Qualcomm Technologies, Inc.

# Telematics SDK

## User Guide

## v1.63.16

80-79288-1 Rev. AF

April 17, 2026

About Qualcomm

Qualcomm relentlessly innovates to deliver intelligent computing everywhere, helping the world tackle some of its most important challenges. Building on our 40 years of technology leadership in creating era-defining breakthroughs, we deliver a broad portfolio of solutions built with our leading-edge AI, high-performance, low-power computing, and unrivaled connectivity. Our Snapdragon® platforms power extraordinary consumer experiences, and our Qualcomm Dragonwing™ products empower businesses and industries to scale to new heights. Together with our ecosystem partners, we enable next-generation digital transformation to enrich lives, improve businesses, and advance societies. At Qualcomm, we are engineering human progress.

# Revision history

| Revision | Date | Description |
|---|---|---|
| AA | July 2024 | Initial release |
| AB | December 2024 | Updated to V1.63.9 |
| AC | May 2025 | Updated to V1.63.11 |
| AD | October 2025 | Updated to V1.63.13 |
| AE | November 2025 | Updated to V1.63.13 |
| AF | April 2026 | Updated to V1.63.16 |

# Software Revision History

| Revision | Date | Description |
|---|---|---|
| **AW** | Apr 2026 | Updated documentation for API reference and user guide for V1.63.16. |
| **AV** | Oct 2025 | Updated documentation for API reference and user guide for V1.63.13. |
| **AU** | May 2025 | Updated documentation for API reference and user guide for V1.63.11. |
| **AT** | Dec 2024 | Updated documentation for API reference and user guide for V1.63.10. |
| **AR** | Jun 2024 | Remove unsupported simulation framework. Updated documentation for API reference and user guide for V1.63.9. |
| **AP** | Oct 2023 | Add support for Diagnostics services. Updated documentation for API reference and user guide for V1.63.0. |
| **AN** | Sep 2023 | Updated documentation for API reference and user guide for V1.62.0. |
| **AM** | Aug 2023 | Updated documentation for API reference and user guide for V1.61.0. |
| **AL** | Jul 2023 | Updated documentation for API reference and user guide for V1.60.0. |
| **AK** | Jun 2023 | Updated documentation for API reference and user guide for V1.59.0. |
| **AJ** | May 2023 | Updated documentation for API reference and user guide for V1.58.0. |
| **AH** | Apr 2023 | Updated documentation for API reference and user guide for V1.57.0. |
| **AG** | Mar 2023 | Added support for thermal notifications(Trip update, Cdev level change). Added sample apps for thermal notifications. Updated documentation for API reference and user guide for V1.56.0. |
| **AF** | Dec 2022 | Added support for Third party service (TPS) eCall over IMS. Added sample apps for crypto accelerator. |
| **AE** | Nov 2022 | Add support for WLAN features. Updated documentation for API reference and user guide for V1.54.0. |
| **AD** | Sep 2022 | Added information of linux group IDs. |
| **AC** | Jul 2022 | Added sample apps for crypto APIs. |
| **AB** | Mar 2022 | Addition of documentation for Call flow of control filesystem for ECALL operation. |
| **AA** | Jan 2022 | Addition of documentation for sensor self test APIs. |
| | Dec 2021 | Addition of documentation for ECALL and OTA operation APIs. |
| | Nov 2021 | Added support for Xtra feature in location. |
| | Aug 2021 | Added telsdk data subsystem behavior details during SSR. Updated EFS APIs related information for platform subsystem. |
| | Jul 2021 | Updated thermal subsystem readiness flow for new methodology. |
| | Jun 2021 | Added documentation for remote SIM provisioning APIs. Added documentation for platform subsystem. |
| | Apr 2021 | Added documentation for sensor subsystem. Added documentation for data serving system features. |
| | Feb 2021 | Added documentation for audio subsystem readiness. |
| | Jan 2021 | Updated data subsystem readiness flow for new methodology. Added documentation for location simulation using stub libraries. |
| | May 2020 | Added documentation for L2TP feature. |
| | Mar 2020 | Added documentation for robust location APIs in location configurator. |
| | Feb 2020 | Added documentation for location configurator. |
| | Jan 2020 | Added documentation for firewall entry and socks proxy. Updated docs for firewall APIs Added documentation for software bridge management. |
| | Nov 2019 | Added documentation for compressed audio format playback on voice path. Added documentation for data networking APIs. Added documentation for audio transcoding APIs. |

**Table 1 Revision history (cont.)**

| Revision | Date | Description |
|---|---|---|
| | Oct 2019 | Update documentation for location concurrent reports and constraint time uncertainty. Added documentation for data filter feature. |
| | Sep 2019 | Added documentation for modem config APIs Added documentation for compressed audio format playback. Added documentation for audio loopback and tone generator APIs. Added documentation for remote SIM feature. |
| | Jul 2019 | Added documentation for thermal shutdown manager APIs. Updated sample app as per location APIs changes. |
| | Jun 2019 | Added audio play, capture and DTMF generation sample apps. |
| | May 2019 | Added documentation for TCU activity management feature. |
| | Mar 2019 | Added sample app for using thermal manager. |
| | Jan 2019 | Added documentation for audio manager and voice call. |
| | Nov 2018 | Addition of C-V2X. |
| | Oct 2018 | Updated steps to build Yocto platform SDK for LE.UM.1.3.2. Added documentation for building Google test. |
| | Jun 2018 | Added sample apps for various data services. Added instruction for creating platform SDK using CAF. |
| | Dec 2017 | Added subscription feature. |
| | Sep 2017 | Initial Release. |

# Contents

# 1 Introduction

## 1.1 Purpose

The Telematics software development kit (TelSDK) is a set of application programming interfaces (APIs) that provide access to the QTI-specific hardware and software capabilities.

The primary purpose of this document is to outline various features of TelSDK and demonstrate how to use various TelSDK APIs to develop an application. It provides sample application's source code to familiarize developer with the API usage.

## 1.2 Scope

This document focusses on how to use TelSDK APIs to implement a given use-case. It assumes that the developer is familiar with Linux and C++11 programming.

# 2 Building TelSDK based applications

## 2.1 You have access to ChipCode portal

The TelSDK deliverables (headers, libraries, prebuilt artifacts, selinux policies etc.) are delivered to the Qualcomm Technologies, Inc customers as part of a software product release through ChipCode™portal. This delivery contains both public and non-public TelSDK deliverables. Given this information, the steps to develop and build an application would be:

1. Sync whole software product release package from ChipCode portal.
2. Write a TelSDK based application.
3. Write a Yocto recipe to build this application.
4. Integrate this application and recipe in the Yocto build system.
5. Finally, build the application and test it on the target hardware.

### 2.1.1 Setting up a build environment

The build environment can be setup by running set_bb_env.sh script in poky directory.

```
$ cd poky
$ source build/conf/set_bb_env.sh
$ build-<target>-image
```

The target can be sa515m, sa415m, sa2150p or 9650 etc.

### 2.1.2 Building a sample application

The build environment is ready once set_bb_env.sh script has been run. The TelSDK supplied sample applications can be built by using telux-samples recipe as shown below.

```
$ bitbake telux-samples
```

## 2.2 You do not have access to ChipCode portal

It is possible to develop an application based on TelSDK up to a certain extent, using only publicly available part of the TelSDK source code. Minimum requirements and how they are met is discussed below:

1. An overall build environment which comprises of a cross-toolchain to compile application for a given QTI target processor's architecture. Standard Linux header files, libraries, root file system directory structure and various environment variables to facilitate build and link process. This build environment is obtained with the help of standard Yocto platform SDK application development and environment creation support feature.

1. The TelSDK header files and libraries to compile and link an application. This is done by enabling TelSDK specific recipes that builds and installs these header files and libraries in root file system used in our build environment.

### 2.2.1 Creating build environment

The steps to create such a build environment are as follows:

1. Install necessary packages on host machine to sync and build the Yocto platform SDK.
```
$ sudo apt-get install repo gawk wget git-core diffstat unzip texinfo xterm
$ sudo apt-get install gcc-multilib build-essential chrpath socat libsdl1.2-dev
```
2. Identify tag of the latest software image (SI) release from Qualcomm Technologies, Inc for the desired target processor from codeaurora forum. LE.UM.4.1.1.C9 is the SI for SA515M target, LE.-UM.3.2.1.C1 is the SI for SA415M target and LE.UM.1.3.r5 is SI for MDM9650 target. TAGS are listed here: https://source.codeaurora.org/quic/le/le/manifest/refs/tags
3. Create fully qualified manifest file name for the source code repository. Add "caf_" as prefix and ".xml" as suffix to the tag obtained at step 2. Example manifest file name: caf_AU_LINUX_EMBEDDED_LE.UM.4.1.1.C9_TARGET_ALL.01.311.164.xml
4. Sync the open-source code part of the software image release using manifest file obtained at step 3.
```
$ repo init -u git://codeaurora.org/quic/le/le/manifest.git -b release -m <caf_TAG.xml>
$ repo sync -j 16
```
5. Enable building and installing TelSDK libraries on target image by appending following lines in poky/build/conf/local.conf file.
(a) For all software images except LE.UM.1.3.r5:
```
CORE_IMAGE_EXTRA_INSTALL += "telux"
CORE_IMAGE_EXTRA_INSTALL += "telux-lib"
```
(b) For LE.UM.1.3.r5 software image:
```
CORE_IMAGE_EXTRA_INSTALL += "telux"
CORE_IMAGE_EXTRA_INSTALL += "telephony-lib"
```
6. Run the SDK environment setup script (required by Yocto build system).
```
$ cd poky
$ source build/conf/set_bb_env.sh
```
7. Setup the machine and OS distribution environment.
```
# For sa515m
$ export MACHINE=sa515m
$ export DISTRO=auto
# For sa415m
$ export MACHINE=sa415m
$ export DISTRO=auto
# For sa2150p
$ export MACHINE=sa2150p
$ export DISTRO=msm
```
8. Build the Yocto platform SDK and generate SDK installer.
```
$ bitbake core-image-minimal -c do_populate_sdk
```
Once the build completes, the SDK installer will be found in poky/build/tmp-glibc/deploy/sdk directory. For example, oecore-x86_64-armv7at2hf-neon-toolchain-nodistro.0.sh is the installer for SA515M LE.2.1.
9. Finally, install the build environment on host machine using installer generated at step 8. When prompted specify directory for installation.
```
$ poky/build/tmp-glibc/deploy/sdk/oecore-x86_64-<arch>-toolchain-nodistro.0.sh
```
10. An application can be now written and compiled against TelSDK libraries by following build procedure mentioned in Yocto SDK project manual here:

https://docs.yoctoproject.org/sdk-manual/index.html

### 2.2.2 Building a sample application

Our build environment is setup now. As an example, TelSDK supplied location sample application for SA515M LE.2.1 can be built by executing following instructions.

```
$ cd <platform_sdk_installation_directory>
$ source environment-setup-armv7at2hf-neon-oe-linux-gnueabi
$ cd telux/public/apps/samples/loc/loc_app/
$ ${CC} SampleLocationApp.cpp -ltelux_loc -std=c++11 -lstdc++ -o testapp
```

# 3 Sample Applications

Sample applications provided by TelSDK quickly demonstrates how TelSDK APIs can be used to achieve end functionality. Full source code for all sample applications is in /telux/public/apps/samples directory.

**Note**: Some of the applications contain default values for parameters that might differ from product to product. These values can be overridden by using configuration files used by these applications. The configuration is specified in the form of key-value pair. For example, the number to dial used by make_call_app can be specified by DIAL_NUMBER item and then application can be run passing this file as command-line argument to application.

```
# Contents of /data/make_call_app.conf
DIAL_NUMBER = +1234512345

# Run the application
$ make_call_app /data/make_call_app.conf
```

- Audio
- CV2X
- Telephony
- Data
- Location
- Modem configuration
- Power
- Thermal
- Sensor
- Platform
- Crypto
- Crypto accelerator
- Cellular connection security
- WiFi connection security
- WLAN
- Diagnostics

## 3.1 Audio

The TelSDK audio APIs provides support for device and stream management, transcoding, recording/playing audio, voice call and DTMF tone generation/detection etc.

- Audio manager and stream
- Playback session
- Capture session
- Loopback session
- Tone generation
- Voice call session
- Volume and mute controls
- Switching audio device
- Play DTMF tone
- Detect DTMF tones
- Transcoding samples
- Compressed format playback
- Playback on voice paths

### 3.1.1 Audio manager and stream

Audio Manager provides APIs to create audio streams and transcoder. It also helps in knowing supported audio devices and audio calibration status.

#### 1. Get an AudioFactory instance

```
#include <telux/audio/AudioFactory.hpp>
#include <telux/audio/AudioManager.hpp>

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

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

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

#### 2. Get an AudioManager instance

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

#### 3. Let audio subsystem be ready

```
// 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;
}
```

### 4. Query supported audio devices

```
// 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);
```

### 5. Query supported audio stream types

```
// 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);
```

### 6. 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;
/*For StreamType::VOICE_CALL, sink and source device are required to be passed.
  First device should be sink (speaker) and then source (mic) should be passed
  for stream creation.*/
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_MIC);

status = audioManager->createStream(config, createStreamCallback);
```

#### 7. Delete an audio stream

```
// 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 the stream (voice call session), which was created earlier
status = audioManager->deleteStream(std::dynamic_pointer_cast<IAudioStream>(audioVoiceStream),
                                    deleteStreamCallback);
```

### 3.1.2 Playback session

This sample application demonstrates how to use audio APIs for a playback session.

#### 1. Get the AudioFactory instance

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

#### 2. Get the AudioManager instance and check for audio subsystem readiness

```
std::promise<ServiceStatus> prom{};
// Get the AudioManager instance
audioManager = audioFactory.getAudioManager([&prom](ServiceStatus serviceStatus) {
    prom.set_value(serviceStatus);
});
if (!audioManager) {
    std::cout << "Failed to get AudioManager instance" << 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 audio 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 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
 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 (we 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
// Very first write starts the 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. Dispose the audio stream

```
// 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 the audio stream
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;
}
```

### 3.1.3 Capture session

This sample application demonstrates how to use audio APIs for recording audio.

#### 1. Get the AudioFactory instance

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

#### 2. Get the AudioManager instance 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 instance" << 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 a capture audio 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
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 audio capture operation

```
// Get an audio buffer (we 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
// Very 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. Dispose audio stream once required number of samples are 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 the audio stream
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;
}
```

### 3.1.4 Loopback session

This sample application demonstrates how to create a loopback audio stream, start and stop the loopback session.

#### 1. Get the AudioFactory instance

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

#### 2. Get the AudioManager instance and check for audio subsystem readiness

```
std::promise<ServiceStatus> prom{};
// Get the 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 callback method 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
StreamConfig config;

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 callback method 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 callback method 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 session (which was started earlier)
status = audioLoopbackStream->stopLoopback(stopLoopbackCallback);
```

#### 6. Dispose the audio stream associated with the loopback session

```
// Implement a response callback method 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);
```

### 3.1.5 Tone generation

This sample application demonstrates how to use audio APIs to play a tone using tone generator stream.

### 1. Get the AudioFactory instance

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

### 2. Get the AudioManager instance 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 callback method 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
StreamConfig config;

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 the audio tone on a sink device

```
// Implement a response callback method 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 callback method 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 playing the tone (which was started earlier)
status = audioToneGeneratorStream->stopTone(stopToneCallback);
```

#### 6. Dispose the audio stream associated with the tone generator session

```
// Implement a response callback method 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);
```

### 3.1.6 Voice call session

This sample application demonstrates how to use audio APIs for a voice call session.

#### 1. Get the AudioFactory instance

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

#### 2. Get the AudioManager instance 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 instance" << 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 a 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 representing a 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;
/*For StreamType::VOICE_CALL, sink and source device are required to be passed.
  First device should be sink (speaker) and then source (mic) should be passed
  for stream creation.*/
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_MIC);

status = audioManager->createStream(config, createStreamCallback);
```

### 4. Start 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;
}

status = audioVoiceStream->startAudio(startAudioCallback);
```

### 5. Stop 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;
}

status = audioVoiceStream->stopAudio(stopAudioCallback);
```

#### 6. Dispose the audio stream

```
// 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 the audio stream (which was created earlier)
status = audioManager->deleteStream(std::dynamic_pointer_cast<IAudioStream>(audioVoiceStream),
                                    deleteStreamCallback);
```

### 3.1.7 Volume and mute controls

This sample application demonstrates how to set audio volume level, mute and unmute audio during an active voice session.

#### 1. Get the AudioFactory instance

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

#### 2. Get the AudioManager instance and check for audio subsystem readiness

```
std::promise<ServiceStatus> prom{};
// Get the AudioManager instance
audioManager = audioFactory.getAudioManager([&prom](ServiceStatus serviceStatus) {
    prom.set_value(serviceStatus);
});
if (!audioManager) {
    std::cout << "Failed to get AudioManager instance" << 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 a 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;
/*For StreamType::VOICE_CALL, sink and source device are required to be passed.
  First device should be sink (speaker) and then source (mic) should be passed
  for stream creation.*/
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_MIC);

status = audioManager->createStream(config, createStreamCallback);
```

### 4. Start the 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;
}

status = audioVoiceStream->startAudio(startAudioCallback);
```

### 5. Set volume level 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 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 current volume level of the audio stream

```
// 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;
    }
}

status = audioVoiceStream->getVolume(StreamDirection::RX, getStreamVolumeCallback);
```

### 7. Mute the audio for the 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;
}

// Mute audio stream for TX direction
StreamMute mute;
mute.dir = StreamDirection::TX;
mute.enable = true; //true: enable, false: disable

status = audioVoiceStream->setMute(mute, setStreamMuteCallback);
```

### 8 Get mute state of stream 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 state of stream for TX direction
status = audioVoiceStream->getMute(StreamDirection::TX, getStreamMuteCallback);
```

### 9. Stop voice 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 voice session (which was started earlier)
status = audioVoiceStream->stopAudio(stopAudioCallback);
```

#### 10. Dispose the audio stream

```
// 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 audio stream
status = audioManager->deleteStream(std::dynamic_pointer_cast<IAudioStream>(audioVoiceStream),
                                    deleteStreamCallback);
```

### 3.1.8 Switching audio device

This sample application demonstrates how to use audio APIs for switching audio devices during active voice session.

#### 1. Get the AudioFactory instance

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

#### 2. Get the AudioManager instance 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 instance" << 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 a 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 representing a 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;
// For StreamType::VOICE_CALL, sink and source device are required to be passed.
// First device requires to sink (speaker) then source (mic) should be passed.
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_MIC);

status = audioManager->createStream(config, createStreamCallback);
```

### 4. Start 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;
}

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;
}

// Switch to new device for the given audio stream
std::vector<DeviceType> devices;
/*For StreamType::VOICE_CALL, sink and source device are required to be passed.
  First device should be sink (speaker) and then source (mic) should be passed for stream creation.*/
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_MIC);
status = audioVoiceStream->setDevice(devices, setStreamDeviceCallback);
```

### 6. Query device used for a given audio stream

```
// 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++;
    }
}

status = audioVoiceStream->getDevice(getStreamDeviceCallback);
```

#### 7. Stop 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;
}

status = audioVoiceStream->stopAudio(stopAudioCallback);
```

#### 8. Dispose audio stream

```
//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();
}

status = audioManager->deleteStream(std::dynamic_pointer_cast<IAudioStream>(audioVoiceStream),
                                    deleteStreamCallback);
```

### 3.1.9 Play DTMF tone

This sample app demostrates how to use audio APIs to play DTMF tones during an active voice call. Please note that only Rx direction is supported currently.

#### 1. Get the AudioFactory instance

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

#### 2. Get the AudioManager instance 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 instance" << 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 a voice call session)

```
// Callback which provides response to createStream
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);
}

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;
/*For StreamType::VOICE_CALL, sink and source device are required to be passed.
  First device should be sink (speaker) and then source (mic) should be passed
  for stream creation.*/
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_MIC);

status = audioManager->createStream(config, createStreamCallback);
```

### 4. Start the voice call session

```
// Callback which provides response to startAudio
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;
}

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

```
// Callback which provides response to stopAudio
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;
}

status = audioVoiceStream->stopAudio(stopAudioCallback);
```

#### 7. Dispose the audio stream associated with the voice call session

```
// Implement a response callback method 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();
}

status = audioManager->deleteStream(std::dynamic_pointer_cast<IAudioStream>(audioVoiceStream),
                                    deleteStreamCallback);
```

### 3.1.10 Detect DTMF tones

This sample app demostrates how to use audio APIs to detect DTMF tones during an active voice call. Note that only Rx direction is supported currently.

#### 1. Get the AudioFactory instance

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

#### 2. Get the AudioManager instance 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 instance" << 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 a voice call session)

```
// Callback which provides response to createStream
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);
}

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;
/*For StreamType::VOICE_CALL, sink and source device are required to be passed.
  First device should be sink (speaker) and then source (mic) should be passed
  for stream creation.*/
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_MIC);

status = audioManager->createStream(config, createStreamCallback);
```

### 5. Start the voice call session

```
// Callback which provides response to startAudio
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;
}

status = audioVoiceStream->startAudio(startAudioCallback);
```

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

```
// Callback which provides response to registerListener
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

```
// Callback which provides response to stopAudio
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;
}

status = audioVoiceStream->stopAudio(stopAudioCallback);
```

#### 9. Dispose the audio stream associated with the voice call session

```
// Implement a response callback method 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();
}

status = audioManager->deleteStream(std::dynamic_pointer_cast<IAudioStream>(audioVoiceStream),
                                    deleteStreamCallback);
```

### 3.1.11 Transcoding samples

This sample app demonstrates how to use the audio APIs for transcoding audio samples.

#### 1. Get the AudioFactory instance

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

#### 2. Get the AudioManager instance 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 (we 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 (we 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;
}
```

### 3.1.12 Compressed format playback

This sample app demonstrates how to use the audio APIs for compressed audio format playback.

#### 1. Get the AudioFactory instance

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

#### 2. Get the AudioManager instance 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 instance" << 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 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);
}

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 (we 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. Dispose the audio stream, once end of operation is reached

```
// 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;
}

Status  status = audioManager->deleteStream(audioPlayStream, deleteStreamCallback);
if (status != Status::SUCCESS) {
    std::cout << "deleteStream failed with error" << static_cast<int>(status) << std::endl;
}
```

### 3.1.13 Playback on voice paths

This sample app demonstrates how to use the audio APIs for compressed audio format playback on voice paths.

#### 1. Get the AudioFactory instance

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

#### 2. Get the AudioManager instance 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 instance" << 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 playback session with voice path 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 don't 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 (we 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. Dispose the audio stream, once end of operation is reached

```
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;
}
```

## 3.2 CV2X

- Get CV2X service status
- Receiving data
- Transmitting data
- Setting verification load
- Obtaining adjust filter rate notification

### 3.2.1 Get CV2X service status

This sample app demonstrates how to use the C-V2X Radio Manager API to get the C-V2X status.

#### 1. Create a RequestCv2xStatusCallback method

```
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 method 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.

#### 2. Implement initialization callback and get the Cv2xRadioManager instance

Optionally initialization callback can be provided with get manager instance. CV2X factory will call callback when manager initialization is complete.

#### 2.1 Create a InitResponseCb lambda function

```
bool cv2xRadioManagerStatusUpdated = false;
telux::common::ServiceStatus cv2xRadioManagerStatus =
    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);
    cv2xRadioManagerStatusUpdated = true;
    cv2xRadioManagerStatus = status;
    cv.notify_all();
};
```

#### 2.2 Get a handle to the ICv2xRadioManager instance

```
auto & cv2xFactory = Cv2xFactory::getInstance();
auto cv2xRadioManager = cv2xFactory.getCv2xRadioManager(statusCb);
```

#### 2.3 Wait for C-V2X Radio Manager readiness

```
std::unique_lock<std::mutex> lck(mtx);
cv.wait(lck, [&] { return cv2xRadioManagerStatusUpdated; });
```

#### 2.4 Check C-V2X Radio Manager initialization state

If cv2xRadioManager initialization failed, new initialization attempt can be accomplished by calling step 2.2. If initialization succeed, proceed to step 3.

