully" << std::endl;
    } else {
        std::cout << " failed\n errorCode: " << static_cast<int>(error) << std::endl;
    }
}
// Send set themal auto Shutdown mode command
auto status = thermShutdownMgr_->setAutoShutdownMode(state, commandResponse);
if(status != telux::common::Status::SUCCESS) {
    std::cout << "setShutdownMode command failed with error" << static_cast<int>(status) << std::endl;
} else {
    std::cout << "Request to set thermal shutdown status sent" << std::endl;
}
```

5.15 sensor

The list of sample apps related to sensor:

- Using Sensor APIs to configure and acquire sensor data
- Using Sensor APIs to control sensor features

5.15.1 Using Sensor APIs to configure and acquire sensor data

Using Sensor APIs to configure and acquire sensor data

Please follow below steps as a guide to configure and acquire sensor data

1. Get sensor factory

```
auto &sensorFactory = telux::sensor::SensorFactory::getInstance();
```

2. Prepare a callback that is invoked when the sensor sub-system initialization is complete

```
std::promise<telux::common::ServiceStatus> p;
auto initCb = [&p](telux::common::ServiceStatus status) {
   std::cout << "Received service status: " << static_cast<int>(status) << std::endl;
   p.set_value(status);
};
```

3. Get the sensor manager. If initialization fails, perform necessary error handling

```
std::shared_ptr<telux::sensor::ISensorManager> sensorManager
   = sensorFactory.getSensorManager(initCb);
if (sensorManager == nullptr) {
   std::cout << "sensor manager is nullptr" << std::endl;
   exit(1);
}
std::cout << "obtained sensor manager" << std::endl;
```

4. Wait until initialization is complete

```
p.get_future().get();
if (sensorManager->getServiceStatus() != telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   std::cout << "Sensor service not available" << std::endl;
   exit(1);
}
```

5. Get information regarding available sensors in the system

```
std::cout << "Sensor service is now available" << std::endl;
std::vector<telux::sensor::SensorInfo> sensorInfo;
telux::common::Status status = sensorManager->getAvailableSensorInfo(sensorInfo);
if (status != telux::common::Status::SUCCESS) {
   std::cout << "Failed to get information on available sensors" << static_cast<int>(status)
             << std::endl;
   exit(1);
}
std::cout << "Received sensor information" << std::endl;
for (auto info : sensorInfo) {
   printSensorInfo(info);
}
```

6. Request the ISensorManager for the desired sensor

```
std::shared_ptr<telux::sensor::ISensor> sensor;
std::cout << "Getting sensor: " << name << std::endl;
status = sensorManager->getSensor(sensor, name);
if (status != telux::common::Status::SUCCESS) {
   std::cout << "Failed to get sensor: " << name << std::endl;
   exit(1);
}
```

7. Create and register a sensor event listener for configuration updates and sensor events

The event listener extends the ISensorEventListener to receive notification about configuration and sensor events.

```
class SensorEventListener : public telux::sensor::ISensorEventListener {
public:
   SensorEventListener(std::shared_ptr<telux::sensor::ISensor> sensor)
      : sensor_(sensor)
      , totalBatches_(0) {
   }

   // [11] Receive sensor events. This notification is received every time the configured batch
   // count is available with the sensor framework
   virtual void onEvent(
      std::shared_ptr<std::vector<telux::sensor::SensorEvent>> events) override {

      PRINT_NOTIFICATION << ": Received " << events->size()
                         << " events from sensor: "
                         << sensor_->getSensorInfo().name << std::endl;

      // I/O intense operations such as below should be avoided since this thread should avoid
      // any time consuming operations
      for (telux::sensor::SensorEvent s : *(events.get())) {
            printSensorEvent(s);
      }
      ++totalBatches_;
      // [11.1] If we have received expected number of batches and want to reconfigure the sensor
      // we will spawn the request to deactivate, configure and activate on a different thread
      // since we are not allowed to invoke the sensor APIs from this thread context
      if (totalBatches_ > TOTAL_BATCHES_REQUIRED) {
            totalBatches_ = 0;
            std::thread t([&] {
               sensor_->deactivate();
```