```
if (status == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   // Go to step 3
}
else {
   // Go to step 2.2 for another initialization attempt
}
```

#### 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;
}

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

### 3.2.2 Receiving data

This sample app demonstrates how to use the C-V2X Radio Manager API to receive C-V2X data.

#### 1. Create Callback methods for ICv2xRadio and ICv2xRadioManager methods

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

// Callback method 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 callbacks.

#### 2. Implement initialization callback and get the Cv2xRadioManager instance

Optionally initialization callback can be provided with get manager instance. CV2X factory will call callback when manager initialization is complete.

#### 2.1 Create a InitResponseCb lambda function

```
bool cv2xRadioManagerStatusUpdated = false;
telux::common::ServiceStatus cv2xRadioManagerStatus =
    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);
    cv2xRadioManagerStatusUpdated = true;
    cv2xRadioManagerStatus = status;
    cv.notify_all();
};
```

#### 2.2 Get a handle to the ICv2xRadioManager instance

```
auto & cv2xFactory = Cv2xFactory::getInstance();
auto cv2xRadioManager = cv2xFactory.getCv2xRadioManager(statusCb);
```

#### 2.3 Wait for C-V2X Radio Manager readiness

```
std::unique_lock<std::mutex> lck(mtx);
cv.wait(lck, [&] { return cv2xRadioManagerStatusUpdated; });
```

### 2.4 Check C-V2X Radio Manager initialization state

If cv2xRadioManager initialization failed, new initialization attempt can be accomplished by calling step 2.2. If initialization succeed, proceed to step 3.

```
if (status == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   // Go to step 3
}
else {
   // Go to step 2.2 for another initialization attempt
}
```

### 3. Request the C-V2X status

We want to verify that the C-V2X RX status is ACTIVE before we trying to receive any 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. Implement initialization callback and get the ICv2xRadio instance

Optionally initialization callback can be provided with get C-V2X Radio instance. CV2X Radio Manager will call callback when C-V2X Radio initialization is complete.

### 4.1 Create a InitResponseCb lambda function

```
bool cv2x_radio_status_updated = false;
telux::common::ServiceStatus cv2xRadioStatus =
    telux::common::ServiceStatus::SERVICE_UNAVAILABLE;

auto cb = [&](ServiceStatus status) {
    std::lock_guard<std::mutex> lock(mtx);
    cv2x_radio_status_updated = true;
    cv2xRadioStatus = status;
    cv.notify_all();
};
```

### 4.2 Get a handle to the ICv2xRadio instance

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

### 4.3 Wait for C-V2X Radio readiness

```
std::unique_lock<std::mutex> lck(mtx);
cv.wait(lck, [&] { return cv2x_radio_status_updated; });
```

### 4.4 Check C-V2X Radio initialization state

If cv2xRadio initialization failed, new initialization attempt can be accomplished by calling step 4.2. If initialization succeed, proceed to step 5.

```
if (cv2xRadioStatus == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   // Go to step 5
}
```

```
else {
   // Go to step 4.2 for another initialization attempt
}
```

#### 5. 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();
}
```

#### 6. Close RX subscription

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());
```

### 3.2.3 Transmitting data

This sample app demonstrates how to use the C-V2X Radio Manager API to send C-V2X data.

#### 1. Create Callback methods for ICv2xRadio and ICv2xRadioManager methods

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

// Callback method 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 callbacks.

#### 2. Implement initialization callback and get the Cv2xRadioManager instance

Optionally initialization callback can be provided with get manager instance. CV2X factory will call callback when manager initialization is complete.

#### 2.1 Create a InitResponseCb lambda function

```
bool cv2xRadioManagerStatusUpdated = false;
telux::common::ServiceStatus cv2xRadioManagerStatus =
    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);
    cv2xRadioManagerStatusUpdated = true;
    cv2xRadioManagerStatus = status;
    cv.notify_all();
};
```

#### 2.2 Get a handle to the ICv2xRadioManager instance

```
auto & cv2xFactory = Cv2xFactory::getInstance();
auto cv2xRadioManager = cv2xFactory.getCv2xRadioManager(statusCb);
```

#### 2.3 Wait for C-V2X Radio Manager readiness

```
std::unique_lock<std::mutex> lck(mtx);
cv.wait(lck, [&] { return cv2xRadioManagerStatusUpdated; });
```

#### 2.4 Check C-V2X Radio Manager initialization state

If cv2xRadioManager initialization failed, new initialization attempt can be accomplished by calling step 2.2. If initialization succeed, proceed to step 3.

```
if (status == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   // Go to step 3
}
else {
   // Go to step 2.2 for another initialization attempt
}
```

#### 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 Tx 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. Implement initialization callback and get the ICv2xRadio instance

Optionally initialization callback can be provided with get C-V2X Radio instance. CV2X Radio Manager will call callback when C-V2X Radio initialization is complete.

#### 4.1 Create a InitResponseCb lambda function

```
bool cv2x_radio_status_updated = false;
telux::common::ServiceStatus cv2xRadioStatus =
    telux::common::ServiceStatus::SERVICE_UNAVAILABLE;

auto cb = [&](ServiceStatus status) {
    std::lock_guard<std::mutex> lock(mtx);
    cv2x_radio_status_updated = true;
    cv2xRadioStatus = status;
    cv.notify_all();
};
```

#### 4.2 Get a handle to the ICv2xRadio instance

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

#### 4.3 Wait for C-V2X Radio readiness

```
std::unique_lock<std::mutex> lck(mtx);
cv.wait(lck, [&] { return cv2x_radio_status_updated; });
```

#### 4.4 Check C-V2X Radio initialization state

If cv2xRadio initialization failed, new initialization attempt can be accomplished by calling step 4.2. If initialization succeed, proceed to step 5.

```
if (cv2xRadioStatus == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   // Go to step 5
}
else {
   // Go to step 4.2 for another initialization attempt
}
```

#### 5. 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);
}
```

#### 6. Close TX SPS flow

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

### 3.2.4 Setting verification load

This sample app demonstrates how to use the C-V2X Radio Manager API to set the verificaton load.

#### 1. Create verification load callback method

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

// Callback method 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 method

```
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 instance

```
// 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>();
```

### 3.2.5 Obtaining adjust filter rate notification

This sample app demonstrates how client can notified to adjust the incoming message filtering rate.

#### 1. Implement ICv2xThrottleManagerListener interface

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

#### 2. Create a initialization status callback method

```
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

```
// 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

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

## 3.3 Telephony

- Make a voice call
- Make eCall
- Make Third Party Service (TPS) eCall over IMS
- Request voice service state updates
- set_radio_power
- Request service domain preference
- Get network subscription information
- Card service APIs to transmit APDU
- Using SAP APIs
- Request network selection mode
- Using remote SIM manager APIs
- Using remote SIM reference apps
- Sending SMS
- Listening for incoming SMS
- Provisioning remote SIM
- Remote SIM provisioning app

### 3.3.1 Make a voice call

This sample application demonstrates how to make a voice call.

#### 1. Implement ResponseCallback interface to receive subsystem initialization status

```
std::promise<telux::common::ServiceStatus> cbProm = std::promise<telux::common::ServiceStatus>();
void initResponseCb(telux::common::ServiceStatus status) {
   if(subSystemsStatus == SERVICE_AVAILABLE) {
      std::cout << Call Manager subsystem is ready << std::endl;
   } else if(subSystemsStatus == SERVICE_FAILED) {
      std::cout << Call Manager subsystem initialization failed << std::endl;
   }
   cbProm.set_value(status);
}
```

#### 2. Get the PhoneFactory and Call Manager instance

```
auto &phoneFactory = PhoneFactory::getInstance();
auto callManager = phoneFactory.getCallManager(initResponseCb);
if(callManager == NULL) {
   std::cout << " Failed to get Call Manager instance" << std::endl;
   return -1;
```

```
}
```

#### 3. Wait for Call Manager subsystem to be ready

```
telux::common::ServiceStatus status = cbProm.get_future().get();
if(status != SERVICE_AVAILABLE) {
   std::cout << Unable to initialize Call Manager subsystem << std::endl;
   return -1;
}
```

#### 4. Optionally, implement IMakeCallCallback interface to receive response for the dial request

```
class DialCallback : public IMakeCallCallback {
public:
   void DialCallback::makeCallResponse(ErrorCode error, std::shared_ptr<ICall> call) {
      // will be invoked with response of makeCall operation
   }
};
std::shared_ptr<DialCallback> dialCb = std::make_shared<DialCallback> ();
```

#### 5. Send a dial request

```
if(callManager) {
   std::string phoneNumber("+18989531755");
   int phoneId = 1;
   auto makeCallStatus = callManager->makeCall(phoneId, phoneNumber, dialCb);
   std::cout << "Dial Call Status:" << (int)makeCallStatus << std::endl;
}
```

### 3.3.2 Make eCall

This sample application demonstrates how to make an emergency (E112) voice call.

#### 1. Implement ResponseCallback interface to receive subsystem initialization status

```
std::promise<telux::common::ServiceStatus> cbProm = std::promise<telux::common::ServiceStatus>();
void initResponseCb(telux::common::ServiceStatus status) {
   if(subSystemsStatus == SERVICE_AVAILABLE) {
      std::cout << Call Manager subsystem is ready << std::endl;
   } else if(subSystemsStatus == SERVICE_FAILED) {
      std::cout << Call Manager subsystem initialization failed << std::endl;
   }
   cbProm.set_value(status);
}
```

#### 2. Get the PhoneFactory and Call Manager instance

```
auto &phoneFactory = PhoneFactory::getInstance();
auto callManager = phoneFactory.getCallManager(initResponseCb);
if(callManager == NULL) {
   std::cout << " Failed to get Call Manager instance" << std::endl;
   return -1;
}
```

#### 3. Wait for Call Manager subsystem to be ready

```
telux::common::ServiceStatus status = cbProm.get_future().get();
if(status != SERVICE_AVAILABLE) {
   std::cout << Unable to initialize Call Manager subsystem << std::endl;
   return -1;
}
```

#### 4. Optionally, implement IMakeCallCallback interface to receive response for the dial request

```
class DialCallback : public IMakeCallCallback {
public:
   void DialCallback::makeCallResponse(ErrorCode error, std::shared_ptr<ICall> call) {
      // will be invoked with response of makeCall operation
   }
};
std::shared_ptr<DialCallback> dialCb = std::make_shared<DialCallback> ();
```

#### 5. 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.msdVersion = 2;
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

```
int phoneId = 1;
if(callManager) {
   auto makeCallStatus = callManager->makeECall(phoneId, eCallMsdData, emergencyCategory,
                                                eCallVariant, dialCb);
   std::cout << "Dial ECall Status:" << (int)makeCallStatus << std::endl;
}
```

### 3.3.3 Make Third Party Service (TPS) eCall over IMS

This sample application demonstrates how to make a TPS eCall over IMS.

### 1. Implement ResponseCallback interface to receive subsystem initialization status

```
std::promise<telux::common::ServiceStatus> cbProm = std::promise<telux::common::ServiceStatus>();
void initResponseCb(telux::common::ServiceStatus status) {
   if(subSystemsStatus == SERVICE_AVAILABLE) {
      std::cout << Call Manager subsystem is ready << std::endl;
   } else if(subSystemsStatus == SERVICE_FAILED) {
      std::cout << Call Manager subsystem initialization failed << std::endl;
   }
   cbProm.set_value(status);
}
```

### 2. Get the PhoneFactory and Call Manager instance

```
auto &phoneFactory = PhoneFactory::getInstance();
auto callManager = phoneFactory.getCallManager(initResponseCb);
if(callManager == NULL) {
   std::cout << " Failed to get Call Manager instance" << std::endl;
   return -1;
}
```

### 3. Wait for Call Manager subsystem to be ready

```
telux::common::ServiceStatus status = cbProm.get_future().get();
if(status != SERVICE_AVAILABLE) {
   std::cout << Unable to initialize Call Manager subsystem << std::endl;
   return -1;
}
```

### 4. Optionally, instantiate dial call instance

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

### 4.1. Optionally, implement IMakeCallCallback interface to receive response for the dial request

```
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
}
```

### 5. Optionally, instantiate dial call instance

```
~~~~~~{.cpp}
CustomSipHeader header;
std::string contentTypeHeader;
std::string acceptInfoHeader;
char delimiter = '\n';
std::string temp = "";
std::cout << "Enter Custom SIP Header for contentType (uses default for no input): ";
std::getline(std::cin, temp, delimiter);
if (!temp.empty()) {
    contentTypeHeader = temp;
} else {
    std::cout << "No input, proceeding with default contentType: " << std::endl;
}
```

```
temp = "";
std::cout << "Enter Custom SIP Header for acceptInfo (uses default for no input): ";
std::getline(std::cin, temp, delimiter);
if (!temp.empty()) {
    acceptInfoHeader = temp;
} else {
    std::cout << "No input, proceeding with default acceptInfo: " << std::endl;
}
if (contentTypeHeader != "") {
    header.contentType = contentType;
} else {
    header.contentType = telux::tel::CONTENT_HEADER;
}
if (acceptInfoHeader != "") {
    header.acceptInfo = acceptInfo;
} else  {
    header.acceptInfo = "";
}
~~~~~~
```

#### 6. Initialize the data required for eCall such as dialnumber, msdData

```
std::string dialNumber = 77777777;
std::vector<uint8_t> rawData;
rawData = { 2, 41, 68, 6, 128, 227, 10, 81, 67, 158, 41, 85, 212, 56, 0, 128, 4, 52, 10, 140,
           65, 89, 164, 56, 119, 207, 131, 54, 210, 63, 65, 104, 16, 24, 8, 32, 19, 198, 68, 0,
           0, 48, 20 };
```

#### 7. Send a eCall request

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

### 3.3.4 Request voice service state updates

This sample application demonstrates how to request voice service state corresponding to a SIM in the device.

#### 1. Implement ResponseCallback interface to receive subsystem initialization status

```
std::promise<telux::common::ServiceStatus> cbProm = std::promise<telux::common::ServiceStatus>();
void initResponseCb(telux::common::ServiceStatus status) {
   if(subSystemsStatus == SERVICE_AVAILABLE) {
      std::cout << Phone Manager subsystem is ready << std::endl;
   } else if(subSystemsStatus == SERVICE_FAILED) {
      std::cout << Phone Manager subsystem initialization failed << std::endl;
   }
   cbProm.set_value(status);
}
```

#### 2. Get the PhoneFactory and PhoneManager instance

```
auto &phoneFactory = PhoneFactory::getInstance();
auto phoneManager = phoneFactory.getPhoneManager(initResponseCb);
if(phoneManager == NULL) {
   std::cout << " Failed to get Phone Manager instance" << std::endl;
   return -1;
}
```

### 3. Wait for Phone Manager subsystem to be ready

```
telux::common::ServiceStatus status = cbProm.get_future().get();
if(status != SERVICE_AVAILABLE) {
   std::cout << Unable to initialize Phone Manager subsystem << std::endl;
   return -1;
}
```

### 4. Get the phone object corresponding to a phone ID

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

### 5. Implement IOperatingModeCallback interface and instantiate MyOperatingModeCallback

```
std::promise<bool> callbackPromise;
telux::tel::OperatingMode operatingMode;
class MyOperatingModeCallback : public telux::tel::IOperatingModeCallback {
public:
   void operatingModeResponse(telux::tel::OperatingMode mode, telux::common::ErrorCode error) {
     if(error == telux::common::ErrorCode::SUCCESS) {
         std::cout << "Operating Mode is : " << static_cast<int>(mode)
             << std::endl;
         operatingMode = static_cast<telux::tel::OperatingMode>(mode);
     } else {
         std::cout << "Operating Mode is : Unknown, errorCode: " << static_cast<int>(error)
         << std::endl;
     }
     callbackPromise.set_value(true);
   }
};

auto myOperatingModeCallback = std::make_shared<MyOperatingModeCallback>();

void setOperatingModeResponse(telux::common::ErrorCode error) {
     std::cout << "Set Operating Mode is :, errorCode: " << static_cast<int>(error)
         << std::endl;
}
```

### 6. Request the operating mode of device and set the operating mode to ONLINE, if the operating mode is OFFLINE to perform any operations on the phone

```
phoneManager->requestOperatingMode(myOperatingModeCallback);
if((callbackPromise.get_future().get()) &&
   (telux::tel::operatingMode == telux::tel::OperatingMode::OFFLINE )) {
   phoneManager->setOperatingMode(telux::tel::OperatingMode::ONLINE, &setOperatingModeResponse);
}
```

### 7. 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;
};
```

### 8. Instantiate MyVoiceServiceStateCallback

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

#### 9. Send voice service state request

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

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

### 3.3.5 Request service domain preference

This sample application demonstrates how to request current 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 serving subsystem can not be initialized

```
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;
}
```

Now serviceDomainResponse() method will be invoked with current service domain preference.

### 3.3.6 Get network subscription information

This sample application demonstrates how to get modem network subscription information.

#### 1. Implement ResponseCallback interface to receive subsystem initialization status

```
std::promise<telux::common::ServiceStatus> cbProm = std::promise<telux::common::ServiceStatus>();
void initResponseCb(telux::common::ServiceStatus status) {
   if(subSystemsStatus == SERVICE_AVAILABLE) {
      std::cout << Subscription Manager subsystem is ready << std::endl;
   } else if(subSystemsStatus == SERVICE_FAILED) {
      std::cout << Subscription Manager subsystem initialization failed << std::endl;
   }
   cbProm.set_value(status);
}
```

#### 2. Get the PhoneFactory and SubscriptionManager instance

```
auto &phoneFactory = PhoneFactory::getInstance();
auto subscriptionMgr = phoneFactory.getSubscriptionManager(initResponseCb);
if(subscriptionMgr == NULL) {
   std::cout << " Failed to get Subscription Manager instance" << std::endl;
   return -1;
}
```

#### 3. Wait for Subscription subsystem subsystem to be ready

```
telux::common::ServiceStatus status = cbProm.get_future().get();
if(status != SERVICE_AVAILABLE) {
   std::cout << Unable to initialize Subscription Manager subsystem << std::endl;
   return -1;
}
```

#### 4. 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;
}
```

### 3.3.7 Card service APIs to transmit APDU

This sample application demonstrates how to use card service APIs to transmit APDU.

#### 1. Implement ResponseCallback interface to receive subsystem initialization status

```
std::promise<telux::common::ServiceStatus> cbProm = std::promise<telux::common::ServiceStatus>();
void initResponseCb(telux::common::ServiceStatus status) {
   if (subSystemsStatus == SERVICE_AVAILABLE) {
      std::cout << Card Manager subsystem is ready << std::endl;
   } else if(subSystemsStatus == SERVICE_FAILED) {
      std::cout << Card Manager subsystem initialization failed << std::endl;
   }
   cbProm.set_value(status);
}
```

#### 2. Get the PhoneFactory and CardManager instances

```
auto &phoneFactory = PhoneFactory::getInstance();
auto cardManager = phoneFactory.getCardManager(initResponseCb);
if (cardManager == NULL) {
   std::cout << " Failed to get Card Manager instance" << std::endl;
   return -1;
}
```

#### 3.Check if CardManager subsystem is ready

```
telux::common::ServiceStatus status = cardManager.getServiceStatus();
```

#### 3.1 Wait for the CardManager subsystem initialization

```
telux::common::ServiceStatus status = cbProm.get_future().get();
if (status != SERVICE_AVAILABLE) {
   std::cout << Unable to initialize Card Manager subsystem << std::endl;
   return -1;
}
```

#### 4. Get number of slots, their IDs 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());
```

#### 5. 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;
   }
}
```

### 6. Instantiate optional IOpenLogicalChannelCallback, ICommandResponseCallback and ITransmitApduResponseCallback

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

### 6.1 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();
   }
}
```

### 6.2. 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();
   }
}
```

### 6.3. 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();
   }
```

```
}
```

#### 7. 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;
```

#### 8. 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;
```

#### 9. Close the opened logical channel and wait for the completion

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

#### 10. 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;
```

### 3.3.8 Using SAP APIs

This sample application demonstrates how to use SAP APIs to transmit APDU and listen to SAP events.

#### 1. Implement ResponseCallback interface to receive subsystem initialization status

```
std::promise<telux::common::ServiceStatus> cbProm = std::promise<telux::common::ServiceStatus>();
void initResponseCb(telux::common::ServiceStatus status) {
   if(status == SERVICE_AVAILABLE) {
      std::cout << Phone Manager subsystem is ready << std::endl;
   } else if(status == SERVICE_FAILED) {
      std::cout << Phone Manager subsystem initialization failed << std::endl;
   }
   cbProm.set_value(status);
}
```

#### 2. Get the PhoneFactory instance and SapCardManager instance

```
auto &phoneFactory = PhoneFactory::getInstance();

std::promise<telux::common::ServiceStatus> prom;
auto sapCardMgr = phoneFactory.getSapCardManager(
    DEFAULT_SLOT_ID, [&](telux::common::ServiceStatus status) {
    prom.set_value(status);
});

if (!sapCardMgr) {
   std::cout << "ERROR - Failed to get SapCardManager instance" << std::endl;
   return;
}
```

### 3. Wait for the SapCardManager subsystem initialization

```
~~~~~~{.cpp}
telux::common::ServiceStatus sapCardMgrStatus = sapCardMgr->getServiceStatus();
if (sapCardMgrStatus != telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   std::cout << "SapCardManager subsystem is not ready , Please wait" << std::endl;
}

sapCardMgrStatus = prom.get_future().get();
if (sapCardMgrStatus == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   std::cout << "SapCardManager subsystem is ready" << std::endl;
} else {
   std::cout << "ERROR - Unable to initialize SapCardManager subsystem" << std::endl;
   return;
}

### 4. Instantiate ICommandResponseCallback, IAtrResponseCallback and ISapCardCommandCallback

   ~~~~~~{.cpp}
   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);
```

### 3.3.9 Request network selection mode

This sample application demonstrates how to request current 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 network selection subsystem can not be initialzed

```
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 response callback

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

Now, selectionModeResponse() callback gets invoked with current network selection mode information.

### 3.3.10 Using remote SIM manager APIs

This sample application demonstrates how to use remote SIM manager APIs for remote SIM card operations.

#### 1. Implement ResponseCallback interface to receive subsystem initialization status

```
std::promise<telux::common::ServiceStatus> cbProm = std::promise<telux::common::ServiceStatus>();
void initResponseCb(telux::common::ServiceStatus status) {
   if (subSystemsStatus == SERVICE_AVAILABLE) {
      std::cout << RemoteSim Manager subsystem is ready << std::endl;
   } else if(subSystemsStatus == SERVICE_FAILED) {
      std::cout << RemoteSim Manager subsystem initialization failed << std::endl;
   }
   cbProm.set_value(status);
}
```

#### 2. Get the PhoneFactory and RemoteSimManager instances

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

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

PhoneFactory &phoneFactory = PhoneFactory::getInstance();
auto remoteSimMgr =
    phoneFactory.getRemoteSimManager(DEFAULT_SLOT_ID, initResponseCb);
 if (remoteSimMgr == NULL) {
   std::cout << " Failed to get RemoteSim Manager instance" << std::endl;
   return -1;
}
```

#### 3.Check if telephony subsystem is ready

```
telux::common::ServiceStatus status = remoteSimMgr.getServiceStatus();
```

#### 3.1 Wait for the telephony subsystem initialization

```
   telux::common::ServiceStatus status = cbProm.get_future().get();
   if (status != SERVICE_AVAILABLE) {
      std::cout << Unable to initialize Card Manager subsystem << std::endl;
      return -1;
   }
```

```
### 4. Instantiate and register RemoteSimListener

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

### 4.1 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
    }
};
```

### 4.2 Implementation of event callback for asynchronous requests

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

### 5. 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;
}
```

### 6. 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;
}
```

#### 7. 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;
}
```

#### 8. 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;
}
```

### 3.3.11 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.

```
brctl delif bridge0 eth0
```

#### 1.2 Configure the IP Address on Both Devices

```
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

```
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

```
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).

### 3.3.12 Sending SMS

This sample application demonstrates how to send a SMS to a given cellphone specified by mobile number.

#### 1. Implement ResponseCallback interface to receive subsystem initialization status

```
std::promise<telux::common::ServiceStatus> cbProm = std::promise<telux::common::ServiceStatus>();
void initResponseCb(telux::common::ServiceStatus status) {
   if(subSystemsStatus == SERVICE_AVAILABLE) {
      std::cout << SmsManager subsystem is ready << std::endl;
   } else if(subSystemsStatus == SERVICE_FAILED) {
      std::cout << SmsManager subsystem initialization failed << std::endl;
   }
   cbProm.set_value(status);
}
```

#### 2. Get the PhoneFactory and default SmsManager instance

```
auto &phoneFactory = PhoneFactory::getInstance();
std::shared_ptr<ISmsManager> smsManager = phoneFactory.getSmsManager(initResponseCb);
if(smsMgr == NULL) {
   std::cout << " Failed to get Sms Manager  instance" << std::endl;
   return -1;
}
```

#### 3. Wait for SmsManager subsystem to be ready

```
telux::common::ServiceStatus status = cbProm.get_future().get();
if(status != SERVICE_AVAILABLE) {
   std::cout << Unable to initialize Sms Manager subsystem << std::endl;
   return -1;
}
```

#### 4. Instantiate SMS sent and delivery callback and implement ICommandResponseCallback interface to know SMS sent and delivery status

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

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;
}
```

#### 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);
}
```

Now, we will receive resonses in callbacks defined at step 3 (smsSentCb, smsDeliveryCb).

### 3.3.13 Listening for incoming SMS

This sample application demonstrates how to listen for an incoming SMS.

#### 1. Implement ResponseCallback interface to receive subsystem initialization status

```
std::promise<telux::common::ServiceStatus> cbProm = std::promise<telux::common::ServiceStatus>();
void initResponseCb(telux::common::ServiceStatus status) {
   if(subSystemsStatus == SERVICE_AVAILABLE) {
      std::cout << SmsManager subsystem is ready << std::endl;
   } else if(subSystemsStatus == SERVICE_FAILED) {
      std::cout << SmsManager subsystem initialization failed << std::endl;
   }
   cbProm.set_value(status);
}
```

#### 2. 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;
}
```

#### 3. Get the PhoneFactory and default SmsManager instance

```
auto &phoneFactory = PhoneFactory::getInstance();
std::shared_ptr<ISmsManager> smsMgr = phoneFactory.getSmsManager(initResponseCb);
if(smsMgr == NULL) {
   std::cout << " Failed to get Sms Manager  instance" << std::endl;
   return -1;
}
```

#### 4. Wait for SmsManager subsystem to be ready

```
telux::common::ServiceStatus status = cbProm.get_future().get();
if(status != SERVICE_AVAILABLE) {
   std::cout << Unable to initialize Sms Manager subsystem << std::endl;
   return -1;
}
```

#### 5. Instantiate global ISmsListener and register for incoming SMS

```
auto mySmsListener = std::make_shared<MySmsListener>();
if(smsMgr) {
   smsMgr->registerListener(mySmsListener);
}
```

#### 6. Wait for incoming SMS

```
std::cout << " *** wait for MySmsListener::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;
```

### 3.3.14 Provisioning remote SIM

This sample app 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. Implement ResponseCallback interface to receive subsystem initialization status

#### 1.1 Implement ResponseCallback interface to receive SimProfile Manager subsystem initialization status

```
std::promise<telux::common::ServiceStatus> cbSimProfileProm = std::promise<telux::common::ServiceStatus>();
void initResponseCb(telux::common::ServiceStatus status) {
   if (subSystemsStatus == SERVICE_AVAILABLE) {
      std::cout << SIM Profile subsystem is ready << std::endl;
   } else if(subSystemsStatus == SERVICE_FAILED) {
      std::cout << SIM Profile subsystem initialization failed << std::endl;
   }
   cbSimProfileProm.set_value(status);
}
```

#### 1.2 Implement ResponseCallback interface to receive Card Manager subsystem initialization status

```
std::promise<telux::common::ServiceStatus> cbCardProm = std::promise<telux::common::ServiceStatus>();
void initResponseCb(telux::common::ServiceStatus status) {
   if (subSystemsStatus == SERVICE_AVAILABLE) {
      std::cout << Card Manager subsystem is ready << std::endl;
   } else if(subSystemsStatus == SERVICE_FAILED) {
      std::cout << Card Manager subsystem initialization failed << std::endl;
   }
   cbCardProm.set_value(status);
}
```

#### 2. Get phone factory, SIM profile manager and Card manager instance

```
auto &phoneFactory = telux::tel::PhoneFactory::getInstance();
auto simProfileManager = phoneFactory.getSimProfileManager(cbSimProfileProm);
if (simProfileManager == NULL) {
   std::cout << " Failed to get SIMProfile Manager instance" << std::endl;
   return -1;
}
auto cardManager = phoneFactory.getCardManager(cbCardProm);
if (cardManager == NULL) {
   std::cout << " Failed to get Card Manager instance" << std::endl;
   return -1;
}
```

#### 3. Check if SIM profile subsystem is ready

```
telux::common::ServiceStatus status = simProfileManager.getServiceStatus();
```

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

```
telux::common::ServiceStatus status = cbSimProfileProm.get_future().get();
if (status != SERVICE_AVAILABLE) {
   std::cout << Unable to initialize SIMProfile Manager subsystem << std::endl;
   return -1;
}
```

### 4. Check if card subsystem is ready

```
telux::common::ServiceStatus status = cardManager.getServiceStatus();
```

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

```
telux::common::ServiceStatus status = cbCardProm.get_future().get();
if (status != SERVICE_AVAILABLE) {
   std::cout << Unable to initialize Card Manager subsystem << std::endl;
   return -1;
}
```

### 5. Return/Exit the application, if SIM profile and card manager subsystem can not be initialized

```
if((status == SERVICE_AVAILABLE) {
   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;
}
```

### 6. Instantiate and register RspListener

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

### 6.1 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
    }
```

### 7. Request EID of the eUICC

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

// Implement a callback method 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);
```

### 8. Add profile on the eUICC

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

// Implement a callback method 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);
```

#### 8.1 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
}
```

#### 8.2 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
}
```

#### 8.3 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.
}
```

### 9. Delete profile on the eUICC

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

// Implement a callback method 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);
```

### 10. 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 method 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);
```

### 11. Enable/disable profile on the eUICC

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

// Implement a callback method 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);
```

### 12. Update Nickname of the profile

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

// Implement a callback method 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);
```

### 13. Set SMDP+ server address on the eUICC

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

// Implement a callback method 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);
```

#### 14. 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 method 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);
```

#### 15. Memory reset on the eUICC

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

// Implement a callback method 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);
```

### 3.3.15 Remote SIM provisioning app

TelSDK provides reference app rsp_httpd which helps in exploring various features of remote provisioning APIs.

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.

#### 1. Run the remote SIM HTTP daemon on the device

```
# rsp_httpd
```

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

## 3.4 Data

- Start/Stop cellular data call
- Get available modem profiles
- Get/Set data filter mode
- Add data filter
- Remove data filter mode
- Enable/Disable firewall
- Create firewall DMZ
- Add Firewall Entry
- Adding a software bridge and enable its management
- Remove a software bridge and its management
- Create VLAN And bind it to a PDN
- Create static NAT entry
- Enable L2TP and add a tunnel
- Enable/Disable socks
- Get dedicated radio bearer status and indication
- Get service status and indication
- Get roaming status and indication
- Establish on-demand PDN connectivity
- Create traffic class and add QoS filter

### 3.4.1 Start/Stop cellular data call

This sample application demonstrates how to start or stop a 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 method

```
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 method

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

Now, response callback will be called for the stopDataCall response.

## 3.4.2 Get available modem profiles

This sample application demonstrates how 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 method

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

Now, receive DataProfileListCallback responses for requestProfileList request.

### 3.4.3 Get/Set data filter mode

This sample application demonstrates how 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;
}
```

### 3.4.4 Add data filter

This sample application demonstrates how to add a 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;
}
```

### 3.4.5 Remove data filter mode

This sample application demonstrates how 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 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;
}
```

### 3.4.6 Enable/Disable firewall

This sample application demonstrates how 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);
```

Now, response callback will be called for the setFirewall response.

### 3.4.7 Create firewall DMZ

This sample application demonstrates how 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 a 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);
```

Now, response callback will be called for the addDmz response.

### 3.4.8 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);
```

### 3.4.9 Adding a software bridge and enable its management

This sample application demonstrates how to add a software bridge and enable its 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

```
~~~~~~{.cpp}
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 enabling its 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);
```

Now, response callback will be invoked with the addBridge response.

#### 6. Enable the software bridge management if not enabled already

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

Now, response callback will be invoked with the enableBridge response.

### 3.4.10 Remove a software bridge and its management

This sample application demonstrates how to remove a software bridge and its 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

~~~~~~{.cpp}
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);
```

Now, response callback will be invoked with the list of software bridges.

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

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

Now, response callback will be invoked with the removeBridge response.

#### 7. Disable the software bridge management (if required)

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

Now, response callback will be invoked with the enableBridge response.

### 3.4.11 Create VLAN And bind it to a PDN

This sample application demonstrates how to create VLAN and bind it to a 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);
```

Now, response callback will be called for the createVlan response.

#### 6. 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;
};
```

#### 7. Bind created Vlan with user provided profile id

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

Now, response callback will be called for the bindWithProfile response.

### 3.4.12 Create static NAT entry

This sample application demonstrates how 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);
```

Now, response callback will be called for the addStaticNatEntry response.

### 3.4.13 Enable L2TP and add a tunnel

This sample application demonstrates how to enable L2TP and add a 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. Optionally, instantiate setConfig callback instance

```
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. Optionally, instantiate addTunnel callback instance

```
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. Add the tunnel to L2TP

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

### 3.4.14 Enable/Disable socks

This sample application demonstrates how 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. Optionally, instantiate enable Socks callback instance

```
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 using enableSocks()

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

Now, response callback will be called for the setFirewall response.

### 3.4.15 Get dedicated radio bearer status and indication

This sample application demonstrates how 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. Optioanlly, instantiate initialization callback

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

#### 3. Get the DataFactory and data serving system manager instance and check if the data serving system manager is ready or not

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

    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);
```

#### 4. Register for serving system listener

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

#### 5. 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;
}
```

Now, wait for dedicated radio bearer notifications.

### 3.4.16 Get service status and indication

This sample application demonstrates how to get service status and indications.

#### 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. Optionally, instantiate an initialization callback

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

#### 3. Get the data factory and data serving system manager instance and if data serving system manager is ready

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

    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);
```

#### 4. Register for serving system listener

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

#### 5. 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);
```

Now, wait for request response and notifications.

### 3.4.17 Get roaming status and indication

This sample application demonstrates how to get roaming status and indications.

#### 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. Optionally, instantiate an initialization callback

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

### 3. Get the data factory and data serving system manager instance and if data serving system manager is ready

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

### 4. 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);
```

Now, wait for request response and notifications.

### 3.4.18 Establish on-demand PDN connectivity

For on-demand PDNs connectivity:

- The DNS servers are not updated in resolv.conf and routines like gethostbyname would fail
- The default routes are not setup and socket connection without binding to an interface would fail The purpose of the section is to provide information about:
- DNS resolution using dig
- Binding to an interface to enable connectivity

#### 1. Get Data Connection Manager and wait for service availability

Get the data factory, data connection manager and wait until the service is available.

```
auto &dataFactory = telux::data::DataFactory::getInstance();
{
     std::promise<telux::common::ServiceStatus> p;
     dataConnMgr = dataFactory.getDataConnectionManager(
         slotId, [&](telux::common::ServiceStatus status) { p.set_value(status); });
     if (dataConnMgr) {
         std::cout << "\n\nInitializing Data connection manager subsystem on slot " << slotId
                   << ", Please wait ..." << std::endl;
         subSystemStatus = p.get_future().get();
     }
     if (subSystemStatus == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
         std::cout << "Data sub-system is ready" << std::endl;
     } else {
         std::cerr << "Unable to initialize data subsystem. Exiting..." << std::endl;
         exit(1);
     }
}
```

#### 2. Register listener

Register the listener with the data connection manager for listening to data call status.

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

#### 3. Start data call on the mentioned slot Id and profile id and operation type

```
{
 std::promise<telux::common::ErrorCode> p;
 telux::data::IpFamilyType ipFamilyType = telux::data::IpFamilyType::IPV4;
 dataConnMgr->startDataCall(
        profileId, ipFamilyType,
        [&](const std::shared_ptr<telux::data::IDataCall> &dataCall,
            telux::common::ErrorCode errorCode) {
            std::cout << "startCallResponse: errorCode: " << static_cast<int>(errorCode)
                      << std::endl;
            p.set_value(errorCode);
        },
        opType);

 telux::common::ErrorCode errorCode = p.get_future().get();
 if (errorCode != telux::common::ErrorCode::SUCCESS) {
     std::cerr << "Failed to start data call. Exiting..." << std::endl;
     exit(1);
 }
}
```

#### 4. Wait for the data call to get connected

When the data call is connected, obtain a reference to the data call that was brought up. This information would be sent to the listener.

```
// In the listener
void onDataCallInfoChanged(const std::shared_ptr<telux::data::IDataCall> &dataCall) override {
     std::cout << "\n onDataCallInfoChanged";
     logDataCallDetails(dataCall);
     if (dataCall->getDataCallStatus() == telux::data::DataCallStatus::NET_CONNECTED) {
         p_.set_value(dataCall);
     }
}

// In the main method
std::shared_ptr<telux::data::IDataCall> dataCall = dataCallFuture.get();
if (dataCall == nullptr) {
     std::cerr << "Could not get data call object. Exiting..." << std::endl;
     exit(1);
}
```

#### 5. Resolve the remote host using the DNS address provided by the data call

Use the DNS address (Ex: primaryDnsAddress) information from the data call object to resolve the domain using dig (Domain Information Groper). dig has multiple modes and accepts a variety of parameters, however, for the purpose of name resolution, we use the "+short" mode where detailed answers are not output by dig, but only the IP addresses are provided. We parse the IP addresses provided by dig to see if it is a valid IP address.

```
// The resolve method
std::string resolve(std::string domain, std::string dnsAddress) {
     std::cout << "Resolving " << domain << " using DNS server at " << dnsAddress << std::endl;
     FILE *cmd;
     std::string ipAddress;
     char cipAddress[SIZE_IP_ADDR_BUF] = {0};

     // Use the provided DNS address to request dig for name resolution
     std::string command = "/usr/bin/dig @" + dnsAddress + " " + domain + " +short";
     std::cout << "Command: " << command << std::endl;
     cmd = popen(command.c_str(), "r");
     if (cmd) {
         sockaddr_in address;
         // Get all the answers from the DNS server
         while (NULL != fgets(cipAddress, SIZE_IP_ADDR_BUF, cmd)) {
             cipAddress[strlen(cipAddress) - 1] = '\0';

             // If the received answer is verified as a valid IP address, return the address
             if (inet_pton(AF_INET, cipAddress, &address.sin_addr) == 1) {
                 ipAddress = cipAddress;
                 std::cout << ipAddress;
                 ipAddress.erase(
                     std::remove(ipAddress.begin(), ipAddress.end(), '\n'), ipAddress.end());
                 break;
             }
         }
     }
     std::cout << "\n\n";  // Declutters output from dig
     return ipAddress;
}

// In the main method
std::string remoteIp = resolve(domain, dataCall->getIpv4Info().addr.primaryDnsAddress);
if (remoteIp == "") {
    std::cerr << "Could not resolve " << domain << ". Exiting..." << std::endl;
    exit(1);
}
std::cout << "Resolved " << domain << " to " << remoteIp << std::endl;
```

#### 6. Connect to the remote host

In addition to the usual connection routine on the client side, we additionally would need to bind to the interface that would allow us to reach the remote host. This is done here by using the setsockopt method.

```
// The connect method
 int connect(std::string ipAddress, std::string outBoundIf, std::string portNumber) {
     std::cout << "Connecting to " << ipAddress << " on port " << portNumber << " via " << outBoundIf
             << std::endl;
     int sockfd = 0;
     sockaddr_in serverIpAddress;
     serverIpAddress.sin_family = AF_INET;
     serverIpAddress.sin_port = htons(stoi(portNumber));

     if (inet_pton(AF_INET, ipAddress.c_str(), &serverIpAddress.sin_addr) >= 0) {
         std::cerr << "Cannot parse IP address" << std::endl;
         return -1;
     }
     // Create the socket
     if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
         std::cerr << "Socket creation failed" << std::endl;
         return -1;
     }

     // Bind the socket to the interface
     ifreq ifr;
     g_strlcpy(ifr.ifr_name, outBoundIf.c_str(), outBoundIf.length());
     if (setsockopt(sockfd, SOL_SOCKET, SO_BINDTODEVICE, (void *)&ifr, sizeof(ifr)) < 0) {
         std::cerr << "Socket bind failed with " << strerror(errno) << std::endl;
         close(sockfd);
         return -1;
     }

     // Connect to the remote host
     if (connect(sockfd, (sockaddr *)&serverIpAddress, sizeof(serverIpAddress)) < 0) {
         std::cerr << "Connect failed: " << strerror(errno) << std::endl;
         close(sockfd);
         return -1;
     }
     return sockfd;
 }

 // In the main method
 int sockfd = connect(remoteIp, dataCall->getInterfaceName(), portNumber);
 if (sockfd < 0) {
     std::cerr << "Could not connect to " << domain << ". Exiting..." << std::endl;
     exit(1);
 }
 std::cout << "Connected to " << domain << std::endl;
```

#### 7. Clean-up

```
std::cout << "Cleaning up" << std::endl;
close(sockfd);
dataConnMgr->deregisterListener(dataListener);
dataConnMgr = nullptr;
```

### 3.4.19 Create traffic class and add QoS filter

Example use cases as per different data paths in system:

- 1. VLAN-based downlink traffic, tethered to apps software path
- 2. VLAN-based uplink traffic, tethered to the apps software path
- 3. IPv4-based downlink traffic, tethered to the WAN hardware accelerated path

- 4. IPv4-based uplink traffic, tethered to the WAN hardware accelerated path
- 5. IPv4-based uplink traffic, from apps to the WAN path

### 1. VLAN-based downlink traffic, tethered to apps software path (ETH <=> Apps):

#### 1.1 Get VLAN and QoS Manager and wait for service availability

- Get the data factory, VLAN and QoS manager and wait until the service is available.

```
std::promise<telux::common::ServiceStatus> vlanProm;
// VLAN Manager instance
vlanManager_  = telux::data::DataFactory::getInstance().getVlanManager(
    telux::data::OperationType::DATA_LOCAL,
    [&vlanProm](telux::common::ServiceStatus status) {
    vlanProm.set_value(status);
    });

if (!vlanManager_) {
    std::cout <<  " Failed to get VLAN Manager object" << std::endl;;
    return false;
}

telux::common::ServiceStatus subSystemStatus = vlanProm.get_future().get();
if (subSystemStatus == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << " *** VLAN manager is Ready *** " << std::endl;
} else {
    std::cout << " *** Unable to initialize VLAN subsystem *** " << std::endl;
    return false;
}

//Register for listener
vlanManager_->registerListener(vlanListener);
```

```
std::promise<telux::common::ServiceStatus> qosProm;

// QoS Manager instance
dataQoSManager_  = telux::data::DataFactory::getInstance().getQoSManager([&qosProm]
    (telux::common::ServiceStatus status) {
    qosProm.set_value(status);
    });

if (!dataQoSManager_) {
    std::cout << " Failed to get DataQoSManager object" << std::endl;
}

telux::common::ServiceStatus subSystemStatus = qosProm.get_future().get();
if (subSystemStatus == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << " *** QoS manager is Ready *** " << std::endl;
} else {
    std::cout << " *** Unable to initialize QoS subsystem *** " << std::endl;
}

//Register for listener
dataQoSManager_->registerListener(qosListener);
```

#### 1.2. Create and wait for VLAN

- Create VLAN with below attributes
  - ID = 20
  - HW Acceleration = False
  - PCP = 7
- Note: Creation VLAN at first time might lead to device restart

```
std::promise<telux::common::ErrorCode> vlanProm;
auto respCb = [&vlanProm, &vlanId, &pcp](bool isAccelerated, telux::common::ErrorCode error) {
    std::cout << std::endl << std::endl;
    std::cout << "VLAN ID: "  << vlanId << ", PCP: "<< pcp <<", IPA HW acceleration: "
            << (isAccelerated ? " enabled" : " not enabled") << std::endl;
    std::cout << "CALLBACK: "
            << "createVlan Response"
            << (error == telux::common::ErrorCode::SUCCESS ? " is successful" : " failed")
            << ". ErrorCode: " << static_cast<int>(error) << std::endl;
    vlanProm.set_value(error);
};

telux::data::VlanConfig config;
config.iface = telux::data::InterfaceType::ETH;
config.vlanId = vlanId;
config.priority = pcp;
config.isAccelerated = isAccelerated;

telux::common::Status status = vlanManager_->createVlan(config, respCb);
if ((status == telux::common::Status::SUCCESS) &&
    (vlanProm.get_future().get() == telux::common::ErrorCode::SUCCESS)) {
    return true;
}
```

### 1.3. Create traffic class

- Create traffic class
  - TC ID = 0
  - Data path = TETHERED_TO_APPS_SW
  - BW Config {min = 5Mbps, max = 10Mbps}
  - Direction = DOWNLINK
- Note: The traffic class is uniquely identified using a combination of the traffic class ID and direction. i.e., it can be the same across different traffic classes.

```
telux::data::net::BandwidthConfig bandwidthConfig;
bandwidthConfig.setDlBandwidthRange(minBandwidth, maxBandwidth);

telux::data::net::TcConfigBuilder tcConfigBuilder;
tcConfigBuilder.setTrafficClass(trafficClass).
    setDirection(telux::data::Direction::DOWNLINK).
    setDataPath(dataPath).
    setBandwidthConfig(bandwidthConfig);

telux::data::net::TcConfigErrorCode tcConfigErrorCode;
telux::common::ErrorCode errorCode
    = dataQoSManager_->createTrafficClass(tcConfigBuilder.build(), tcConfigErrorCode);
if (errorCode == telux::common::ErrorCode::SUCCESS) {
    std::cout << " Create traffic class is successful." << std::endl;
    trafficClass_.push_back(trafficClass);
} else {
    std::cout << " Create traffic class is failed. ErrorCode: " << static_cast<int>(errorCode)
              << " " << static_cast<int>(tcConfigErrorCode) << std::endl;
    return false;
}
```

### 1.4. Add VLAN based QoS filter

- Add VLAN based QoS filter with following parameter:
  - TC ID = 0
  - Data path = TETHERED_TO_APPS_SW
  - Direction = DOWNLINK
  - PCP = 7 Note: PCP is not a mandatory parameter to provide, but it is recommended to provide if available.
  - VLAN IDs = [20] Note: Instead of VLAN, a 5-tuple (i.e., L3, L4 parameters) can be provided.
- When a QoS filter is added successfully, a policy handle is provided as an output parameter.

```
telux::data::TrafficFilterBuilder tfBuilder;
// In the downlink direction, the field type of VLAN is expected to be of the destination.
tfBuilder.setDirection(direction).setVlanList({vlanId}, telux::data::FieldType::DESTINATION).
    setDataPath(dataPath).setPCP(pcp);

// Configure QoS filter
telux::data::net::QoSFilterConfig qosFilterConfig = {0};
// traffic class
qosFilterConfig.trafficClass = trafficClass;
// traffic filter
qosFilterConfig.trafficFilter = tfBuilder.build();

// Add QoS filter
uint32_t policyHandle;
telux::data::net::QoSFilterErrorCode qosFilterErrorCode;
telux::common::ErrorCode errorCode =
    dataQoSManager_->addQoSFilter(qosFilterConfig, policyHandle, qosFilterErrorCode);
if (errorCode == telux::common::ErrorCode::SUCCESS) {
    std::cout << " Add QoS filter is successful. Handle of the QoS filter = " << policyHandle
              << std::endl;
    qosFilterHandles_.push_back(policyHandle);
} else {
    std::cout << " Add QoS filter is failed. ErrorCode: " << static_cast<int>(errorCode)
              << " "  << static_cast<int>(qosFilterErrorCode) << std::endl;
    return 0;
}
```

### 2. VLAN-based uplink traffic, tethered to the apps software path (ETH <=> Apps):

### 2.1 Get VLAN and QoS Manager and wait for service availability

- Get the data factory, VLAN and QoS manager and wait until the service is available.
- Step same as 1.1.

### 2.2. Create and wait for VLAN

- Create VLAN with below attributes
  - ID = 19
  - HW Acceleration = False
  - PCP = 7
- Parameters are different, but the step is similar to 1.2.

### 2.3. Create traffic class

- Create traffic class
  - TC ID = 0
  - Data path = TETHERED_TO_APPS_SW (ETH <=> Apps)
  - Direction = UPLINK
- Note: Bandwidth configuration/traffic shaping in the uplink direction is not supported.

```
telux::data::net::TcConfigBuilder tcConfigBuilder;
tcConfigBuilder.setTrafficClass(trafficClass).
    setDirection(telux::data::Direction::UPLINK).
    setDataPath(dataPath);

telux::data::net::TcConfigErrorCode tcConfigErrorCode;
telux::common::ErrorCode errorCode
    = dataQoSManager_->createTrafficClass(tcConfigBuilder.build(), tcConfigErrorCode);
if (errorCode == telux::common::ErrorCode::SUCCESS) {
    std::cout << " Create traffic class is successful." << std::endl;
    trafficClass_.push_back(trafficClass);
} else {
    std::cout << " Create traffic class is failed. ErrorCode: " << static_cast<int>(errorCode)
                << " " << static_cast<int>(tcConfigErrorCode) << std::endl;
    return false;
}
```

### 2.4. Add VLAN based QoS filter

- Add VLAN based QoS filter with following parameter:
  - TC ID = 0
  - Data path = TETHERED_TO_APPS_SW (ETH <=> Apps)
  - Direction = UPLINK
  - PCP = 7 Note: It is not mandatory, but if available, providing PCP in the uplink direction for data paths involving the ETH module is beneficial.
  - VLAN IDs = [19] Note: Instead of VLAN, a 5-tuple (i.e., L3, L4 parameters) can be provided.
- When a QoS filter is added successfully, a policy handle is provided as an output parameter.