```
                sensor_->configure(sensor_->getConfiguration());
                sensor_->activate();
            });
            // Be sure to detach the thread
            t.detach();
        }
    }

    // [9] Receive configuration updates
    virtual void onConfigurationUpdate(
        telux::sensor::SensorConfiguration configuration) override {
        PRINT_NOTIFICATION
            << ": Received configuration update from sensor: " << sensor_->getSensorInfo().name
            << ": [" << configuration.samplingRate << ", " << configuration.batchCount << " ]"
            << std::endl;
    }

private:
    bool isUncalibratedSensor(telux::sensor::SensorType type) {
        return ((type == telux::sensor::SensorType::GYROSCOPE_UNCALIBRATED)
                || (type == telux::sensor::SensorType::ACCELEROMETER_UNCALIBRATED));
    }
    void printSensorEvent(telux::sensor::SensorEvent &s) {
        telux::sensor::SensorInfo info = sensor_->getSensorInfo();
        if (isUncalibratedSensor(sensor_->getSensorInfo().type)) {
            PRINT_NOTIFICATION << ": " << sensor_->getSensorInfo().name << ": " << s.timestamp
                               << ", " << s.uncalibrated.data.x << ", " << s.uncalibrated.data.y
                               << ", " << s.uncalibrated.data.z << ", " << s.uncalibrated.bias.x
                               << ", " << s.uncalibrated.bias.y << ", " << s.uncalibrated.bias.z
                               << std::endl;
        } else {
            PRINT_NOTIFICATION << ": " << sensor_->getSensorInfo().name << ": " << s.timestamp
                               << ", " << s.calibrated.x << ", " << s.calibrated.y << ", "
                               << s.calibrated.z << std::endl;
        }
    }
    std::shared_ptr<telux::sensor::ISensor> sensor_;
    uint32_t totalBatches_;
};
```

Create a event listener and register it with the sensor.

```
std::shared_ptr<SensorEventListener> sensorEventListener
    = std::make_shared<SensorEventListener>(sensor->getSensorInfo());
sensor->registerListener(sensorEventListener);
```

8. Configure the sensor with required configuration setting the necessary validityMask

```
telux::sensor::SensorConfiguration config;
config.samplingRate = getMinimumSamplingRate(sensor->getSensorInfo());
config.batchCount = sensor->getSensorInfo().maxBatchCountSupported;
std::cout << "Configuring sensor with samplingRate, batchCount [" << config.samplingRate << ", "
        << config.batchCount << "]" << std::endl;
config.validityMask.set(telux::sensor::SensorConfigParams::SAMPLING_RATE);
config.validityMask.set(telux::sensor::SensorConfigParams::BATCH_COUNT);
status = sensor->configure(config);
if (status != telux::common::Status::SUCCESS) {
    std::cout << "Failed to configure sensor: " << name << std::endl;
    exit(1);
}
```

9. Receive updates on sensor configuration

```
virtual void onConfigurationUpdate(telux::sensor::SensorConfiguration configuration) override {
    PRINT_NOTIFICATION << ": Received configuration update from sensor: " << info_.name << ": ["
                       << configuration.samplingRate << ", " << configuration.batchCount << " ]"
                       << std::endl;
```

```
}
```

10. Activate the sensor to receive sensor data

```
status = sensor->activate();
if (status != telux::common::Status::SUCCESS) {
   std::cout << "Failed to activate sensor: " << name << std::endl;
   exit(1);
}
```

11. Receive sensor data with the registered listener

Avoid any time consuming operation in this callback. This thread should be released back the SDK library to avoid latency.

If any sensor APIs need to be called in this method, they should be done on a different thread. One such method is to spawn a detached thread that invokes the required API.