```
telux::data::TrafficFilterBuilder tfBuilder;
// In the downlink direction, the field type of VLAN is expected to be of the destination.
// For uplink, it is supposed to be the source.
tfBuilder.setDirection(direction).setVlanList({vlanId}, telux::data::FieldType::SOURCE).
    setDataPath(dataPath).setPCP(pcp);

// Configure QoS filter
telux::data::net::QoSFilterConfig qosFilterConfig = {0};
// traffic class
qosFilterConfig.trafficClass = trafficClass;
// traffic filter
qosFilterConfig.trafficFilter = tfBuilder.build();

// Add QoS filter
uint32_t policyHandle;
telux::data::net::QoSFilterErrorCode qosFilterErrorCode;
telux::common::ErrorCode errorCode =
    dataQoSManager_->addQoSFilter(qosFilterConfig, policyHandle, qosFilterErrorCode);
if (errorCode == telux::common::ErrorCode::SUCCESS) {
    std::cout << " Add QoS filter is successful. Handle of the QoS filter = " << policyHandle
                << std::endl;
    qosFilterHandles_.push_back(policyHandle);
```

```
} else {
    std::cout << " Add QoS filter is failed. ErrorCode: " << static_cast<int>(errorCode)
                << " "  << static_cast<int>(qosFilterErrorCode) << std::endl;
    return 0;
}
```

### 3. IPv4-based downlink traffic, tethered to the WAN hardware accelerated path (ETH <=> IPA <=> Modem):

#### 3.1 Get VLAN, connection and QoS Manager and wait for service availability

- Get the data factory, VLAN, connection and QoS manager and wait until the service is available.
- In addition to 1.1, we will also need a data connection manager for some steps in this use case.
- The steps to get the data connection manager are as follows:

```
// data connection mananger
dataConnectionManager_ =
    telux::data::DataFactory::getInstance().getDataConnectionManager(slotId,
                                        [&dcmProm](telux::common::ServiceStatus status)
                                        { dcmProm.set_value(status); });

if (!dataConnectionManager_) {
    std::cout << " Failed to get DataConnectionManager object" << std::endl;
    return false;
}

//wait for connection manager to get ready
std::cout << " Initializing Data connection manager subsystem Please wait" << std::endl;
telux::common::ServiceStatus subSystemStatus = dcmProm.get_future().get();

if (subSystemStatus == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
    std::shared_ptr<telux::data::IDataConnectionListener> dcmListener =  shared_from_this();
    std::cout << " *** Data Connection Manager is ready *** " << std::endl;
    telux::common::Status status =
    dataConnectionManager_->registerListener(dcmListener);

    if (status != telux::common::Status::SUCCESS) {
    std::cout << " Unable to register data connection manager listener" << std::endl;
    return false;
    }
} else {
    std::cout << " Data Connection Manager is failed" << std::endl;
    dataConnectionManager_ = nullptr;
    return false;
}
```

#### 3.2. Create and wait for VLAN

- Create VLAN with below attributes
  - ID = 18
  - HW Acceleration = True
  - PCP = 6
- Parameters are different, but the step is similar to 1.2.

#### 3.3. Bind VLAN with Backhaul

- Bind VLAN-18 to WWAN default Backhaul ( for example slotId = 1, profileId = 1)

```
std::promise<telux::common::ErrorCode> vlanProm;
telux::data::net::VlanBindConfig vlanBindConfig = {};
```

```
vlanBindConfig.bhInfo.backhaul = telux::data::BackhaulType::WWAN;
vlanBindConfig.bhInfo.profileId = profileId_;
vlanBindConfig.bhInfo.slotId = slotId_;
vlanBindConfig.vlanId = vlanId;

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

telux::common::Status status = vlanManager_->bindToBackhaul(vlanBindConfig, respCb);
if ((status == telux::common::Status::SUCCESS) &&
        (vlanProm.get_future().get() == telux::common::ErrorCode::SUCCESS)) {
    return true;
}
```

### 3.4. Bring-up data call

- The data call is expected to be brought up on the same profile to successfully add a filter in the modem.

```
void logDataCallDetails(const std::shared_ptr<telux::data::IDataCall> &dataCall) {
    std::cout << " ** DataCall Details **\n";
    std::cout << " SlotID: " << dataCall->getSlotId() << std::endl;
    std::cout << " ProfileID: " << dataCall->getProfileId() << std::endl;
    std::cout << " interfaceName: " << dataCall->getInterfaceName() << std::endl;
    std::cout << " DataCallStatus: " << (int)dataCall->getDataCallStatus() << std::endl;
    std::cout
        << " DataCallEndReason: Type = " << static_cast<int>(dataCall->getDataCallEndReason().type)
        << std::endl;
    std::list<telux::data::IpAddrInfo> ipAddrList = dataCall->getIpAddressInfo();
    if(dataCall->getDataCallStatus() == telux::data::DataCallStatus::NET_CONNECTED) {

        // Check if the data call on the expected slot and profile is up.
        if((dataCall->getSlotId() == slotId_) && (dataCall->getProfileId() == profileId_)) {
            std::cout << " ** Active data call on default profile **\n";
            std::lock_guard<std::mutex> lock(dataCallMtx_);
            rmnetIp_ = ipAddrList.begin()->ifAddress;
            dataCallCv_.notify_all();
        }
        for(auto &it : ipAddrList) {
            std::cout << "\n ifAddress: " << it.ifAddress
                    << "\n primaryDnsAddress: " << it.primaryDnsAddress
                    << "\n secondaryDnsAddress: " << it.secondaryDnsAddress << '\n';
        }
        std::cout << "IpFamilyType: " << static_cast<int>(dataCall->getIpFamilyType()) << '\n';
        std::cout << "TechPreference: " << static_cast<int>(dataCall->getTechPreference()) << '\n';
    }
}

// This is a listener API in the data connection manager listener, which is invoked in changes to the
       information of a data call.
void onDataCallInfoChanged(
    const std::shared_ptr<telux::data::IDataCall> &dataCall) {
    std::cout << "\n onDataCallInfoChanged";
    logDataCallDetails(dataCall);
}

// To avoid issue in start data call Before staring data call make sure APN is update signal strength and
       RAT band are selected as expected etc.
// steps for this is not part of this sample app.
telux::common::Status status =
    dataConnectionManager_->startDataCall(profileId_, ipFamilyType,
    [this](const std::shared_ptr<telux::data::IDataCall> &dataCall,
        telux::common::ErrorCode errorCode) {
```

```
            std::cout << "startCallResponse: errorCode: "
            << static_cast<int>(errorCode) << std::endl;

        // Check if the data call is already available, or else wait for the indication
       'onDataCallInfoChanged'.
        logDataCallDetails(dataCall);
    }, telux::data::OperationType::DATA_LOCAL);

    if(status == telux::common::Status::SUCCESS) {
        std::unique_lock<std::mutex> lck(dataCallMtx_);
        dataCallCv_.wait(lck, [&]{return !rmnetIp_.empty();});
        return true;
    }
```

### 3.5. Create traffic class

- Parameters are different, but the step is similar to 1.3.
- Create traffic class
  - TC ID = 1
  - BW Config {min = 5Mbps, max = 10Mbps}
  - Data path = TETHERED_TO_WAN_HW (ETH <=> IPA <=> Modem)
  - Direction = DOWNLINK
- Parameters are different, but the step is similar to 1.3.

### 3.6. Add IP based QoS filter

- Add IP based QoS filter
  - TC ID = 1
  - Data path = TETHERED_TO_WAN_HW (ETH <=> IPA <=> Modem) Note: Prioritization in the modem happens based on a 5-tuple (i.e., L3, L4 parameters).
  - Direction = DOWNLINK
  - Source IP = Remote server Note: An IP-based filter involving the modem needs the source IP, protocol, and (destination port or destination IP) as mandatory parameters.
  - Destination port = 30044
  - Protocol = TCP (6 as per IANA)
  - Source port = 8080
- When a QoS filter is added successfully, a policy handle is provided as an output parameter.

```
telux::data::TrafficFilterBuilder tfBuilder;
tfBuilder.setDirection(direction).
    setIPv4Address(srcIPv4, telux::data::FieldType::SOURCE).
    setIPProtocol(protocol).
    setDataPath(dataPath).
    setPort(destPort, telux::data::FieldType::DESTINATION).
    setPort(srcPort, telux::data::FieldType::SOURCE);

// Configure QoS filter
telux::data::net::QoSFilterConfig qosFilterConfig = {0};
// traffic class
qosFilterConfig.trafficClass = trafficClass;
// traffic filter
qosFilterConfig.trafficFilter = tfBuilder.build();
```

```
// Add QoS filter
uint32_t policyHandle;
telux::data::net::QoSFilterErrorCode qosFilterErrorCode;
telux::common::ErrorCode errorCode =
    dataQoSManager_->addQoSFilter(qosFilterConfig, policyHandle, qosFilterErrorCode);
if (errorCode == telux::common::ErrorCode::SUCCESS) {
    std::cout << " Add QoS filter is successful. Handle of the QoS filter = " << policyHandle
              << std::endl;
    qosFilterHandles_.push_back(policyHandle);
    return policyHandle;
} else {
    std::cout << " Add QoS filter is failed. ErrorCode: " << static_cast<int>(errorCode)
              << " "  << static_cast<int>(qosFilterErrorCode) << std::endl;
    return 0;
}
```

**4. IPv4-based uplink traffic, tethered to the WAN hardware accelerated path (ETH <=> IPA <=> Modem):**

**4.1 Get VLAN, connection and QoS Manager and wait for service availability**

- Get the data factory, VLAN, connection and QoS manager and wait until the service is available.
- Step same as 3.1.

**4.2. Create and wait for VLAN**

- Create VLAN with below attributes
  - ID = 18
  - HW Acceleration = True
  - PCP = 6
- Parameters are different, but the step is similar to 1.2.

**4.3. Bind VLAN with Backhaul**

- Bind VLAN-18 to WWAN default Backhaul ( for example slotId = 1, profileId = 1)
- Step same as 3.3.

**4.4. Bring-up data call**

- The data call is expected to be brought up on the same profile to successfully add a filter in the modem.
- Step same as 3.4.

**4.5. Create traffic class**

- Create traffic class
  - TC ID = 2
  - Data path = TETHERED_TO_WAN_HW
  - Direction = UPLINK
- Parameters are different, but the step is similar to 2.3.

### 4.6. Add IP based QoS filter

- Add IP based QoS filter
  - TC ID = 1
  - Data path = TETHERED_TO_WAN_HW (ETH <=> IPA <=> Modem) Note: Prioritization in the modem happens based on a 5-tuple (i.e., L3, L4 parameters).
  - Direction = UPLINK
  - Source IP = <Public ip>=""> from IDataCall object (from step 4.4.)
  - Note: An IP-based filter involving the modem needs the source IP, protocol, and (destination port or destination IP) as mandatory parameters.
  - Destination port = 8081
  - Protocol = TCP (6 as per IANA)
  - PCP = 6 Note: It is not mandatory, but if available, providing PCP in the uplink direction for data paths involving the ETH module is beneficial.
- When a QoS filter is added successfully, a policy handle is provided as an output parameter.

```
telux::data::TrafficFilterBuilder tfBuilder;
// The source IP in the uplink filter is expected to be the public IP of the data call from step 4.4.
tfBuilder.setDirection(direction).
    setIPv4Address(srcIPv4, telux::data::FieldType::SOURCE).
    setIPProtocol(protocol).
    setDataPath(dataPath).
    setPort(destPort, telux::data::FieldType::DESTINATION)
    setPCP(6);

// Configure QoS filter
telux::data::net::QoSFilterConfig qosFilterConfig = {0};
// traffic class
qosFilterConfig.trafficClass = trafficClass;
// traffic filter
qosFilterConfig.trafficFilter = tfBuilder.build();

// Add QoS filter
uint32_t policyHandle;
telux::data::net::QoSFilterErrorCode qosFilterErrorCode;
telux::common::ErrorCode errorCode =
    dataQoSManager_->addQoSFilter(qosFilterConfig, policyHandle, qosFilterErrorCode);
if (errorCode == telux::common::ErrorCode::SUCCESS) {
    std::cout << " Add QoS filter is successful. Handle of the QoS filter = " << policyHandle
              << std::endl;
    qosFilterHandles_.push_back(policyHandle);
    return policyHandle;
} else {
    std::cout << " Add QoS filter is failed. ErrorCode: " << static_cast<int>(errorCode)
              << " "  << static_cast<int>(qosFilterErrorCode) << std::endl;
    return 0;
}
```

### 5. IPv4-based uplink traffic, from apps to the WAN path (Apps <=> Modem):

### 5.1 Data connection and QoS Manager and wait for service availability

- Get the data factory, data connection and QoS manager and wait until the service is available.

```
std::promise<telux::common::ServiceStatus> qosProm;

// QoS Manager instance
dataQoSManager_  = telux::data::DataFactory::getInstance().getQoSManager([&qosProm]
```

```
            (telux::common::ServiceStatus status) {
            qosProm.set_value(status);
            });

        if (!dataQoSManager_) {
            std::cout << " Failed to get DataQoSManager object" << std::endl;
        }

        telux::common::ServiceStatus subSystemStatus = qosProm.get_future().get();
        if (subSystemStatus == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
            std::cout << " *** QoS manager is Ready *** " << std::endl;
        } else {
            std::cout << " *** Unable to initialize QoS subsystem *** " << std::endl;
        }

        //Register for listener
        dataQoSManager_->registerListener(qosListener);
```

```
std::promise<telux::common::ServiceStatus> dcmProm;
SlotId slotId = DEFAULT_SLOT_ID;
// data connection mananger
dataConnectionManager_ =
    telux::data::DataFactory::getInstance().getDataConnectionManager(slotId,
                                        [&dcmProm](telux::common::ServiceStatus status)
                                        { dcmProm.set_value(status); });

if (!dataConnectionManager_) {
    std::cout << " Failed to get DataConnectionManager object" << std::endl;
    return false;
}

//wait for connection manager to get ready
std::cout << " Initializing Data connection manager subsystem Please wait" << std::endl;
telux::common::ServiceStatus subSystemStatus = dcmProm.get_future().get();

if (subSystemStatus == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
    std::shared_ptr<telux::data::IDataConnectionListener> dcmListener =  shared_from_this();
    std::cout << " *** Data Connection Manager is ready *** " << std::endl;
    telux::common::Status status =
    dataConnectionManager_->registerListener(dcmListener);

    if (status != telux::common::Status::SUCCESS) {
    std::cout << " Unable to register data connection manager listener" << std::endl;
    return false;
    }
} else {
    std::cout << " Data Connection Manager is failed" << std::endl;
    dataConnectionManager_ = nullptr;
    return false;
}
```

### 5.2. Bring-up data call

- The data call is expected to be brought up on the same profile to successfully add a filter in the modem.
- Step same as 3.4.

### 5.3. Create traffic class

- Create traffic class
  - TC ID = 3
  - Data path = APPS_TO_WAN (Apps <=> Modem)
  - Direction = UPLINK

- Parameters are different, but the step is similar to 2.3.

### 5.4. Add IP based QoS filter

- Add IP based QoS filter
  - TC ID = 3
  - Data path = APPS_TO_WAN (Apps <=> Modem) Note: Prioritization in the modem happens based on a 5-tuple (i.e., L3, L4 parameters).
  - Direction = UPLINK
  - Source IP = <Public ip>=""> from IDataCall object (from 5.2) Note: An IP-based filter involving the modem needs the source IP, protocol, and (destination port or destination IP) as mandatory parameters.
  - Destination port = 8080
  - Protocol = UPD (17 as per IANA)
- When a QoS filter is added successfully, a policy handle is provided as an output parameter.

```
telux::data::TrafficFilterBuilder tfBuilder;
tfBuilder.setDirection(direction).
    setIPv4Address(srcIPv4, telux::data::FieldType::SOURCE).
    setIPProtocol(protocol).
    setDataPath(dataPath).
    setPort(destPort, telux::data::FieldType::DESTINATION);

// Configure QoS filter
telux::data::net::QoSFilterConfig qosFilterConfig = {0};
// traffic class
qosFilterConfig.trafficClass = trafficClass;
// traffic filter
qosFilterConfig.trafficFilter = tfBuilder.build();

// Add QoS filter
uint32_t policyHandle;
telux::data::net::QoSFilterErrorCode qosFilterErrorCode;
telux::common::ErrorCode errorCode =
    dataQoSManager_->addQoSFilter(qosFilterConfig, policyHandle, qosFilterErrorCode);
if (errorCode == telux::common::ErrorCode::SUCCESS) {
    std::cout << " Add QoS filter is successful. Handle of the QoS filter = " << policyHandle
              << std::endl;
    qosFilterHandles_.push_back(policyHandle);
    return policyHandle;
} else {
    std::cout << " Add QoS filter is failed. ErrorCode: " << static_cast<int>(errorCode)
              << " "  << static_cast<int>(qosFilterErrorCode) << std::endl;
    return 0;
}
```

## 3.5 Location

- Location, SV and Jammer reports
- Using location configurator APIs

### 3.5.1 Location, SV and Jammer reports

This sample app demonstrates how to get location, satellite vehicle and jammer info reports.

#### 1. Implement a command response method

```
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.

// CmdResponse callback is invoked with error code indicating
// SUCCESS/FAILURE of the operation
locationManager_->startDetailedReports(minIntervalInput, CmdResponse);
```

#### 10. 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;
}
```

Now 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.

### 3.5.2 Using location configurator APIs

This sample app gives basic idea about using location configurator APIs.

#### 1. Implement a command response method

```
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

```
// CmdResponse callback is invoked with error code indicating
// SUCCESS/FAILURE of the operation
locConfigurator_->configureCTunc(enable, CmdResponse, optThreshold, optPower);
```

## 3.6 Modem configuration

- Load and activate modem configuration
- Enable/Disable auto selection mode
- Deactivate/Delete modem configuration file

### 3.6.1 Load and activate modem configuration

This sample app demonstrates how to load and activate modem configuration.

#### 1. Get the ConfigFactory instance

```
auto &configFactory = telux::config::ConfigFactory::getInstance();
```

#### 2. Get ModemConfigManager instance and wait for sub system initialization

```
std::promise<telux::common::ServiceStatus> prom{};
modemConfigManager_ = configFactory.getModemConfigManager(
     [&prom](telux::common::ServiceStatus status) {
         prom.set_value(status);
});

if (!modemConfigManager_) {
    std::cout << "Failed to get modem config Manager" << std::endl;
    return;
}

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

 // Exit the application, if TelSDK is unable to initialize modem config subsystem
```

```
 if (managerStatus != telux::common::ServiceStatus::SERVICE_AVAILABLE) {
     std::cout << "ERROR - Unable to initialize subsystem" << std::endl;
     return;
 }
```

### 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 configuration 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 configuration

```
// 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 modem configuration" << 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;
}
```

### 3.6.2 Enable/Disable auto selection mode

This sample app demonstrates how to enable/disable auto selection mode for modem configuration.

#### 1. Get the ConfigFactory instance

```
auto &configFactory = telux::config::ConfigFactory::getInstance();
```

#### 2. Get ModemConfigManager instance and Wait for sub system initialization

```
std::promise<telux::common::ServiceStatus> prom{};
modemConfigManager_ = configFactory.getModemConfigManager(
     [&prom](telux::common::ServiceStatus status) {
         prom.set_value(status);
});

if (!modemConfigManager_) {
    std::cout << "Failed to get modem config Manager" << std::endl;
    return;
}

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

// Exit the application, if SDK is unable to initialize modem config subsystems
if (managerStatus != telux::common::ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "ERROR - Unable to initialize subSystem" << std::endl;
    return;
}
```

#### 3. Set auto configuration 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 current auto configuration 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;
    }
}
```

### 3.6.3 Deactivate/Delete modem configuration file

This sample app demonstrates how to deactivate and delete modem configuration file.

#### 1. Get the ConfigFactory instance

```
auto &configFactory = telux::config::ConfigFactory::getInstance();
```

#### 2. Get ModemConfigManager instance and Wait for sub system initialization

```
std::promise<telux::common::ServiceStatus> prom{};
modemConfigManager_ = configFactory.getModemConfigManager(
     [&prom](telux::common::ServiceStatus status) {
         prom.set_value(status);
});

if (!modemConfigManager_) {
    std::cout << "Failed to get modem config Manager" << std::endl;
    return;
}

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

// Exit the application, if SDK is unable to initialize modem config subsystems
```

```
if (managerStatus != telux::common::ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << "ERROR - Unable to initialize subSystem" << std::endl;
    return;
}
```

### 3. Deactivate modem configuration

```
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 configuration 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 modem configuration 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;
}
```

# 3.7 Power

- Get TCU power state updates
- Set TCU activity state

## 3.7.1 Get TCU power state updates

This sample application demonstrates how to register for TCU-activity state notifications on the local machine, for performing any tasks before the state transition.

#### 1. Implement ITcuActivityListener and IServiceStatusListener interface

```
class MyTcuActivityStateListener : public ITcuActivityListener,
                                   public IServiceStatusListener {
public:
    void onTcuActivityStateUpdate(TcuActivityState state, std::string machineName) override;
    void onServiceStatusChange(ServiceStatus status) override;
};
```

#### 2. Get an instance of PowerFactory, and then obtain a TCU-activity manager instance with clientType as SLAVE and machine name as LOCAL_MACHINE

```
auto &powerFactory = PowerFactory::getInstance();
std::promise<telux::common::ServiceStatus> prom = std::promise<telux::common::ServiceStatus>();
ClientInstanceConfig config;
config.clientType = ClientType::SLAVE;
config.clientName = "client_name_" + std::to_string(getpid());
config.machineName =  LOCAL_MACHINE;
auto tcuActivityManager = powerFactory.getTcuActivityManager(config,
                              [&](telux::common::ServiceStatus status) {
                                 std::cout << " Init Callback called " << std::endl;
                                 prom.set_value(status);
                              });
```

#### 3. Wait for the TCU-activity management services to be initialized and ready

```
if (tcuActivityStateMgr_ == nullptr) {
   std::cout << " ERROR - Failed to get manager instance" << std::endl;
}
std::cout << "  Waiting for TCU Activity Manager to be ready" << std::endl;
telux::common::ServiceStatus serviceStatus = prom.get_future().get();
```

#### 4. Exit the application, if SDK is unable to initialize TCU-activity management service

```
if (serviceStatus == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   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;
```

```
}
```

#### 5. Instantiate MyTcuActivityStateListener and register for updates on TCU-activity state and its management service status

```
auto myTcuStateListener = std::make_shared<MyTcuActivityStateListener>();
tcuActivityManager->registerListener(myTcuStateListener);
tcuActivityManager->registerServiceStateListener(myTcuStateListener);
```

#### 6. When the TCU activity state of the local machine is about to change, the below listener callback is invoked with the new activity state

```
void MyTcuActivityStateListener::onTcuActivityStateUpdate(TcuActivityState tcuState, std::string
      machineName) {
     std::cout << std::endl << "********* TCU-activity state update *********" << std::endl;
     // Avoid long blocking calls when handling notifications
     // Perform necessary tasks and prepare for the state transition
}
```

#### 7. After successfully processing the SUSPEND/SHUTDOWN notification, send an acknowledgement to convey the readiness of the application for the state transition

```
tcuActivityManager->sendActivityStateAck(StateChangeResponse::ACK, tcuState);
```

#### 8. Implement onServiceStatusChange callback to know when TCU-activity management service goes down

```
// 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
}
```

### 3.7.2 Set TCU activity state

This sample application demonstrates how to change the TCU activity state of all the application processors (machines) in the system

#### 1. Implement a command response callback method

```
// This is invoked with error code indicating whether the request
// was SUCCESS or FAILURE
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 onMachineUpdate(const std::string machineName, const MachineEvent machineEvent);
   void onSlaveAckStatusUpdate(const telux::common::Status status,
      const std::string machineName, const std::vector<ClientInfo> unresponsiveClients,
      const std::vector<ClientInfo> nackResponseClients) override;
```

```
   void onServiceStatusChange(ServiceStatus status) override;
};
```

### 3. Get an instance of PowerFactory, and then obtain a TCU-activity manager instance with ClientType as MASTER

```
auto &powerFactory = PowerFactory::getInstance();
std::promise<telux::common::ServiceStatus> prom = std::promise<telux::common::ServiceStatus>();
ClientInstanceConfig config;
config.clientType = ClientType::MASTER;
config.clientName = "client_name_" + std::to_string(getpid());
config.machineName =  ALL_MACHINES;
auto tcuActivityManager = powerFactory.getTcuActivityManager(config,
                              [&](telux::common::ServiceStatus status) {
                                 std::cout << " Init Callback called " << std::endl;
                                 prom.set_value(status);
                              });
```

### 4. Wait for the TCU-activity management services to be initialized and ready

```
if (tcuActivityStateMgr_ == nullptr) {
   std::cout << " ERROR - Failed to get manager instance" << std::endl;
}
std::cout << "  Waiting for TCU Activity Manager to be ready" << std::endl;
telux::common::ServiceStatus serviceStatus = prom.get_future().get();
```

### 5. Exit the application, if SDK is unable to initialize TCU-activity management service

```
if (serviceStatus == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   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 and register for updates on the triggered TCU-activity state and its management service status

```
auto myTcuStateListener = std::make_shared<MyTcuActivityStateListener>();
tcuActivityManager->registerListener(myTcuStateListener);
tcuActivityManager->registerServiceStateListener(myTcuStateListener);
```

### 7. Set the TCU activity state (SUSPEND here) of all the machines in the system

```
telux::common::Status status = tcuActivityManager->setActivityState(TcuActivityState::SUSPEND, ALL_MACHINES
      , &commandResponse);

if (status == telux::common::Status::SUCCESS) {
   // successfully passed trigger, wait for the command response callback (commandResponse) to know if
       trigger is accepted and getting processed
} else {
   std::cout << " *** ERROR - Failed to trigger TCU activity state change " << std::endl;
   // failed to pass trigger check status for more information
}
```

### 8. The below listener API is invoked to notify the consolidated acknowledgement status of all the SLAVE clients, corresponding to the machine that is undergoing the transition

```
void void MyTcuActivityStateListener::onSlaveAckStatusUpdate(const telux::common::Status status,
   const std::string machineName, const std::vector<ClientInfo> unresponsiveClients,
   const std::vector<ClientInfo> nackResponseClients) {
   std::cout << "********* TCU-activity state update status and consolidate slave acknowledgement
       information *********" << std::endl;
   // Avoid long blocking calls when handling notifications
```

```
   if (status != telux::common::Status::SUCCESS) {
      std::cout << " Analyze which clients have not responded or responded with nack " << std::endl;
      if(unresponsiveClients.size() > 0) {
         std::cout << " Number of unresponsive clients : "<< unresponsiveClients.size() << std::endl;
         for (size_t i = 0; i < unresponsiveClients.size(); i++) {
            std::cout << " client name : "<< unresponsiveClients[i].first
                      << " , machine name : "<< unresponsiveClients[i].second << std::endl;
         }
      }

      if(nackResponseClients.size() > 0) {
         std::cout << " Number of clients responded with nack : "<< nackResponseClients.size()
               << std::endl;
         for (size_t i = 0; i < nackResponseClients.size(); i++) {
               std::cout << " client name : "<< nackResponseClients[i].first
                         << " , machine name : "<< nackResponseClients[i].second << std::endl;
         }
      }

      // If master expects to stop state transition considering the above information, then call
      // setActivityState and revert state, or else state transition will proceed after the configured
      // timeout
      tcuActivityManager->setActivityState(TcuActivityState::RESUME, desiredVMName, &commandResponse);
   }
}
```

#### 9. Implement onServiceStatusChange callback to know when TCU-activity management service goes down

```
// 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
}
```

## 3.8 Thermal

- Get thermal zones, cooling devices and thermal notifications
- Get thermal autoshutdown mode updates
- Get/Set autoshutdown modes

### 3.8.1 Get thermal zones, cooling devices and thermal notifications

This sample app demonstrates how to get thermal zones and cooling devices.

#### 1. Get thermal factory instance

```
auto &thermalFactory = telux::therm::ThermalFactory::getInstance();
```

#### 2. Prepare initialization callback

```
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 thermal shutdown manager instance

```
std::shared_ptr<telux::therm::IThermalManager> thermalMgr
   = thermalFactory.getThermalManager(initCb);
if (!thermalMgr) {
   std::cout << "Thermal manager is nullptr" << std::endl;
   return -1;
}
```

### 4. Wait for the initialization callback and check the service status

```
telux::common::ServiceStatus serviceStatus = p.get_future().get();
if (serviceStatus != telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   std::cout << "Thermal manager initialization failed" << std::endl;
   return -2;
}
```

### 5. Create the listener object

```
std::shared_ptr<ThermalListener> thermalListener
   = std::make_shared<ThermalListener>();
```

### 6. To register only trip update notification

Note: likewise it can be registered for only cooling device changes notification.

```
thermalMgr->registerListener(thermalListener, 1 << TNT_TRIP_UPDATE);
```

### 7. Receive service status notification

```
virtual void onServiceStatusChange(ServiceStatus serviceStatus) override {
   PRINT_NOTIFICATION << "Thermal service status: ";
   std::string status;
   switch (serviceStatus) {
      case ServiceStatus::SERVICE_AVAILABLE: {
         status = "Available";
         break;
      }
      case ServiceStatus::SERVICE_UNAVAILABLE: {
         status = "Unavailable";
         break;
      }
      case ServiceStatus::SERVICE_FAILED: {
         status = "Failed";
         break;
      }
      default: {
         status = "Unknown";
         break;
      }
   }
   std::cout << status << std::endl;
}
```

### 8. Receive notification when trip event occurs

Note: Receives notification only if it is registered in step 5.

```
virtual void onTripEvent(std::shared_ptr<ITripPoint> tripPoint, TripEvent tripEvent) override {
   if (tripPoint) {
      PRINT_NOTIFICATION << ": TRIP UPDATE EVENT" << std::endl;
      printTripPointHeader();
      printTripPointInfo(tripPoint, tripEvent);
      return;
   }
   PRINT_NOTIFICATION << ": Invalid trip point" << std::endl;
```

```
}
```

#### 9. To de-register only trip update notification

Note: likewise it can be de-registered for only cooling device changes notification. The SSR notification will not de-registered by default except mask: 0xFFFF.

```
thermalMgr->registerListener(thermalListener, 1 << TNT_TRIP_UPDATE);
```

#### 10. Send get thermal zones request using thermal manager object

```
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;
}
```

#### 11. Send get cooling devices request using thermal manager instance

```
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;
}
```

#### 12. Send request to get thermal zone for specific id using thermal manager object

```
int thermalZoneId = THERMAL_ZONE_ID;
std::cout << "Thermal zone info by Id: " << thermalZoneId << std::endl;
std::shared_ptr<telux::therm::IThermalZone> tzInfo = thermalMgr->getThermalZone(thermalZoneId);
if (tzInfo != nullptr) {
   printThermalZoneHeader();
   printZoneInfo(tzInfo);
   printBindingInfo(tzInfo);
}
```

#### 13. Cleanup when we don't need to listen to anything and when exit the application.

Note: Here the APP is de-registering all notifications. However, the client can choose to de-register specific as well. For example, to deregister only trip update notifications, the client may provide mask: 0x0001 or mask: 0x0002 to deregister only the notification for change in cdev level.

```
thermalMgr->deregisterListener(thermalListener);
thermalListener = nullptr;
thermalMgr = nullptr;
```

### 3.8.2 Get thermal autoshutdown mode updates

This sample app demonstrates how to get thermal autoshutdown 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 instance

```
auto &thermalFactory = telux::therm::ThermalFactory::getInstance();
```

#### 3. Prepare initialization callback

```
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);
};
```

#### 4. Get thermal shutdown manager instance

```
thermShutdownMgr_ = thermalFactory.getThermalShutdownManager(initCb);
if(thermShutdownMgr_ == NULL)
{
    std::cout << APP_NAME << " *** ERROR - Failed to get manager instance" << std::endl;
    return -1;
}
```

#### 5. Wait for the initialization callback and check the service status

```
telux::common::ServiceStatus serviceStatus = p.get_future().get();
if(serviceStatus == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   std::cout << APP_NAME << " Thermal-Shutdown management services are ready !" << std::endl;
} else {
   std::cout << APP_NAME << " *** ERROR - Unable to initialize Thermal-Shutdown management "
                            "services" << std::endl;
   return -1;
}
```

#### 6. Instantiate MyThermalStateListener

```
auto myThermalModeListener = std::make_shared<MyThermalShutdownModeListener>();
```

#### 7. Register for updates on thermal autoshutdown mode and its management service status

```
thermShutdownMgr_->registerListener(myThermalModeListener);
```

#### 8. 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;
}
```

#### 9. Implement onServiceStatusChange callback to know when thermal shutdown management service goes down

```
// When the thermal management service goes down, this API is invoked
// with status UNAVAILABLE. All thermal auto shutdown mode notifications
// 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
}
```

### 3.8.3 Get/Set autoshutdown modes

This sample app demonstrates how to get/set thermal autoshutdown mode.

#### 1. Get thermal factory instance

```
auto &thermalFactory = telux::therm::ThermalFactory::getInstance();
```

#### 2. Prepare initialization callback

```
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 thermal shutdown manager instance

```
thermShutdownMgr_ = thermalFactory.getThermalShutdownManager(initCb);
if(thermShutdownMgr_ == NULL)
{
   std::cout << APP_NAME << " *** ERROR - Failed to get manager instance" << std::endl;
   return -1;
}
```

#### 4. Wait for the initialization callback and check the service status

```
telux::common::ServiceStatus serviceStatus = p.get_future().get();
if(serviceStatus == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   std::cout << APP_NAME << " Thermal-Shutdown management services are ready !" << std::endl;
} else {
   std::cout << APP_NAME << " *** ERROR - Unable to initialize Thermal-Shutdown management "
                            "services" << std::endl;
   return -1;
}
```

#### 5. 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;
}
```

#### 6. 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;
}
```

## 3.9 Sensor

- Configure and acquire sensor data
- Control sensor features

### 3.9.1 Configure and acquire sensor data

This sample application demonstrates how to configure and acquire sensor data. The sample application also shows how to acquire data from the same sensor with different configurations

#### 1. Get sensor factory instance

```
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::ISensorClient> lowRateSensorClient;
std::cout << "Getting sensor: " << name << std::endl;
status = sensorManager->getSensorClient(lowRateSensorClient, 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::string name, std::shared_ptr<telux::sensor::ISensorClient> sensor)
      : name_(name)
      , sensorClient_(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 << "(" << name_ << "): Received " << events->size()
                         << " events from sensor: " << sensorClient_->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([&] {
               sensorClient_->deactivate();
               sensorClient_->configure(sensorClient_->getConfiguration());
               sensorClient_->activate();
            });
            // Be sure to detach the thread
            t.detach();
      }
   }

   // [9] Receive configuration updates
   virtual void onConfigurationUpdate(telux::sensor::SensorConfiguration configuration) override {
      PRINT_NOTIFICATION << "(" << name_ << "): Received configuration update from sensor: "
                         << sensorClient_->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 = sensorClient_->getSensorInfo();
      if (isUncalibratedSensor(sensorClient_->getSensorInfo().type)) {
            PRINT_NOTIFICATION << ": " << sensorClient_->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 << ": " << sensorClient_->getSensorInfo().name << ": " << s.timestamp
                               << ", " << s.calibrated.x << ", " << s.calibrated.y << ", "
                               << s.calibrated.z << std::endl;
      }
   }

   std::string name_;
   std::shared_ptr<telux::sensor::ISensorClient> sensorClient_;
   uint32_t totalBatches_;
};
```

### Create a event listener and register it with the sensor.

```
std::shared_ptr<SensorEventListener> lowRateSensorClientEventListener
   = std::make_shared<SensorEventListener>("Low-rate", lowRateSensorClient);
lowRateSensorClient->registerListener(lowRateSensorClientEventListener);
```

### 8. Configure the sensor with required configuration setting the necessary validityMask

```
telux::sensor::SensorConfiguration lowRateConfig;
lowRateConfig.samplingRate = getMinimumSamplingRate(lowRateSensorClient->getSensorInfo());
lowRateConfig.batchCount = lowRateSensorClient->getSensorInfo().maxBatchCountSupported;
std::cout << "Configuring sensor with samplingRate, batchCount [" << lowRateConfig.samplingRate
          << ", " << lowRateConfig.batchCount << "]" << std::endl;
lowRateConfig.validityMask.set(telux::sensor::SensorConfigParams::SAMPLING_RATE);
lowRateConfig.validityMask.set(telux::sensor::SensorConfigParams::BATCH_COUNT);
status = lowRateSensorClient->configure(lowRateConfig);
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 << "(" << name_ << "): Received configuration update from sensor: "
                      << sensorClient_->getSensorInfo().name << ": ["
                      << configuration.samplingRate << ", " << configuration.batchCount << " ]"
                      << std::endl;
}
```

### 10. Activate the sensor to receive sensor data

```
status = lowRateSensorClient->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 << "(" << name_ << "): Received " << events->size()
                      << " events from sensor: " << sensorClient_->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([&] {
            sensorClient_->deactivate();
            sensorClient_->configure(sensorClient_->getConfiguration());
            sensorClient_->activate();
      });
      // Be sure to detach the thread
      t.detach();
   }
}
```

### 12. Create another sensor client for the same sensor and it's corresponding listener

```
std::shared_ptr<telux::sensor::ISensorClient> highRateSensorClient;
std::cout << "Getting sensor: " << name << std::endl;
status = sensorManager->getSensorClient(highRateSensorClient, name);
if (status != telux::common::Status::SUCCESS) {
   std::cout << "Failed to get sensor: " << name << std::endl;
   exit(1);
}
std::shared_ptr<SensorEventListener> highRateSensorEventListener
   = std::make_shared<SensorEventListener>("High-rate", highRateSensorClient);
highRateSensorClient->registerListener(highRateSensorEventListener);
```

### 13. Configure this sensor client with a different configuration, as necessary

```
telux::sensor::SensorConfiguration highRateConfig;
highRateConfig.samplingRate = getMaximumSamplingRate(highRateSensorClient->getSensorInfo());
highRateConfig.batchCount = highRateSensorClient->getSensorInfo().maxBatchCountSupported;
std::cout << "Configuring sensor with samplingRate, batchCount [" << highRateConfig.samplingRate
          << ", " << highRateConfig.batchCount << "]" << std::endl;
highRateConfig.validityMask.set(telux::sensor::SensorConfigParams::SAMPLING_RATE);
highRateConfig.validityMask.set(telux::sensor::SensorConfigParams::BATCH_COUNT);
status = highRateSensorClient->configure(highRateConfig);
if (status != telux::common::Status::SUCCESS) {
   std::cout << "Failed to configure sensor: " << name << std::endl;
   exit(1);
}
```

### 14. Activate this sensor as well

```
status = highRateSensorClient->activate();
if (status != telux::common::Status::SUCCESS) {
   std::cout << "Failed to activate sensor: " << name << std::endl;
   exit(1);
}
```

### 15. When data acquisition is no longer necessary, deactivate the sensors

```
status = lowRateSensorClient->deactivate();
if (status != telux::common::Status::SUCCESS) {
   std::cout << "Failed to deactivate sensor: " << name << std::endl;
   exit(1);
}
status = highRateSensorClient->deactivate();
if (status != telux::common::Status::SUCCESS) {
   std::cout << "Failed to deactivate sensor: " << name << std::endl;
   exit(1);
}
```

### 16. Release the instances of ISensorClient if no longer required

```
lowRateSensorClient = nullptr;
highRateSensorClient = nullptr;
```

### 17. Release the instance of ISensorManager to cleanup resources

```
sensorManager = nullptr;
```

## Using Sensor APIs to initiate a self test and acquires the self test result

Please follow below steps as a guide to initiate a self test and acquires the self test result

### 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::ISensorClient> sensor;
std::cout << "Getting sensor: " << name << std::endl;
status = sensorManager->getSensorClient(sensor, name);
if (status != telux::common::Status::SUCCESS) {
   std::cout << "Failed to get sensor: " << name << std::endl;
   exit(1);
}
```

### 7. Invoke the self test with the required self test type and provide the callback

```
status = sensor->selfTest(selfTestType, [](telux::common::ErrorCode result) {
   PRINT_CB << "Received self test response: " << static_cast<int>(result) << std::endl;
});
if (status != telux::common::Status::SUCCESS) {
   std::cout << "Self test request failed";
} else {
   std::cout << "Self test request successful, waiting for callback" << std::endl;
}
```

### 8. Release the instance of ISensorClient if no longer required

```
sensor = nullptr;
```

### 9. Release the instance of ISensorManager to cleanup resources

```
sensorManager = nullptr;
```

### 3.9.2 Control sensor features

This sample application demonstrates how to control sensor features.

#### 1. Get sensor factory instance

```
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);
}
```

Note: Enabling a sensor feature when the system is active would additionally require enabling the corresponding sensor which is used by the sensor feature. If the sensor feature only needs to be enabled during suspend mode, just enabling the sensor feature using this method would be sufficient. The underlying framework would take care to enable the required sensor when the system is about to enter suspend state.

#### 8. Receive sensor feature events with the registered listener when device is not suspended

```
virtual void onEvent(telux::sensor::SensorFeatureEvent event) override {
   printSensorFeatureEvent(event);
}
```

#### 9. Receive sensor feature events with the registered listener when device in suspended state

```
virtual void onBufferedEvent(std::string sensorName,
                 std::shared_ptr<std::vector<SensorEvent>> events, bool isLast) override {
   printSensorFeatureEvent(event);
}
```

#### 10. 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;
```

## 3.10 Platform

- EFS backup and restore
- OTA operation
- Ecall operation

### 3.10.1 EFS backup and restore

This sample app demonstrates how to use APIs to request EFS backup and listen to EFS restore and backup indications.

#### 1. Get platform factory instance

```
auto &platformFactory = telux::platform::PlatformFactory::getInstance();
```

### 2. Prepare a callback that is invoked when the filesystem 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 filesystem manager

```
std::shared_ptr<telux::platform::IFsManager> fsManager = platformFactory.getFsManager(initCb);
if (fsManager == nullptr) {
   std::cout << "filesystem manager is nullptr" << std::endl;
   exit(1);
}
std::cout << "Obtained filesystem manager" << std::endl;
```

### 4. Wait until initialization is complete

```
p.get_future().get();
if (fsManager->getServiceStatus() != telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   std::cout << "Filesystem service not available" << std::endl;
   exit(1);
}
std::cout << "Filesystem service is now available" << std::endl;
```

### 5. Create the listener object and register as a listener

```
std::shared_ptr<EfsEventListener> efsEventListener = std::make_shared<EfsEventListener>();
fsManager->registerListener(efsEventListener);
```

### 6. Receive service status notifications

```
virtual void onServiceStatusChange(telux::common::ServiceStatus serviceStatus) override {
   PRINT_NOTIFICATION << "Filesystem service status: ";
   std::string status;
   switch (serviceStatus) {
      case telux::common::ServiceStatus::SERVICE_AVAILABLE: {
            status = "Available";
            break;
      }
      case telux::common::ServiceStatus::SERVICE_UNAVAILABLE: {
            status = "Unavailable";
            break;
      }
      case telux::common::ServiceStatus::SERVICE_FAILED: {
            status = "Failed";
            break;
      }
      default: {
            status = "Unknown";
            break;
      }
   }
   std::cout << status << std::endl;
}
```

### 7. Start EFS backup whenever necessary

```
telux::common::Status status = fsManager->startEfsBackup();
if(status != telux::common::Status::SUCCESS) {
   std::cout << "Unable to start EFS backup: ";
   Utils::printStatus(status);
}
```

#### 8. Receive EFS restore and backup notifications

```
virtual void OnEfsRestoreEvent(telux::platform::EfsEventInfo event) override {
   PRINT_NOTIFICATION
      << ": Received efs event: Restore"
      << ((event.event == telux::platform::EfsEvent::START) ? " started" : "ended");
   if (event.event == telux::platform::EfsEvent::END) {
      std::cout << " with result: " << Utils::getErrorCodeAsString(event.error) << std::endl;
   }
}

virtual void OnEfsBackupEvent(telux::platform::EfsEventInfo event) override {
   PRINT_NOTIFICATION
      << ": Received efs event: Backup"
      << ((event.event == telux::platform::EfsEvent::START) ? " started" : "ended");
   if (event.event == telux::platform::EfsEvent::END) {
      std::cout << " with result: " << Utils::getErrorCodeAsString(event.error) << std::endl;
   }
}
```

#### 9. Clean-up when we do not need to listen anything

```
fsManager->deregisterListener(efsEventListener);
efsEventListener = nullptr;
fsManager = nullptr;
```

### 3.10.2 OTA operation

This sample app demonstrates how to prepare for an ota operation, when to initiate an ota update and handle post-ota operations.

#### 1. Get platform factory instance

```
auto &platformFactory = telux::platform::PlatformFactory::getInstance();
```

#### 2. Prepare a callback that is invoked when the filesystem 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 filesystem manager

```
std::shared_ptr<telux::platform::IFsManager> fsManager = platformFactory.getFsManager(initCb);
if (fsManager == nullptr) {
   std::cout << "filesystem manager is nullptr" << std::endl;
   exit(1);
}
std::cout << "Obtained filesystem manager" << std::endl;
```

#### 4. Wait until initialization is complete

```
p.get_future().get();
if (fsManager->getServiceStatus() != telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   std::cout << "Filesystem service not available" << std::endl;
   exit(1);
}
std::cout << "Filesystem service is now available" << std::endl;
```

### 5. Create the listener object and register as a listener

```
std::shared_ptr<OtaOperationsListener> otaOperationsListener
    = std::make_shared<OtaOperationsListener>();
fsManager->registerListener(otaOperationsListener);
```

### 6. Receive service status notifications

```
virtual void onServiceStatusChange(telux::common::ServiceStatus serviceStatus) override {
   PRINT_NOTIFICATION << "Ota operation service status: ";
   std::string status;
   switch (serviceStatus) {
      case ServiceStatus::SERVICE_AVAILABLE: {
         status = "Available";
         break;
      }
      case ServiceStatus::SERVICE_UNAVAILABLE: {
         status = "Unavailable";
         break;
      }
      case ServiceStatus::SERVICE_FAILED: {
         status = "Failed";
         break;
      }
      default: {
         status = "Unknown";
         break;
      }
   }
   std::cout << status << std::endl;
 }
```

### 7. Download the package

### 8. Prepare for OTA start

```
telux::common::Status otaStartStatus = telux::common::Status::FAILED;
OtaOperation otaOperation = OtaOperation::START;
std::promise<ErrorCode> p;

otaStartStatus = fsManager->prepareForOta(
    otaOperation, [&p, fsManager](ErrorCode error) { p.set_value(error); });

if (otaStartStatus != telux::common::Status::SUCCESS) {
   std::cout << "Request to prepare for ota start : ";
   Utils::printStatus(otaStartStatus);
   exit(1);
} else {
   std::cout << "Request to prepare for ota start successful";
   ErrorCode error = p.get_future().get();
   std::cout << "Prepare for ota start with result: " << Utils::getErrorCodeAsString(error)
             << std::endl;
}
```

### 9. The client can start the OTA update

### 10. Once the OTA is complete, indicate the update status to filesystem manager

```
telux::common::Status otaEndStatus = telux::common::Status::FAILED;
OperationStatus operationStatus = OperationStatus::SUCCESS;
std::promise<ErrorCode> p;

otaEndStatus = fsManager->otaCompleted(
    operationStatus, [&p, fsManager](ErrorCode error) { p.set_value(error); });

if (otaEndStatus != telux::common::Status::SUCCESS) {
```

```
   std::cout << "Ota completion for update succeed request : ";
   Utils::printStatus(otaEndStatus);
   // Note: If OTA completion result in a failure, the client needs to start
   // with prepareForOta
} else {
   std::cout << "Ota completion for update succeed request successful";
   ErrorCode error = p.get_future().get();
   std::cout << " ota completed for update succeed with result: " << Utils::getErrorCodeAsString(error)
             << std::endl;
}
```

#### 11. If user decides to mirror the filesystem, start AB sync

```
telux::common::Status abSyncStatus = telux::common::Status::FAILED;
std::promise<ErrorCode> p;

abSyncStatus = fsManager->startAbSync([&p, fsManager](ErrorCode error) { p.set_value(error); });

if (abSyncStatus != telux::common::Status::SUCCESS) {
   std::cout << "Request to start absync : ";
   Utils::printStatus(abSyncStatus);
} else {
   std::cout << "Request to start absync successful";
   ErrorCode error = p.get_future().get();
   std::cout << "Start absync with result: " << Utils::getErrorCodeAsString(error) << std::endl;
}
```

#### 12. Clean-up when we do not need to listen anything

```
fsManager->deregisterListener(otaOperationsListener);
otaOperationsListener = nullptr;
fsManager = nullptr;
```

### 3.10.3 Ecall operation

This sample app demonstrates how to prepare for an eCall operation and indicate eCall completion.