```
virtual void onEvent(std::shared_ptr<std::vector<telux::sensor::SensorEvent>> events) override {

   PRINT_NOTIFICATION << ": Received " << events->size()
                      << " events from sensor: " << sensor_->getSensorInfo().name << std::endl;

   // I/O intense operations such as below should be avoided since this thread should avoid
   // any time consuming operations
   for (telux::sensor::SensorEvent s : *(events.get())) {
      printSensorEvent(s);
   }
   ++totalBatches_;
   // [11.1] If we have received expected number of batches and want to reconfigure the sensor
   // we will spawn the request to deactivate, configure and activate on a different thread
   // since we are not allowed to invoke the sensor APIs from this thread context
   if (totalBatches_ > TOTAL_BATCHES_REQUIRED) {
      totalBatches_ = 0;
      std::thread t([&] {
            sensor_->deactivate();
            sensor_->configure(sensor_->getConfiguration());
            sensor_->activate();
      });
      // Be sure to detach the thread
      t.detach();
   }
}
```

12. When data acquisition is no longer necessary, deactivate the sensor

```
status = sensor->deactivate();
if (status != telux::common::Status::SUCCESS) {
   std::cout << "Failed to deactivate sensor: " << name << std::endl;
   exit(1);
}
```

13. Release the instance of ISensor if no longer required

```
sensor = nullptr;
```

14. Release the instance of ISensorManager to cleanup resources

```
sensorManager = nullptr;
```

5.15.2 Using Sensor APIs to control sensor features

Using Sensor APIs to control sensor features

Please follow below steps as a guide to control sensor features

1. Get sensor factory

```
auto &sensorFactory = telux::sensor::SensorFactory::getInstance();
```

2. Prepare a callback that is invoked when the sensor sub-system initialization is complete

```
std::promise<telux::common::ServiceStatus> p;
auto initCb = [&p](telux::common::ServiceStatus status) {
   std::cout << "Received service status: " << static_cast<int>(status) << std::endl;
   p.set_value(status);
};
```

3. Get the sensor feature manager. If initialization fails, perform necessary error handling

```
std::shared_ptr<telux::sensor::ISensorFeatureManager> sensorFeatureManager
    = sensorFactory.getSensorFeatureManager(initCb);
if (sensorFeatureManager == nullptr) {
   std::cout << "sensor feature manager is nullptr" << std::endl;
   exit(1);
}
std::cout << "obtained sensor feature manager" << std::endl;
```

4. Wait until initialization is complete

```
p.get_future().get();
if (sensorManager->getServiceStatus() != telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   std::cout << "Sensor service not available" << std::endl;
   exit(1);
}
```

5. Get information regarding available sensor features

```
std::cout << "Sensor feature service is now available" << std::endl;
std::vector<telux::sensor::SensorFeature> sensorFeatures;
telux::common::Status status = sensorFeatureManager->getAvailableFeatures(sensorFeatures);
if (status != telux::common::Status::SUCCESS) {
   std::cout << "Failed to get information on available features" << static_cast<int>(status)
             << std::endl;
   exit(1);
}
std::cout << "Received sensor features" << std::endl;
for (auto feature : sensorFeatures) {
   printSensorFeatureInfo(feature);
}
```

6. Create and register a sensor feature event listener

```
std::shared_ptr<SensorFeatureEventListener> sensorFeatureEventListener
    = std::make_shared<SensorFeatureEventListener>();
sensorFeatureManager->registerListener(sensorFeatureEventListener);
```

7. Enable the required features

```
status = sensorFeatureManager->enableFeature(name);
if (status != telux::common::Status::SUCCESS) {
   std::cout << "Failed to enable feature: " << name << std::endl;
   exit(1);
}
```

8. Receive sensor feature events with the registered listener

```
virtual void onEvent(telux::sensor::SensorFeatureEvent event) override {
   printSensorFeatureEvent(event);
}
```

9. When the sensor feature(s) are no longer necessary, disable them

```
status = sensorFeatureManager->disableFeature(name);
if (status != telux::common::Status::SUCCESS) {
   std::cout << "Failed to disable feature: " << name << std::endl;
   exit(1);
}
```

10. Release the instance of ISensorFeatureManager to cleanup resources

```
sensorFeatureManager = nullptr;
```