#### 1. Get platform factory instance

```
auto &platformFactory = PlatformFactory::getInstance();
```

#### 2. Prepare a callback that is invoked when the filesystem 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 filesystem manager

```
std::shared_ptr<telux::platform::IFsManager> fsManager = platformFactory.getFsManager(initCb);
if (fsManager == nullptr) {
   std::cout << "filesystem manager is nullptr" << std::endl;
   exit(1);
}
std::cout << "Obtained filesystem manager" << std::endl;
```

### 4. Wait until initialization is complete

```
p.get_future().get();
if (fsManager->getServiceStatus() != telux::common::ServiceStatus::SERVICE_AVAILABLE) {
   std::cout << "Filesystem service not available" << std::endl;
   exit(1);
}
std::cout << "Filesystem service is now available" << std::endl;
```

### 5. Create the listener object and register as a listener

```
std::shared_ptr<EcallOperationListener> ecallOperationListener
    = std::make_shared<EcallOperationListener>();
fsManager->registerListener(ecallOperationListener);
```

### 6. Receive service status notifications

```
virtual void onServiceStatusChange(telux::common::ServiceStatus serviceStatus) override {
   PRINT_NOTIFICATION << "Ecall operation service status: ";
   std::string status;
   switch (serviceStatus) {
      case ServiceStatus::SERVICE_AVAILABLE: {
         status = "Available";
         break;
      }
      case ServiceStatus::SERVICE_UNAVAILABLE: {
         status = "Unavailable";
         break;
      }
      case ServiceStatus::SERVICE_FAILED: {
         status = "Failed";
         break;
      }
      default: {
         status = "Unknown";
         break;
      }
   }
   std::cout << status << std::endl;
}
```

### 7. Before starting an eCall, prepare the filesystem manager for an eCall

Note: It is recommended to start an eCall even if the request to prepare for an eCall fails.this API can re-invoke while an eCall is ongoing.

```
telux::common::Status ecallStartStatus = telux::common::Status::FAILED;
std::cout << "Request to prepare for eCall start invoked " << std::endl;

ecallStartStatus = fsManager->prepareForEcall();

if (ecallStartStatus == telux::common::Status::SUCCESS) {
   std::cout << "Request to prepare for eCall start successful";
} else {
   std::cout << "Request to prepare for eCall start failed : ";
   Utils::printStatus(ecallStartStatus);
}
```

### 8. Initiate an eCall

#### 9. Receive notification when filesystem operation is about to resumes in seconds - timeLetftToStart

Note: *On this notification, the client can still suspand the filesystem operation and continue the eCall by invoking prepareForEcall API (Step[7])*

```
virtual void OnFsOperationImminentEvent(uint32_t timeLeftToStart) override {
     PRINT_NOTIFICATION << "Filesystem operation resumes in seconds: ";
     std::cout << timeLeftToStart << std::endl;
}
```

#### 9. Once an eCall is completed, indicate to the filesystem manager

```
telux::common::Status ecallEndStatus = telux::common::Status::FAILED;
std::cout << "eCall completion request being invoked " << std::endl;

ecallEndStatus = fsManager->eCallCompleted();

if (ecallEndStatus == telux::common::Status::SUCCESS) {
   std::cout << "eCall completion request successful";
} else {
   std::cout << "eCall completion request failed : ";
   Utils::printStatus(ecallEndStatus);
}
```

#### 10. Clean-up when we do not need to listen anything

```
fsManager->deregisterListener(ecallOperationListener);
ecallOperationListener = nullptr;
fsManager = nullptr;
```

## 3.11 Crypto

- Sign and verify using HMAC key
- Sign and verify using EC key
- Sign and verify using RSA key
- Encrypt and decrypt using AES key
- Export a RSA key
- Import a RSA key

### 3.11.1 Sign and verify using HMAC key

This sample app demonstrates how to generate a HMAC key, sign given data using this key and verify data using this key.

#### 1. Include headers files pertaining to security APIs

```
#include <telux/sec/CryptoParamBuilder.hpp>
#include <telux/sec/CryptoManager.hpp>
#include <telux/sec/SecurityFactory.hpp>
```

#### 2. Get a SecurityFactory instance

```
auto &secFact = telux::sec::SecurityFactory::getInstance();
```

### 3. Get a CryptoManager instance

```
std::shared_ptr<telux::sec::ICryptoManager> cryptMgr;

cryptMgr = secFact.getCryptoManager(ec);
if (!cryptMgr) {
    std::cout << "Can't allocate CryptoManager, err: " <<
            static_cast<int>(ec) << std::endl;
    return -1;
}
```

### 4. Define parameters using CryptoParamBuilder for the key

```
std::shared_ptr<telux::sec::ICryptoParam> cp;

cp = telux::sec::CryptoParamBuilder()
        .setAlgorithm(telux::sec::Algorithm::ALGORITHM_HMAC)
        .setCryptoOperation(telux::sec::CryptoOperation::CRYPTO_OP_SIGN |
                    telux::sec::CryptoOperation::CRYPTO_OP_VERIFY)
        .setKeySize(128)
        .setDigest(telux::sec::Digest::DIGEST_SHA_2_256)
        .setMinimumMacLength(64)
        .build();
```

### 5. Generate HMAC key using generateKey() API

```
std::vector<uint8_t> kb;
telux::common::ErrorCode ec;

ec = cryptMgr->generateKey(cp, kb);
if (ec != telux::common::ErrorCode::SUCCESS) {
    std::cout << "Can't generate HMAC key, err: " <<
            static_cast<int>(ec) << std::endl;
    return -1;
}
```

### 6. Define data to be signed

```
std::vector<uint8_t> pt {'h', 'e', 'l', 'l', 'o'};
```

### 7. Define parameters using CryptoParamBuilder for signing/verifying data

```
cp = telux::sec::CryptoParamBuilder()
        .setAlgorithm(telux::sec::Algorithm::ALGORITHM_HMAC)
        .setDigest(telux::sec::Digest::DIGEST_SHA_2_256)
        .build();
```

### 8. Sign the required data using signData() API

```
std::vector<uint8_t> sg;

ec = cryptMgr->signData(cp, kb, pt, sg);
if (ec != telux::common::ErrorCode::SUCCESS) {
    std::cout << "Can't sign data, err: " << static_cast<int>(ec) << std::endl;
    return -1;
}
```

### 9. Verify authenticity of data using verifyData() API

```
ec = cryptMgr->verifyData(cp, kb, pt, sg);
if (ec != telux::common::ErrorCode::SUCCESS) {
    if (ec == telux::common::ErrorCode::VERIFICATION_FAILED)
        std::cout << "Invalid signature for given data!" << std::endl;
    else
        std::cout << "Can't verify data, err: " << static_cast<int>(ec) << std::endl;
```

```
    return -1;
}
```

### 3.11.2 Sign and verify using EC key

This sample app demonstrates how to generate a EC key, sign given data using this key and verify data using this key.

#### 1. Get a SecurityFactory instance

```
auto &secFact = telux::sec::SecurityFactory::getInstance();
```

#### 2. Get a CryptoManager instance

```
std::shared_ptr<telux::sec::ICryptoManager> cryptMgr;

cryptMgr = secFact.getCryptoManager(ec);
if (!cryptMgr) {
    std::cout << "Can't allocate CryptoManager, err: " <<
            static_cast<int>(ec) << std::endl;
    return -1;
}
```

#### 3. Define parameters using CryptoParamBuilder for the key

```
std::shared_ptr<telux::sec::ICryptoParam> cp;

cp = telux::sec::CryptoParamBuilder()
        .setAlgorithm(telux::sec::Algorithm::ALGORITHM_EC)
        .setCryptoOperation(telux::sec::CryptoOperation::CRYPTO_OP_SIGN |
                    telux::sec::CryptoOperation::CRYPTO_OP_VERIFY)
        .setKeySize(256)
        .setDigest(telux::sec::Digest::DIGEST_SHA_2_256)
        .build();
```

#### 4. Generate EC key using generateKey() API

```
std::vector<uint8_t> kb;
telux::common::ErrorCode ec;

ec = cryptMgr->generateKey(cp, kb);
if (ec != telux::common::ErrorCode::SUCCESS) {
    std::cout << "Can't generate EC asym key, err: " <<
            static_cast<int>(ec) << std::endl;
    return -1;
}
```

#### 5. Define data to be signed

```
std::vector<uint8_t> pt {'h', 'e', 'l', 'l', 'o'};
```

#### 6. Define parameters using CryptoParamBuilder for signing/verifying data

```
std::shared_ptr<telux::sec::ICryptoParam> cp;

cp = telux::sec::CryptoParamBuilder()
        .setAlgorithm(telux::sec::Algorithm::ALGORITHM_EC)
        .setDigest(telux::sec::Digest::DIGEST_SHA_2_256)
        .build();
```

#### 7. Sign the required data using signData() API

```
std::vector<uint8_t> sg;

ec = cryptMgr->signData(cp, kb, pt, sg);
if (ec != telux::common::ErrorCode::SUCCESS) {
    std::cout << "Can't sign data, err: " <<
            static_cast<int>(ec) << std::endl;
    return -1;
}
```

#### 8. Verify authenticity of data using verifyData() API

```
ec = cryptMgr->verifyData(cp, kb, pt, sg);
if (ec != telux::common::ErrorCode::SUCCESS) {
    if (ec == telux::common::ErrorCode::VERIFICATION_FAILED)
        std::cout << "Invalid signature for given data!" << std::endl;
    else
        std::cout << "Can't verify data, err: " <<
                        static_cast<int>(ec) << std::endl;
    return -1;
}
```

### 3.11.3 Sign and verify using RSA key

This sample app demonstrates how to generate a RSA key, sign given data using this key and verify data using this key.

#### 1. Get a SecurityFactory instance

```
auto &secFact = telux::sec::SecurityFactory::getInstance();
```

#### 2. Get a CryptoManager instance

```
std::shared_ptr<telux::sec::ICryptoManager> cryptMgr;

cryptMgr = secFact.getCryptoManager(ec);
if (!cryptMgr) {
    std::cout << "Can't allocate CryptoManager, err: " <<
            static_cast<int>(ec) << std::endl;
    return -1;
}
```

#### 3. Define parameters using CryptoParamBuilder for the key

```
std::shared_ptr<telux::sec::ICryptoParam> cp;

cp = telux::sec::CryptoParamBuilder()
        .setAlgorithm(telux::sec::Algorithm::ALGORITHM_RSA)
        .setCryptoOperation(telux::sec::CryptoOperation::CRYPTO_OP_SIGN |
                    telux::sec::CryptoOperation::CRYPTO_OP_VERIFY)
        .setKeySize(2048)
        .setPublicExponent(65537)
        .setDigest(telux::sec::Digest::DIGEST_SHA_2_256 |
                telux::sec::Digest::DIGEST_SHA_2_512)
        .setPadding(telux::sec::Padding::PADDING_RSA_PSS |
                telux::sec::Padding::PADDING_RSA_PKCS1_1_5_SIGN)
        .build();
```

#### 4. Generate RSA key using generateKey() API

```
telux::common::ErrorCode ec;
std::vector<uint8_t> kb;

ec = cryptMgr->generateKey(cp, kb);
if (ec != telux::common::ErrorCode::SUCCESS) {
    std::cout << "Can't generate RSA asym key, err: " <<
            static_cast<int>(ec) << std::endl;
    return -1;
}
```

#### 5. Define data to be signed

```
std::vector<uint8_t> pt {'h', 'e', 'l', 'l', 'o'};
```

#### 6. Define parameters using CryptoParamBuilder for signing/verifying data

```
cp = telux::sec::CryptoParamBuilder()
        .setAlgorithm(telux::sec::Algorithm::ALGORITHM_RSA)
        .setDigest(telux::sec::Digest::DIGEST_SHA_2_256)
        .setPadding(telux::sec::Padding::PADDING_RSA_PKCS1_1_5_SIGN)
        .build();
```

#### 7. Sign the required data using signData() API

```
std::vector<uint8_t> sg;

ec = cryptMgr->signData(cp, kb, pt, sg);
if (ec != telux::common::ErrorCode::SUCCESS) {
    std::cout << "Can't sign data, err: " <<
            static_cast<int>(ec) << std::endl;
    return -1;
}
```

#### 8. Verify authenticity of data using verifyData() API

```
ec = cryptMgr->verifyData(cp, kb, pt, sg);

if (ec != telux::common::ErrorCode::SUCCESS) {
    if (ec == telux::common::ErrorCode::VERIFICATION_FAILED)
        std::cout << "Invalid signature for given data!" << std::endl;
    else
        std::cout << "Can't verify data, err: " << static_cast<int>(ec) << std::endl;
    return -1;
}
```

### 3.11.4 Encrypt and decrypt using AES key

This sample app demonstrates how to generate a AES key, encrypt and decrypt given data using this key.

#### 1. Get a SecurityFactory instance

```
auto &secFact = telux::sec::SecurityFactory::getInstance();
```

#### 2. Get a CryptoManager instance

```
std::shared_ptr<telux::sec::ICryptoManager> cryptMgr;

cryptMgr = secFact.getCryptoManager(ec);
if (!cryptMgr) {
    std::cout << "Can't allocate CryptoManager, err: " <<
            static_cast<int>(ec) << std::endl;
```

```
    return -1;
}
```

### 3. Define parameters using CryptoParamBuilder for the key

```
std::shared_ptr<telux::sec::ICryptoParam> cp;

cp = telux::sec::CryptoParamBuilder()
        .setAlgorithm(telux::sec::Algorithm::ALGORITHM_AES)
        .setCryptoOperation(telux::sec::CryptoOperation::CRYPTO_OP_ENCRYPT |
                    telux::sec::CryptoOperation::CRYPTO_OP_DECRYPT)
        .setKeySize(128)
        .setBlockMode(telux::sec::BlockMode::BLOCK_MODE_CBC |
                    telux::sec::BlockMode::BLOCK_MODE_CTR)
        .setPadding(telux::sec::Padding::PADDING_PKCS7 |
                    telux::sec::Padding::PADDING_NONE)
        .setCallerNonce(true)
        .build();
```

### 4. Generate AES key using generateKey() API

```
std::vector<uint8_t> kb;
telux::common::ErrorCode ec;

ec = cryptMgr->generateKey(cp, kb);
if (ec != telux::common::ErrorCode::SUCCESS) {
    std::cout << "Can't generate AES sym key, err: " <<
            static_cast<int>(ec) << std::endl;
    return -1;
}
```

### 5. Define data to be encrypted

```
std::vector<uint8_t> pt {'h', 'e', 'l', 'l', 'o'};
```

### 6. Specify initialization vector for encryption/decryption

```
std::vector<uint8_t> initVector((128/8), 0x01);
```

### 7. Define parameters using CryptoParamBuilder for encryption/decryption

```
cp = telux::sec::CryptoParamBuilder()
        .setAlgorithm(telux::sec::Algorithm::ALGORITHM_AES)
        .setBlockMode(telux::sec::BlockMode::BLOCK_MODE_CBC)
        .setPadding(telux::sec::Padding::PADDING_PKCS7)
        .setInitVector(initVector)
        .build();
```

### 8. Encrypt the required data using encryptData() API

```
std::vector<uint8_t> et;

ec = cryptMgr->encryptData(cp, kb, pt, et);
if (ec != telux::common::ErrorCode::SUCCESS) {
    std::cout << "Can't encrypt data, err: " <<
            static_cast<int>(ec) << std::endl;
    return -1;
}
```

#### 9. Decrypt data using decryptData() API

```
std::vector<uint8_t> dt;

ec = cryptMgr->decryptData(cp, kb, et, dt);
if (ec != telux::common::ErrorCode::SUCCESS) {
    std::cout << "Can't decrypt data, err: " <<
            static_cast<int>(ec) << std::endl;
    return -1;
}
```

### 3.11.5 Export a RSA key

This sample app demonstrates how to generate a RSA key and export it given standard format.

#### 1. Get a SecurityFactory instance

```
auto &secFact = telux::sec::SecurityFactory::getInstance();
```

#### 2. Get a CryptoManager instance

```
std::shared_ptr<telux::sec::ICryptoManager> cryptMgr;

cryptMgr = secFact.getCryptoManager(ec);
if (!cryptMgr) {
    std::cout << "Can't allocate CryptoManager, err: " <<
            static_cast<int>(ec) << std::endl;
    return -1;
}
```

#### 3. Define parameters using CryptoParamBuilder for the key

```
std::shared_ptr<telux::sec::ICryptoParam> cp;

cp = telux::sec::CryptoParamBuilder()
        .setAlgorithm(telux::sec::Algorithm::ALGORITHM_RSA)
        .setCryptoOperation(telux::sec::CryptoOperation::CRYPTO_OP_SIGN |
                    telux::sec::CryptoOperation::CRYPTO_OP_VERIFY)
        .setKeySize(2048)
        .setPublicExponent(65537)
        .setDigest(telux::sec::Digest::DIGEST_SHA_2_256 |
                telux::sec::Digest::DIGEST_SHA_2_512)
        .setPadding(telux::sec::Padding::PADDING_RSA_PSS |
                telux::sec::Padding::PADDING_RSA_PKCS1_1_5_SIGN)
        .build();
```

#### 4. Generate RSA key using generateKey() API

```
telux::common::ErrorCode ec;
std::vector<uint8_t> kb;

ec = cryptMgr->generateKey(cp, kb);
if (ec != telux::common::ErrorCode::SUCCESS) {
    std::cout << "Can't generate RSA asym key, err: " <<
            static_cast<int>(ec) << std::endl;
    return -1;
}
```

#### 5. Export key in given format in an array using exportKey() API

```
std::vector<uint8_t> keyData;

ec = cryptMgr->exportKey(telux::sec::KeyFormat::KEY_FORMAT_X509, kb, keyData);
if (ec != telux::common::ErrorCode::SUCCESS) {
```

```
    std::cout << "Can't export key, err: " <<
            static_cast<int>(ec) << std::endl;
    return -1;
}
```

## 3.11.6 Import a RSA key

This sample app demonstrates how to generate a RSA key and export it given standard format.

### 1. Get a SecurityFactory instance

```
auto &secFact = telux::sec::SecurityFactory::getInstance();
```

### 2. Get a CryptoManager instance

```
std::shared_ptr<telux::sec::ICryptoManager> cryptMgr;

cryptMgr = secFact.getCryptoManager(ec);
if (!cryptMgr) {
    std::cout << "Can't allocate CryptoManager, err: " <<
            static_cast<int>(ec) << std::endl;
    return -1;
}
```

### 3. Define the key data to be imported

```
/*
 * Generated using openssl:
 * openssl genpkey -algorithm rsa -pkeyopt rsa_keygen_bits:2048 -pkeyopt rsa_keygen_pubexp:3 -out
       rsa_key.der -outform der
 * openssl pkcs8 -in rsa_key.der -inform der -out rsa_key_pkcs8.der -outform der -topk8 -nocrypt
 * xxd -i rsa_key_pkcs8.der
 */
uint8_t rsaKey[] = {
        0x30, 0x82, 0x04, 0xbd, 0x02, 0x01, 0x00, 0x30, 0x0d, 0x06, 0x09, 0x2a,
        0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x01, 0x01, 0x05, 0x00, 0x04, 0x82,
        0x04, 0xa7, 0x30, 0x82, 0x04, 0xa3, 0x02, 0x01, 0x00, 0x02, 0x82, 0x01,
        0x01, 0x00, 0xba, 0xdb, 0xd3, 0x09, 0x92, 0x24, 0xd9, 0x18, 0xd7, 0x75,
        0x6c, 0xd5, 0x27, 0xef, 0xe5, 0x9c, 0xca, 0xd3, 0x81, 0xbf, 0xf0, 0xf4,
        0x0a, 0xa0, 0x3f, 0x2b, 0x54, 0x68, 0x67, 0xef, 0x34, 0xce, 0x5d, 0x3d,
        0xde, 0xa5, 0x15, 0xcb, 0xd7, 0xae, 0xa2, 0xf3, 0x57, 0xb7, 0x02, 0xca,
        0x02, 0x65, 0x85, 0x4d, 0xc7, 0xea, 0x6a, 0xbd, 0xa6, 0xd7, 0x32, 0xce,
        0xe5, 0xeb, 0xa4, 0xf9, 0xf1, 0xfb, 0x76, 0x2a, 0x9d, 0xb5, 0x99, 0x4f,
        0xb5, 0x69, 0x06, 0x88, 0xe3, 0x3a, 0xcb, 0xd0, 0x43, 0x4b, 0x11, 0xd4,
        0x74, 0xbe, 0x6f, 0xf9, 0x50, 0x04, 0xba, 0x3e, 0x08, 0xeb, 0x0c, 0xba,
        0xcd, 0x57, 0x95, 0x1e, 0x0f, 0xd7, 0x42, 0x09, 0x9a, 0x4d, 0x20, 0x86,
        0x5a, 0xcc, 0xa0, 0xaa, 0x20, 0x40, 0x72, 0xd4, 0xe8, 0x39, 0xd5, 0x75,
        0x53, 0x73, 0x8b, 0x58, 0x4d, 0xe5, 0x2f, 0xb2, 0x7a, 0xc2, 0x59, 0xce,
        0xcf, 0x69, 0x8c, 0x19, 0x35, 0x8d, 0x0f, 0x18, 0x30, 0xda, 0xc0, 0x9d,
        0x4f, 0x9e, 0xfa, 0xec, 0xca, 0x39, 0x6c, 0xac, 0x02, 0xa7, 0x6a, 0xfd,
        0xa9, 0x10, 0x53, 0xd1, 0x78, 0x96, 0x68, 0x0a, 0x0b, 0xd5, 0x13, 0xb9,
        0x09, 0xa3, 0x14, 0x85, 0xd4, 0xb1, 0x9d, 0x3d, 0x7e, 0x2d, 0xbc, 0x81,
        0x18, 0x81, 0x0c, 0x81, 0x6f, 0xf5, 0xed, 0x05, 0x25, 0x7d, 0x12, 0xd1,
        0x1b, 0xc1, 0x4b, 0x5f, 0x5d, 0x3c, 0xfd, 0x0e, 0x76, 0xf0, 0x1d, 0x95,
        0xb7, 0x41, 0xbc, 0xdb, 0x3c, 0x34, 0x04, 0x83, 0xcc, 0xff, 0x67, 0xe0,
        0x48, 0x6e, 0x40, 0xb1, 0x40, 0xd8, 0xef, 0xe6, 0xed, 0x36, 0xe7, 0xe8,
        0xf8, 0x68, 0x71, 0x25, 0xb4, 0x96, 0x05, 0x73, 0xa4, 0x8a, 0x71, 0x36,
        0xca, 0x01, 0x6c, 0xfe, 0xd0, 0xd2, 0x36, 0xac, 0x0f, 0xff, 0x29, 0x94,
        0xf3, 0xcf, 0x9a, 0x36, 0x88, 0xf9, 0x02, 0x01, 0x03, 0x02, 0x82, 0x01,
        0x00, 0x7c, 0x92, 0x8c, 0xb1, 0x0c, 0x18, 0x90, 0xbb, 0x3a, 0x4e, 0x48,
        0x8e, 0x1a, 0x9f, 0xee, 0x68, 0x87, 0x37, 0xab, 0xd5, 0x4b, 0x4d, 0x5c,
        0x6a, 0xd4, 0xc7, 0x8d, 0x9a, 0xef, 0xf4, 0xcd, 0xde, 0xe8, 0xd3, 0xe9,
        0xc3, 0x63, 0xdd, 0x3a, 0x74, 0x6c, 0xa2, 0x3a, 0x7a, 0x01, 0xdc, 0x01,
        0x99, 0x03, 0x89, 0x2f, 0xf1, 0x9c, 0x7e, 0x6f, 0x3a, 0x21, 0xdf, 0x43,
        0xf2, 0x6d, 0xfb, 0xf6, 0xa7, 0xa4, 0x1c, 0x69, 0x23, 0xbb, 0x8a, 0x78,
        0xf0, 0xaf, 0x05, 0xec, 0xd1, 0xdd, 0x35, 0x82, 0x32, 0x0b, 0xe2, 0xf8,
```

```
0x7e, 0xf5, 0x50, 0xe0, 0x03, 0x26, 0xd4, 0x05, 0xf2, 0x08, 0x7c, 0x88,
0xe5, 0x0e, 0x14, 0x0a, 0x8f, 0x81, 0x5b, 0xbc, 0x33, 0x6b, 0x04, 0x3c,
0x88, 0x6b, 0x1c, 0x15, 0x80, 0x4c, 0x8d, 0xf0, 0x26, 0x8e, 0x4e, 0x37,
0xa2, 0x5c, 0xe5, 0x89, 0x43, 0x75, 0x21, 0xa7, 0x2b, 0x17, 0x8c, 0xa2,
0xfa, 0xb2, 0x7c, 0x0c, 0x74, 0x96, 0x44, 0x1c, 0x96, 0x3e, 0xd7, 0xd7,
0x94, 0xc1, 0x31, 0x3b, 0x96, 0x57, 0x7b, 0xc6, 0xdc, 0xac, 0x2c, 0x4c,
0x5a, 0xbc, 0xa7, 0x91, 0x63, 0x8f, 0x40, 0x3b, 0x0d, 0x7b, 0xb9, 0xc9,
0x56, 0xd5, 0x19, 0x81, 0xb2, 0x3f, 0x1e, 0xe4, 0xca, 0xdd, 0x91, 0x63,
0xfd, 0xf5, 0xbe, 0x43, 0x64, 0xa0, 0x1a, 0xbf, 0xa1, 0x52, 0x75, 0x81,
0x16, 0x58, 0x71, 0x7e, 0x9b, 0x10, 0x29, 0x2f, 0x4e, 0xc9, 0x13, 0x2f,
0xe6, 0x76, 0xdf, 0x86, 0x2a, 0xd6, 0x29, 0x33, 0x88, 0x83, 0x16, 0x2d,
0xd0, 0xf1, 0xf7, 0x37, 0xd7, 0x0d, 0x99, 0x40, 0x0b, 0x15, 0xbc, 0xe4,
0x84, 0x01, 0xbf, 0x21, 0xd2, 0xc5, 0x06, 0x62, 0xc4, 0xce, 0x88, 0xfa,
0xec, 0x23, 0xc9, 0x28, 0x75, 0x7d, 0x0b, 0xbc, 0xef, 0x3d, 0x83, 0x77,
0x06, 0x43, 0x30, 0x10, 0xab, 0x02, 0x81, 0x81, 0x00, 0xea, 0x72, 0x60,
0xa3, 0xfc, 0xff, 0x54, 0x7f, 0xc5, 0x67, 0x81, 0x00, 0xd4, 0x02, 0x91,
0x9b, 0xa3, 0x18, 0x22, 0x57, 0xfe, 0x92, 0xe5, 0x76, 0xfc, 0xc0, 0xbe,
0xb6, 0xa2, 0x88, 0xad, 0x07, 0x5d, 0xf9, 0xce, 0x1f, 0x37, 0xbe, 0xac,
0x91, 0xf2, 0x88, 0xf6, 0xe1, 0xec, 0x94, 0x93, 0x10, 0x9e, 0x4b, 0x38,
0xa7, 0x19, 0x43, 0x69, 0x55, 0x41, 0x2e, 0x18, 0x52, 0x4a, 0x3d, 0xd8,
0x08, 0xbb, 0xb0, 0x61, 0x3d, 0x61, 0xd5, 0xb2, 0x30, 0x25, 0x11, 0xb1,
0xd7, 0xe6, 0x3e, 0x9c, 0x15, 0x0e, 0xf0, 0x85, 0x10, 0xd9, 0x81, 0x9e,
0x3f, 0x36, 0xc4, 0xcf, 0x5c, 0xf1, 0xf3, 0x63, 0xb5, 0x14, 0x6e, 0xc8,
0x9c, 0x53, 0x88, 0xff, 0x87, 0x91, 0x43, 0x0e, 0x94, 0x75, 0xe7, 0xae,
0x0a, 0xc8, 0x43, 0x19, 0x89, 0xcc, 0xc1, 0xc0, 0xc4, 0x92, 0xb6, 0x11,
0xfe, 0xb6, 0xa4, 0xf8, 0x61, 0x02, 0x81, 0x81, 0x00, 0xcc, 0x09, 0x7a,
0x4d, 0x83, 0x5f, 0xce, 0x5e, 0x68, 0x4a, 0x84, 0xf8, 0x8e, 0x56, 0xfa,
0xa4, 0x36, 0x0a, 0xce, 0x7f, 0xea, 0xdf, 0x72, 0xe5, 0x6b, 0xfa, 0x77,
0xd1, 0x96, 0x4d, 0x71, 0x79, 0xb3, 0x2f, 0xe5, 0x21, 0xa2, 0x63, 0xaf,
0x0e, 0xe2, 0x56, 0x9b, 0x44, 0x51, 0xfa, 0x93, 0xec, 0xd1, 0xdb, 0xc9,
0xdd, 0x02, 0xa0, 0xa1, 0x89, 0xbb, 0xae, 0xed, 0xb8, 0xcc, 0xe3, 0x02,
0x17, 0x0b, 0x04, 0xbe, 0x17, 0x02, 0xfa, 0xfd, 0xc9, 0xca, 0xe7, 0x3d,
0x90, 0x24, 0x4d, 0x56, 0xde, 0xb4, 0x55, 0x7a, 0xa1, 0xc9, 0xbd, 0x65,
0x75, 0x9e, 0xf9, 0x9d, 0xb9, 0x69, 0x8d, 0xa9, 0x71, 0x32, 0xde, 0xd9,
0x06, 0x1a, 0xfe, 0x02, 0x52, 0x4c, 0xa7, 0x01, 0xce, 0xc5, 0x81, 0xa3,
0x94, 0x6f, 0x0d, 0xfa, 0x98, 0x2e, 0x58, 0xb3, 0xd3, 0xba, 0x99, 0xaf,
0x47, 0x7e, 0xc9, 0x77, 0x99, 0x02, 0x81, 0x81, 0x00, 0x9c, 0x4c, 0x40,
0x6d, 0x53, 0x54, 0xe2, 0xff, 0xd8, 0xef, 0xab, 0x55, 0xe2, 0xac, 0x61,
0x12, 0x6c, 0xba, 0xc1, 0x8f, 0xff, 0x0c, 0x98, 0xf9, 0xfd, 0xd5, 0xd4,
0x79, 0xc1, 0xb0, 0x73, 0x5a, 0x3e, 0xa6, 0x89, 0x6a, 0x25, 0x29, 0xc8,
0x61, 0x4c, 0x5b, 0x4f, 0x41, 0x48, 0x63, 0x0c, 0xb5, 0xbe, 0xdc, 0xd0,
0x6f, 0x66, 0x2c, 0xf0, 0xe3, 0x80, 0xc9, 0x65, 0x8c, 0x31, 0x7e, 0x90,
0x05, 0xd2, 0x75, 0x96, 0x28, 0xeb, 0xe3, 0xcc, 0x20, 0x18, 0xb6, 0x76,
0x8f, 0xee, 0xd4, 0x68, 0x0e, 0x09, 0xf5, 0xae, 0x0b, 0x3b, 0xab, 0xbe,
0xd4, 0xcf, 0x2d, 0xdf, 0x93, 0x4b, 0xf7, 0x97, 0xce, 0x0d, 0x9f, 0x30,
0x68, 0x37, 0xb0, 0xaa, 0x5a, 0x60, 0xd7, 0x5f, 0x0d, 0xa3, 0xef, 0xc9,
0x5c, 0x85, 0x82, 0x11, 0x06, 0x88, 0x81, 0x2b, 0x2d, 0xb7, 0x24, 0x0b,
0xff, 0x24, 0x6d, 0xfa, 0xeb, 0x02, 0x81, 0x81, 0x00, 0x88, 0x06, 0x51,
0x89, 0x02, 0x3f, 0xde, 0xe9, 0x9a, 0xdc, 0x58, 0xa5, 0xb4, 0x39, 0xfc,
0x6d, 0x79, 0x5c, 0x89, 0xaa, 0x9c, 0x94, 0xf7, 0x43, 0x9d, 0x51, 0xa5,
0x36, 0x64, 0x33, 0xa0, 0xfb, 0xcc, 0xca, 0x98, 0xc1, 0x16, 0xed, 0x1f,
0x5f, 0x41, 0x8f, 0x12, 0x2d, 0x8b, 0xfc, 0x62, 0x9d, 0xe1, 0x3d, 0x31,
0x3e, 0x01, 0xc0, 0x6b, 0xb1, 0x27, 0xc9, 0xf3, 0xd0, 0x88, 0x97, 0x56,
0xba, 0x07, 0x58, 0x7e, 0xba, 0x01, 0xfc, 0xa9, 0x31, 0x31, 0xef, 0x7e,
0x60, 0x18, 0x33, 0x8f, 0x3f, 0x22, 0xe3, 0xa7, 0x16, 0x86, 0x7e, 0x43,
0xa3, 0xbf, 0x51, 0x13, 0xd0, 0xf1, 0x09, 0x1b, 0xa0, 0xcc, 0x94, 0x90,
0xae, 0xbc, 0xa9, 0x56, 0xe1, 0x88, 0x6f, 0x56, 0x89, 0xd9, 0x01, 0x17,
0xb8, 0x4a, 0x09, 0x51, 0xba, 0xc9, 0x90, 0x77, 0xe2, 0x7c, 0x66, 0x74,
0xda, 0x54, 0x86, 0x4f, 0xbb, 0x02, 0x81, 0x81, 0x00, 0xbd, 0x98, 0x4a,
0xd6, 0xee, 0xb6, 0xe6, 0xab, 0x14, 0x80, 0x17, 0x4c, 0x59, 0x67, 0xc4,
0x62, 0x2d, 0x77, 0x89, 0x83, 0x31, 0x49, 0x5e, 0x5a, 0x45, 0x3a, 0x05,
0x93, 0x46, 0x30, 0x4c, 0x82, 0xcd, 0xe7, 0xcc, 0xbe, 0x12, 0x54, 0x34,
0x54, 0xe0, 0x47, 0x9a, 0x9b, 0x68, 0x12, 0x4e, 0x7b, 0x2d, 0xf2, 0x73,
0x61, 0x17, 0x4a, 0x69, 0x6c, 0xfc, 0x34, 0xdf, 0x63, 0x25, 0xf0, 0x33,
0x27, 0x20, 0x8b, 0xcd, 0x51, 0x49, 0xe7, 0x12, 0xff, 0x31, 0x64, 0x8b,
0xe6, 0x67, 0x9e, 0x5a, 0xd6, 0x0e, 0xfd, 0x29, 0xee, 0xf6, 0x6e, 0xca,
0x04, 0xfa, 0xd4, 0x07, 0x55, 0xbd, 0xc5, 0xdf, 0xeb, 0x50, 0xad, 0x98,
0x2a, 0x5c, 0x1a, 0x9c, 0x74, 0x4c, 0x8c, 0xfe, 0x04, 0xbe, 0x57, 0x67,
0xb1, 0xc4, 0xa7, 0xe9, 0xd5, 0xbe, 0x31, 0xbf, 0xf4, 0x3b, 0xbf, 0x7c,
0x7b, 0x32, 0x8c, 0x48, 0x23 };
```

```
std::vector<uint8_t> keyData(rsaKey, rsaKey + (sizeof(rsaKey)/sizeof(rsaKey[0])));
```

#### 4. Define parameters using CryptoParamBuilder for the key

```
std::shared_ptr<telux::sec::ICryptoParam> cp;

cp = telux::sec::CryptoParamBuilder()
        .setAlgorithm(telux::sec::Algorithm::ALGORITHM_RSA)
        .setCryptoOperation(telux::sec::CryptoOperation::CRYPTO_OP_SIGN |
                    telux::sec::CryptoOperation::CRYPTO_OP_VERIFY)
        .setDigest(telux::sec::Digest::DIGEST_SHA_2_256 |
                telux::sec::Digest::DIGEST_SHA_2_512)
        .setPadding(telux::sec::Padding::PADDING_RSA_PSS |
                telux::sec::Padding::PADDING_RSA_PKCS1_1_5_SIGN)
        .build();
```

#### 5. Import key by specifying format using importKey() API

```
telux::common::ErrorCode ec;
std::vector<uint8_t> kb;

ec = cryptMgr->importKey(cp, telux::sec::KeyFormat::KEY_FORMAT_PKCS8, keyData, kb);
if (ec != telux::common::ErrorCode::SUCCESS) {
    std::cout << "Can't import key, err: " <<
            static_cast<int>(ec) << std::endl;
    return -1;
}
```

## 3.12 Crypto accelerator

- Calculate ECQV point synchronously
- Calculate ECQV point asynchronously - poll mode
- Calculate ECQV point asynchronously - listener mode
- Verify ECDSA digest synchronously
- Verify ECDSA digest asynchronously - poll mode
- Verify ECDSA digest asynchronously - listener mode

### 3.12.1 Calculate ECQV point synchronously

This sample app demonstrates how to calculate ECQV point synchronously. The steps are:

1. Get SecurityFactory instance.
2. Get ICryptoAcceleratorManager instance from SecurityFactory.
3. Define parameters for calculation process.
4. Send parameters for calculation (method will block here). When the calculation is complete, get the result.

```
#include <iostream>

#include <telux/sec/SecurityFactory.hpp>

/* Scalar (hash construct) */
uint8_t scl[] = {
  0xd1, 0x07, 0x3b, 0x4e, 0xbf, 0x65, 0x0a, 0xfe, 0xff, 0x59, 0x7b, 0x1f,
  0x03, 0xe7, 0x51, 0xb4, 0x29, 0x6f, 0x6b, 0x3e, 0x12, 0xe4, 0xff, 0x31,
```

```
  0x61, 0xbb, 0x60, 0x5b, 0x0f, 0xa4, 0xc9, 0x39 };

/* Point to multiply (public key reconstruction value) */
uint8_t mulPointX[] = {
  0x79, 0xb3, 0x11, 0x42, 0xc1, 0xd8, 0x25, 0xcc, 0x17, 0xe5, 0xe0, 0xdd,
  0x75, 0xd1, 0xc2, 0x72, 0xb8, 0x7e, 0x7b, 0xd8, 0xe0, 0x21, 0x4a, 0xfc,
  0x32, 0x5d, 0xe3, 0xce, 0x83, 0x02, 0x7d, 0xa6 };
uint8_t mulPointY[] = {
  0xa5, 0x96, 0x93, 0x75, 0x7c, 0x9e, 0xb5, 0x91, 0xbc, 0xa6, 0x21, 0xbd,
  0xb7, 0x16, 0x03, 0xbc, 0x8f, 0xa6, 0xba, 0xc6, 0xd1, 0xde, 0x3d, 0xb0,
  0xf6, 0x8f, 0xb5, 0x7e, 0x93, 0x07, 0xa9, 0xa5 };

/* Point to add (CA public key) */
uint8_t addPointX[] = {
  0x5c, 0x48, 0x40, 0xb1, 0x67, 0xb6, 0xea, 0xb4, 0xc2, 0x79, 0x9b, 0xbe,
  0x32, 0x13, 0x7b, 0x4c, 0x68, 0xb5, 0xb6, 0x80, 0x11, 0x7b, 0x93, 0x4d,
  0x90, 0xce, 0x92, 0x1b, 0x1f, 0x94, 0x6d, 0xe9 };
uint8_t addPointY[] = {
  0x6f, 0xb1, 0x84, 0xe7, 0xcb, 0x35, 0xb2, 0x4a, 0x34, 0x5a, 0x7d, 0x40,
  0x29, 0x55, 0xa3, 0x0c, 0x5b, 0x7b, 0x59, 0x5f, 0x56, 0x98, 0xd7, 0x17,
  0xd6, 0x1c, 0x9d, 0x4c, 0x9f, 0x3c, 0xca, 0x40 };

/* Expected result of the calculation */
uint8_t outPointX[] = {
  0xa8, 0xfa, 0x30, 0x69, 0xb7, 0xf0, 0xe0, 0x8b, 0xe6, 0x31, 0x95, 0x8e,
  0xe4, 0x47, 0x2c, 0x7b, 0x0b, 0xf9, 0xa4, 0x68, 0x52, 0x96, 0xdc, 0x63,
  0x5c, 0x27, 0xc6, 0xd3, 0x4e, 0xc6, 0x2b, 0x9b };
uint8_t outPointY[] = {
  0x2d, 0x94, 0x35, 0xaa, 0xa6, 0x65, 0xec, 0xe0, 0x3f, 0x83, 0xad, 0x0a,
  0xa0, 0x41, 0x65, 0x4c, 0xe3, 0x43, 0x80, 0xc1, 0x35, 0x7e, 0xef, 0xc7,
  0x1a, 0xd8, 0x97, 0x80, 0x5b, 0x62, 0x74, 0xf3 };

int main(int argc, char **argv) {

    telux::common::ErrorCode ec;
    std::vector<uint8_t> resultData;
    std::shared_ptr<telux::sec::ICryptoAcceleratorManager> cryptAccelMgr;

    uint32_t uniqueId;
    telux::sec::ECCCurve curve;
    telux::sec::ECCPoint multiplicandPoint{};
    telux::sec::ECCPoint addendPoint{};
    telux::sec::Scalar scalar{};
    telux::sec::RequestPriority priority;

    /* Step - 1 */
    auto &secFact = telux::sec::SecurityFactory::getInstance();

    /* Step - 2 */
    cryptAccelMgr = secFact.getCryptoAcceleratorManager(ec, telux::sec::Mode::MODE_SYNC);
    if (!cryptAccelMgr) {
        std::cout <<
         "can't get ICryptoAcceleratorManager, err " << static_cast<int>(ec) << std::endl;
        return -ENOMEM;
    }

    /* Step - 3 */
    uniqueId = 1;
    curve = telux::sec::ECCCurve::CURVE_NISTP256;
    priority = telux::sec::RequestPriority::REQ_PRIORITY_NORMAL;
    scalar.scalar = scl;
    scalar.scalarLength = sizeof(scl)/sizeof(scl[0]);
    multiplicandPoint.x = mulPointX;
    multiplicandPoint.xLength = sizeof(mulPointX)/sizeof(mulPointX[0]);
    multiplicandPoint.y = mulPointY;
    multiplicandPoint.yLength = sizeof(mulPointY)/sizeof(mulPointY[0]);
    addendPoint.x = addPointX;
    addendPoint.xLength = sizeof(addPointX)/sizeof(addPointX[0]);
    addendPoint.y = addPointY;
    addendPoint.yLength = sizeof(addPointY)/sizeof(addPointY[0]);
```

```
    /* Step - 4 */
    ec = cryptAccelMgr->ecqvPointMultiplyAndAdd(multiplicandPoint, addendPoint,
            scalar, curve, uniqueId, priority, resultData);
    if (ec != telux::common::ErrorCode::SUCCESS) {
        std::cout << "calculation failed, " << static_cast<int>(ec) << std::endl;
        cryptAccelMgr = nullptr;
        return -1;
    }
    std::cout << "calculation done, " << static_cast<int>(ec) << std::endl;

    for (uint32_t x=0; x < resultData.size(); x++) {
        printf("%02x", resultData.at(x) & 0xffU);
        if ((x == 31) || (x == 63)) {
            printf("\n");
        }
    }
    printf("\n");

    return 0;
}
```

### 3.12.2 Calculate ECQV point asynchronously - poll mode

This sample app demonstrates how to calculate ECQV point and obtain result asynchronously. The steps are:

1. Get SecurityFactory instance.
2. Get ICryptoAcceleratorManager instance from SecurityFactory.
3. Define parameters for calculation process.
4. Send parameters for calculation.
5. Obtain the result by polling (method will block here).
6. Parse the result to extract calculation result and ECC point.
7. Finally, release ICryptoAcceleratorManager to release resources.

```
#include <iostream>

#include <telux/sec/SecurityFactory.hpp>

/* Scalar (hash construct) */
uint8_t scl[] = {
  0xd1, 0x07, 0x3b, 0x4e, 0xbf, 0x65, 0x0a, 0xfe, 0xff, 0x59, 0x7b, 0x1f,
  0x03, 0xe7, 0x51, 0xb4, 0x29, 0x6f, 0x6b, 0x3e, 0x12, 0xe4, 0xff, 0x31,
  0x61, 0xbb, 0x60, 0x5b, 0x0f, 0xa4, 0xc9, 0x39 };

/* Point to multiply (public key reconstruction value) */
uint8_t mulPointX[] = {
  0x79, 0xb3, 0x11, 0x42, 0xc1, 0xd8, 0x25, 0xcc, 0x17, 0xe5, 0xe0, 0xdd,
  0x75, 0xd1, 0xc2, 0x72, 0xb8, 0x7e, 0x7b, 0xd8, 0xe0, 0x21, 0x4a, 0xfc,
  0x32, 0x5d, 0xe3, 0xce, 0x83, 0x02, 0x7d, 0xa6 };
uint8_t mulPointY[] = {
  0xa5, 0x96, 0x93, 0x75, 0x7c, 0x9e, 0xb5, 0x91, 0xbc, 0xa6, 0x21, 0xbd,
  0xb7, 0x16, 0x03, 0xbc, 0x8f, 0xa6, 0xba, 0xc6, 0xd1, 0xde, 0x3d, 0xb0,
  0xf6, 0x8f, 0xb5, 0x7e, 0x93, 0x07, 0xa9, 0xa5 };

/* Point to add (CA public key) */
uint8_t addPointX[] = {
  0x5c, 0x48, 0x40, 0xb1, 0x67, 0xb6, 0xea, 0xb4, 0xc2, 0x79, 0x9b, 0xbe,
  0x32, 0x13, 0x7b, 0x4c, 0x68, 0xb5, 0xb6, 0x80, 0x11, 0x7b, 0x93, 0x4d,
  0x90, 0xce, 0x92, 0x1b, 0x1f, 0x94, 0x6d, 0xe9 };
uint8_t addPointY[] = {
  0x6f, 0xb1, 0x84, 0xe7, 0xcb, 0x35, 0xb2, 0x4a, 0x34, 0x5a, 0x7d, 0x40,
  0x29, 0x55, 0xa3, 0x0c, 0x5b, 0x7b, 0x59, 0x5f, 0x56, 0x98, 0xd7, 0x17,
```

```
  0xd6, 0x1c, 0x9d, 0x4c, 0x9f, 0x3c, 0xca, 0x40 };

/* Expected result of the calculation */
uint8_t outPointX[] = {
  0xa8, 0xfa, 0x30, 0x69, 0xb7, 0xf0, 0xe0, 0x8b, 0xe6, 0x31, 0x95, 0x8e,
  0xe4, 0x47, 0x2c, 0x7b, 0x0b, 0xf9, 0xa4, 0x68, 0x52, 0x96, 0xdc, 0x63,
  0x5c, 0x27, 0xc6, 0xd3, 0x4e, 0xc6, 0x2b, 0x9b };
uint8_t outPointY[] = {
  0x2d, 0x94, 0x35, 0xaa, 0xa6, 0x65, 0xec, 0xe0, 0x3f, 0x83, 0xad, 0x0a,
  0xa0, 0x41, 0x65, 0x4c, 0xe3, 0x43, 0x80, 0xc1, 0x35, 0x7e, 0xef, 0xc7,
  0x1a, 0xd8, 0x97, 0x80, 0x5b, 0x62, 0x74, 0xf3 };

int main(int argc, char **argv) {

    uint8_t *data;
    uint32_t numResultsRead = 0;
    telux::common::ErrorCode ec;
    std::vector<telux::sec::OperationResult> results(1);
    std::shared_ptr<telux::sec::ICryptoAccelerationManager> cryptAccelMgr;

    uint32_t uniqueId;
    telux::sec::ECCCurve curve;
    telux::sec::ECCPoint multiplicandPoint{};
    telux::sec::ECCPoint addendPoint{};
    telux::sec::Scalar scalar{};
    telux::sec::RequestPriority priority;

    /* Step - 1 */
    auto &secFact = telux::sec::SecurityFactory::getInstance();

    /* Step - 2 */
    cryptAccelMgr = secFact.getCryptoAccelerationManager(ec, telux::sec::Mode::MODE_ASYNC_POLL);
    if (!cryptAccelMgr) {
        std::cout <<
         "can't get ICryptoAccelerationManager, err " << static_cast<int>(ec) << std::endl;
        return -ENOMEM;
    }

    /* Step - 3 */
    uniqueId = 1;
    curve = telux::sec::ECCCurve::CURVE_NISTP256;
    priority = telux::sec::RequestPriority::REQ_PRIORITY_NORMAL;
    scalar.scalar = scl;
    scalar.scalarLength = sizeof(scl)/sizeof(scl[0]);
    multiplicandPoint.x = mulPointX;
    multiplicandPoint.xLength = sizeof(mulPointX)/sizeof(mulPointX[0]);
    multiplicandPoint.y = mulPointY;
    multiplicandPoint.yLength = sizeof(mulPointY)/sizeof(mulPointY[0]);
    addendPoint.x = addPointX;
    addendPoint.xLength = sizeof(addPointX)/sizeof(addPointX[0]);
    addendPoint.y = addPointY;
    addendPoint.yLength = sizeof(addPointY)/sizeof(addPointY[0]);

    /* Step - 4 */
    ec = cryptAccelMgr->ecqvPostDataForMultiplyAndAdd(multiplicandPoint, addendPoint,
            scalar, curve, uniqueId, priority);
    if (ec != telux::common::ErrorCode::SUCCESS) {
        std::cout << "request not sent, " << static_cast<int>(ec) << std::endl;
        return -EIO;
    }

    /* Step - 5 */
    ec = cryptAccelMgr->getAsyncResults(results, 1, -1, numResultsRead);
    if (ec != telux::common::ErrorCode::SUCCESS) {
        std::cout << "can't get result, " << static_cast<int>(ec) << std::endl;
        cryptAccelMgr = nullptr;
        return -EIO;
    }

    /* Step - 6 */
    std::cout << "uniqueId: " << telux::sec::ResultParser::getId(results[0]) << std::endl;
```

```
    std::cout << "operation type: " <<
        static_cast<int>(telux::sec::ResultParser::getOperationType(results[0])) << std::endl;
    std::cout << "error code: " <<
        static_cast<int>(telux::sec::ResultParser::getCAErrorCode(results[0])) << std::endl;

    data = telux::sec::ResultParser::getData(results[0]);
    for (uint32_t x=0; x < telux::sec::CA_RESULT_DATA_LENGTH; x++) {
        printf("%02x", data[x] & 0xffU);
        if ((x == 31) || (x == 63)) {
            printf("\n");
        }
    }
    printf("\n");

    /* Step - 7 */
    cryptAccelMgr = nullptr;

    return 0;
}
```

### 3.12.3 Calculate ECQV point asynchronously - listener mode

This sample app demonstrates how to calculate ECQV point and obtain result asynchronously. The steps are:

1. Define listener that will receive verification result.
2. Get SecurityFactory instance.
3. Get ICryptoAcceleratorManager instance from SecurityFactory.
4. Define parameters for verification process.
5. Send parameters for verification.
6. Receive result in the registered listener.

```
#include <iostream>
#include <chrono>
#include <thread>

#include <telux/sec/SecurityFactory.hpp>

/* Scalar (hash construct) */
uint8_t scl[] = {
  0xd1, 0x07, 0x3b, 0x4e, 0xbf, 0x65, 0x0a, 0xfe, 0xff, 0x59, 0x7b, 0x1f,
  0x03, 0xe7, 0x51, 0xb4, 0x29, 0x6f, 0x6b, 0x3e, 0x12, 0xe4, 0xff, 0x31,
  0x61, 0xbb, 0x60, 0x5b, 0x0f, 0xa4, 0xc9, 0x39 };

/* Point to multiply (public key reconstruction value) */
uint8_t mulPointX[] = {
  0x79, 0xb3, 0x11, 0x42, 0xc1, 0xd8, 0x25, 0xcc, 0x17, 0xe5, 0xe0, 0xdd,
  0x75, 0xd1, 0xc2, 0x72, 0xb8, 0x7e, 0x7b, 0xd8, 0xe0, 0x21, 0x4a, 0xfc,
  0x32, 0x5d, 0xe3, 0xce, 0x83, 0x02, 0x7d, 0xa6 };
uint8_t mulPointY[] = {
  0xa5, 0x96, 0x93, 0x75, 0x7c, 0x9e, 0xb5, 0x91, 0xbc, 0xa6, 0x21, 0xbd,
  0xb7, 0x16, 0x03, 0xbc, 0x8f, 0xa6, 0xba, 0xc6, 0xd1, 0xde, 0x3d, 0xb0,
  0xf6, 0x8f, 0xb5, 0x7e, 0x93, 0x07, 0xa9, 0xa5 };

/* Point to add (CA public key) */
uint8_t addPointX[] = {
  0x5c, 0x48, 0x40, 0xb1, 0x67, 0xb6, 0xea, 0xb4, 0xc2, 0x79, 0x9b, 0xbe,
  0x32, 0x13, 0x7b, 0x4c, 0x68, 0xb5, 0xb6, 0x80, 0x11, 0x7b, 0x93, 0x4d,
  0x90, 0xce, 0x92, 0x1b, 0x1f, 0x94, 0x6d, 0xe9 };
uint8_t addPointY[] = {
  0x6f, 0xb1, 0x84, 0xe7, 0xcb, 0x35, 0xb2, 0x4a, 0x34, 0x5a, 0x7d, 0x40,
  0x29, 0x55, 0xa3, 0x0c, 0x5b, 0x7b, 0x59, 0x5f, 0x56, 0x98, 0xd7, 0x17,
  0xd6, 0x1c, 0x9d, 0x4c, 0x9f, 0x3c, 0xca, 0x40 };
```

```
/* Expected result of the calculation */
uint8_t outPointX[] = {
  0xa8, 0xfa, 0x30, 0x69, 0xb7, 0xf0, 0xe0, 0x8b, 0xe6, 0x31, 0x95, 0x8e,
  0xe4, 0x47, 0x2c, 0x7b, 0x0b, 0xf9, 0xa4, 0x68, 0x52, 0x96, 0xdc, 0x63,
  0x5c, 0x27, 0xc6, 0xd3, 0x4e, 0xc6, 0x2b, 0x9b };
uint8_t outPointY[] = {
  0x2d, 0x94, 0x35, 0xaa, 0xa6, 0x65, 0xec, 0xe0, 0x3f, 0x83, 0xad, 0x0a,
  0xa0, 0x41, 0x65, 0x4c, 0xe3, 0x43, 0x80, 0xc1, 0x35, 0x7e, 0xef, 0xc7,
  0x1a, 0xd8, 0x97, 0x80, 0x5b, 0x62, 0x74, 0xf3 };

class ResultListener : public telux::sec::ICryptoAcceleratorListener {

    /* Step - 6 */
    void onCalculationResult(uint32_t uniqueId, telux::common::ErrorCode ec,
        std::vector<uint8_t> resultData) {

        if (ec != telux::common::ErrorCode::SUCCESS) {
            std::cout <<
                "calculation failed, err: " << static_cast<int>(ec) <<
                " uniqueId: " << uniqueId << std::endl;
            return;
        }

        std::cout << "calculation done, uniqueId: " << uniqueId << std::endl;

        uint8_t *data = resultData.data();
        for (uint32_t x = 0; x < telux::sec::CA_RESULT_DATA_LENGTH; x++) {
            printf("%02x", data[x] & 0xffU);
            if ((x == 31) || (x == 63)) {
                printf("\n");
            }
        }
        printf("\n");
    }

    void onVerificationResult(uint32_t uniqueId,
        telux::common::ErrorCode ec, std::vector<uint8_t> resultData) {
    }
};

int main(int argc, char **argv) {

    telux::common::ErrorCode ec;
    std::shared_ptr<ResultListener> resultListener;
    std::shared_ptr<telux::sec::ICryptoAcceleratorManager> cryptAccelMgr;

    uint32_t uniqueId;
    telux::sec::ECCCurve curve;
    telux::sec::ECCPoint multiplicandPoint{};
    telux::sec::ECCPoint addendPoint{};
    telux::sec::Scalar scalar{};
    telux::sec::RequestPriority priority;

    /* Step - 1 */
    resultListener = std::make_shared<ResultListener>();

    /* Step - 2 */
    auto &secFact = telux::sec::SecurityFactory::getInstance();

    /* Step - 3 */
    cryptAccelMgr = secFact.getCryptoAcceleratorManager(ec,
        telux::sec::Mode::MODE_ASYNC_LISTENER, resultListener);
    if (!cryptAccelMgr) {
        std::cout <<
          "can't get ICryptoAcceleratorManager, err " << static_cast<int>(ec) << std::endl;
        return -ENOMEM;
    }

    /* Step - 4 */
    uniqueId = 1;
```

```
    curve = telux::sec::ECCCurve::CURVE_NISTP256;
    priority = telux::sec::RequestPriority::REQ_PRIORITY_NORMAL;
    scalar.scalar = scl;
    scalar.scalarLength = sizeof(scl)/sizeof(scl[0]);
    multiplicandPoint.x = mulPointX;
    multiplicandPoint.xLength = sizeof(mulPointX)/sizeof(mulPointX[0]);
    multiplicandPoint.y = mulPointY;
    multiplicandPoint.yLength = sizeof(mulPointY)/sizeof(mulPointY[0]);
    addendPoint.x = addPointX;
    addendPoint.xLength = sizeof(addPointX)/sizeof(addPointX[0]);
    addendPoint.y = addPointY;
    addendPoint.yLength = sizeof(addPointY)/sizeof(addPointY[0]);

    /* Step - 5 */
    ec = cryptAccelMgr->ecqvPostDataForMultiplyAndAdd(multiplicandPoint, addendPoint,
            scalar, curve, uniqueId, priority);
    if (ec != telux::common::ErrorCode::SUCCESS) {
        std::cout << "request not sent, " << static_cast<int>(ec) << std::endl;
        return -EIO;
    }

    /* Let result become available, listener invoked, before we exit the application */
    std::this_thread::sleep_for(std::chrono::milliseconds(2000));

    return 0;
}
```

### 3.12.4 Verify ECDSA digest synchronously

This sample app demonstrates how to verify ECDSA digest synchronously. The steps are:

1. Get SecurityFactory instance.
2. Get ICryptoAcceleratorManager instance from SecurityFactory.
3. Define parameters for verification process.
4. Send parameters for verification (method will block here). When the digest is verified, get the result.

```
#include <iostream>

#include <telux/sec/SecurityFactory.hpp>

/* Digest to verify */
uint8_t dig[] = {
    0x67, 0x45, 0x8b, 0x6b, 0xc6, 0x23, 0x7b, 0x32, 0x69, 0x98, 0x3c, 0x64,
    0x73, 0x48, 0x33, 0x66, 0x51, 0xdc, 0xb0, 0x74, 0xff, 0x5c, 0x49, 0x19,
    0x4a, 0x94, 0xe8, 0x2a, 0xec, 0x58, 0x55, 0x62 };

/* Public key */
uint8_t pubKeyX[] = {
    0x62, 0xd5, 0xe2, 0x2a, 0xff, 0x7a, 0x60, 0x27, 0xe9, 0x0a, 0xd1, 0x0e,
    0x01, 0xa1, 0x3c, 0x23, 0x01, 0xa5, 0x02, 0xa3, 0x79, 0xf9, 0x99, 0x0b,
    0xf3, 0x8e, 0xec, 0xb3, 0x15, 0x0a, 0xb2, 0x3b };
uint8_t pubKeyY[] ={
    0xa7, 0x2f, 0xaf, 0xeb, 0xbc, 0x72, 0xaf, 0xc2, 0x7c, 0x57, 0x82, 0x0e,
    0x9f, 0xef, 0xe2, 0xe9, 0xbd, 0x6c, 0x52, 0x29, 0x1d, 0x85, 0xa4, 0xdf,
    0xe1, 0xaf, 0x17, 0x14, 0xec, 0x00, 0x27, 0x90 };

/* Signature of digest */
uint8_t rSig[] = {
    0xfa, 0xc3, 0x51, 0xad, 0xe4, 0x4e, 0x7a, 0xf9, 0x52, 0xfd, 0x0a, 0x93,
    0x61, 0xc2, 0x8e, 0x32, 0x3c, 0x13, 0x45, 0xa6, 0x60, 0x6a, 0x1c, 0x85,
    0x1c, 0x73, 0x5c, 0x78, 0x0f, 0x16, 0xd4, 0x51 };
uint8_t sSig[] = {
    0x42, 0x82, 0x47, 0xd5, 0xab, 0xe4, 0xae, 0x3f, 0x42, 0xe8, 0x11, 0xac,
    0x04, 0x88, 0x73, 0xe4, 0x04, 0xa1, 0x8c, 0xa8, 0x80, 0x1b, 0x65, 0xdb,
    0x38, 0xb1, 0xb6, 0x10, 0x12, 0x6a, 0x78, 0xd2 };
```

```
int main(int argc, char **argv) {

    uint8_t *data;
    telux::common::ErrorCode ec;
    std::vector<uint8_t> resultData;
    std::shared_ptr<telux::sec::ICryptoAcceleratorManager> cryptAccelMgr;

    uint32_t uniqueId;
    telux::sec::ECCCurve curve;
    telux::sec::DataDigest digest{};
    telux::sec::ECCPoint publicKey{};
    telux::sec::Signature signature{};
    telux::sec::RequestPriority priority;

    /* Step - 1 */
    auto &secFact = telux::sec::SecurityFactory::getInstance();

    /* Step - 2 */
    cryptAccelMgr = secFact.getCryptoAcceleratorManager(ec, telux::sec::Mode::MODE_SYNC);
    if (!cryptAccelMgr) {
        std::cout <<
         "can't get ICryptoAcceleratorManager, err " << static_cast<int>(ec) << std::endl;
        return -ENOMEM;
    }

    /* Step - 3 */
    uniqueId = 1;
    curve = telux::sec::ECCCurve::CURVE_NISTP256;
    priority = telux::sec::RequestPriority::REQ_PRIORITY_NORMAL;
    digest.digest = dig;
    digest.digestLength = sizeof(dig)/sizeof(dig[0]);
    publicKey.x = pubKeyX;
    publicKey.xLength = sizeof(pubKeyX)/sizeof(pubKeyX[0]);
    publicKey.y = pubKeyY;
    publicKey.yLength = sizeof(pubKeyY)/sizeof(pubKeyY[0]);
    signature.rSignature = rSig;
    signature.sSignature = sSig;
    signature.rsLength = sizeof(rSig)/sizeof(rSig[0]);

    /* Step - 4 */
    ec = cryptAccelMgr->eccVerifyDigest(digest, publicKey, signature,
            curve, uniqueId, priority, resultData);
    if (ec != telux::common::ErrorCode::SUCCESS) {
        std::cout << "verification failed, " << static_cast<int>(ec) << std::endl;
        cryptAccelMgr = nullptr;
        return -1;
    }
    std::cout << "verification passed" << std::endl;

    data = resultData.data();
    for (uint32_t x=0; x < telux::sec::CA_RESULT_DATA_LENGTH; x++) {
        printf("%02x", data[x] & 0xffU);
        if ((x == 31) || (x == 63)) {
            printf("\n");
        }
    }
    printf("\n");

    return 0;
}
```

### 3.12.5 Verify ECDSA digest asynchronously - poll mode

This sample app demonstrates how to verify ECDSA digest asynchronously. The steps are:

1. Get SecurityFactory instance.
2. Get ICryptoAcceleratorManager instance from SecurityFactory.

3. Define parameters for calculation process.
4. Send parameters for calculation.
5. Obtain the result by polling (method will block here).
6. Parse the result to extract calculation result and ECC point.
7. Finally, release ICryptoAcceleratorManager to release resources.

```
#include <iostream>

#include <telux/sec/SecurityFactory.hpp>

/* Digest to verify */
uint8_t dig[] = {
    0x67, 0x45, 0x8b, 0x6b, 0xc6, 0x23, 0x7b, 0x32, 0x69, 0x98, 0x3c, 0x64,
    0x73, 0x48, 0x33, 0x66, 0x51, 0xdc, 0xb0, 0x74, 0xff, 0x5c, 0x49, 0x19,
    0x4a, 0x94, 0xe8, 0x2a, 0xec, 0x58, 0x55, 0x62 };

/* Public key */
uint8_t pubKeyX[] = {
    0x62, 0xd5, 0xe2, 0x2a, 0xff, 0x7a, 0x60, 0x27, 0xe9, 0x0a, 0xd1, 0x0e,
    0x01, 0xa1, 0x3c, 0x23, 0x01, 0xa5, 0x02, 0xa3, 0x79, 0xf9, 0x99, 0x0b,
    0xf3, 0x8e, 0xec, 0xb3, 0x15, 0x0a, 0xb2, 0x3b };
uint8_t pubKeyY[] ={
    0xa7, 0x2f, 0xaf, 0xeb, 0xbc, 0x72, 0xaf, 0xc2, 0x7c, 0x57, 0x82, 0x0e,
    0x9f, 0xef, 0xe2, 0xe9, 0xbd, 0x6c, 0x52, 0x29, 0x1d, 0x85, 0xa4, 0xdf,
    0xe1, 0xaf, 0x17, 0x14, 0xec, 0x00, 0x27, 0x90 };

/* Signature of digest */
uint8_t rSig[] = {
    0xfa, 0xc3, 0x51, 0xad, 0xe4, 0x4e, 0x7a, 0xf9, 0x52, 0xfd, 0x0a, 0x93,
    0x61, 0xc2, 0x8e, 0x32, 0x3c, 0x13, 0x45, 0xa6, 0x60, 0x6a, 0x1c, 0x85,
    0x1c, 0x73, 0x5c, 0x78, 0x0f, 0x16, 0xd4, 0x51 };
uint8_t sSig[] = {
    0x42, 0x82, 0x47, 0xd5, 0xab, 0xe4, 0xae, 0x3f, 0x42, 0xe8, 0x11, 0xac,
    0x04, 0x88, 0x73, 0xe4, 0x04, 0xa1, 0x8c, 0xa8, 0x80, 0x1b, 0x65, 0xdb,
    0x38, 0xb1, 0xb6, 0x10, 0x12, 0x6a, 0x78, 0xd2 };

int main(int argc, char **argv) {

    uint8_t *data;
    uint32_t numResultsRead = 0;
    telux::common::ErrorCode ec;
    std::vector<telux::sec::OperationResult> results(1);
    std::shared_ptr<telux::sec::ICryptoAcceleratorManager> cryptAccelMgr;

    uint32_t uniqueId;
    telux::sec::ECCCurve curve;
    telux::sec::DataDigest digest{};
    telux::sec::ECCPoint publicKey{};
    telux::sec::Signature signature{};
    telux::sec::RequestPriority priority;

    /* Step - 1 */
    auto &secFact = telux::sec::SecurityFactory::getInstance();

    /* Step - 2 */
    cryptAccelMgr = secFact.getCryptoAcceleratorManager(ec, telux::sec::Mode::MODE_ASYNC_POLL);
    if (!cryptAccelMgr) {
        std::cout <<
         "can't get ICryptoAcceleratorManager, err " << static_cast<int>(ec) << std::endl;
        return -ENOMEM;
    }

    /* Step - 3 */
    uniqueId = 1;
    curve = telux::sec::ECCCurve::CURVE_NISTP256;
    priority = telux::sec::RequestPriority::REQ_PRIORITY_NORMAL;
    digest.digest = dig;
```

```
    digest.digestLength = sizeof(dig)/sizeof(dig[0]);
    publicKey.x = pubKeyX;
    publicKey.xLength = sizeof(pubKeyX)/sizeof(pubKeyX[0]);
    publicKey.y = pubKeyY;
    publicKey.yLength = sizeof(pubKeyY)/sizeof(pubKeyY[0]);
    signature.rSignature = rSig;
    signature.sSignature = sSig;
    signature.rsLength = sizeof(rSig)/sizeof(rSig[0]);

    /* Step - 4 */
    ec = cryptAccelMgr->eccPostDigestForVerification(digest, publicKey, signature,
            curve, uniqueId, priority);
    if (ec != telux::common::ErrorCode::SUCCESS) {
        std::cout << "request not sent, " << static_cast<int>(ec) << std::endl;
        return -EIO;
    }

    /* Step - 5 */
    ec = cryptAccelMgr->getAsyncResults(results, 1, -1, numResultsRead);
    if (ec != telux::common::ErrorCode::SUCCESS) {
        std::cout << "can't get result, " << static_cast<int>(ec) << std::endl;
        cryptAccelMgr = nullptr;
        return -EIO;
    }

    /* Step - 6 */
    std::cout << "uniqueId: " << telux::sec::ResultParser::getId(results[0]) << std::endl;
    std::cout << "operation type: " <<
        static_cast<int>(telux::sec::ResultParser::getOperationType(results[0])) << std::endl;
    std::cout << "error code: " <<
        static_cast<int>(telux::sec::ResultParser::getCAErrorCode(results[0])) << std::endl;

    if (telux::sec::ResultParser::getCAErrorCode(results[0]) !=
            telux::common::ErrorCode::SUCCESS) {
        std::cout << "verification failed" << std::endl;
        return -1;
    } else {
        std::cout << "verification passed" << std::endl;
    }

    data = telux::sec::ResultParser::getData(results[0]);
    for (uint32_t x=0; x < telux::sec::CA_RESULT_DATA_LENGTH; x++) {
        printf("%02x", data[x] & 0xffU);
        if ((x == 31) || (x == 63)) {
            printf("\n");
        }
    }
    printf("\n");

    /* Step - 7 */
    cryptAccelMgr = nullptr;

    return 0;
}
```

### 3.12.6 Verify ECDSA digest asynchronously - listener mode

This sample app demonstrates how to verify ECDSA digest asynchronously. The steps are:

1. Define listener that will receive verification result.
2. Get SecurityFactory instance.
3. Get ICryptoAcceleratorManager instance from SecurityFactory.
4. Define parameters for verification process.
5. Send parameters for verification.

6. Receive result in the registered listener.

```
#include <iostream>
#include <chrono>
#include <thread>

#include <telux/sec/SecurityFactory.hpp>

/* Digest to verify */
uint8_t dig[] = {
    0x67, 0x45, 0x8b, 0x6b, 0xc6, 0x23, 0x7b, 0x32, 0x69, 0x98, 0x3c, 0x64,
    0x73, 0x48, 0x33, 0x66, 0x51, 0xdc, 0xb0, 0x74, 0xff, 0x5c, 0x49, 0x19,
    0x4a, 0x94, 0xe8, 0x2a, 0xec, 0x58, 0x55, 0x62 };

/* Public key */
uint8_t pubKeyX[] = {
    0x62, 0xd5, 0xe2, 0x2a, 0xff, 0x7a, 0x60, 0x27, 0xe9, 0x0a, 0xd1, 0x0e,
    0x01, 0xa1, 0x3c, 0x23, 0x01, 0xa5, 0x02, 0xa3, 0x79, 0xf9, 0x99, 0x0b,
    0xf3, 0x8e, 0xec, 0xb3, 0x15, 0x0a, 0xb2, 0x3b };
uint8_t pubKeyY[] ={
    0xa7, 0x2f, 0xaf, 0xeb, 0xbc, 0x72, 0xaf, 0xc2, 0x7c, 0x57, 0x82, 0x0e,
    0x9f, 0xef, 0xe2, 0xe9, 0xbd, 0x6c, 0x52, 0x29, 0x1d, 0x85, 0xa4, 0xdf,
    0xe1, 0xaf, 0x17, 0x14, 0xec, 0x00, 0x27, 0x90 };

/* Signature of digest */
uint8_t rSig[] = {
    0xfa, 0xc3, 0x51, 0xad, 0xe4, 0x4e, 0x7a, 0xf9, 0x52, 0xfd, 0x0a, 0x93,
    0x61, 0xc2, 0x8e, 0x32, 0x3c, 0x13, 0x45, 0xa6, 0x60, 0x6a, 0x1c, 0x85,
    0x1c, 0x73, 0x5c, 0x78, 0x0f, 0x16, 0xd4, 0x51 };
uint8_t sSig[] = {
    0x42, 0x82, 0x47, 0xd5, 0xab, 0xe4, 0xae, 0x3f, 0x42, 0xe8, 0x11, 0xac,
    0x04, 0x88, 0x73, 0xe4, 0x04, 0xa1, 0x8c, 0xa8, 0x80, 0x1b, 0x65, 0xdb,
    0x38, 0xb1, 0xb6, 0x10, 0x12, 0x6a, 0x78, 0xd2 };

class ResultListener : public telux::sec::ICryptoAcceleratorListener {

    /* Step - 6 */
    void onVerificationResult(uint32_t uniqueId, telux::common::ErrorCode ec,
        std::vector<uint8_t> resultData) {

        if (ec != telux::common::ErrorCode::SUCCESS) {
            std::cout <<
                "verification failed, err: " << static_cast<int>(ec) <<
                " uniqueId: " << uniqueId << std::endl;
            return;
        }

        std::cout << "verification passed, uniqueId: " << uniqueId << std::endl;

        uint8_t *data = resultData.data();
        for (uint32_t x = 0; x < telux::sec::CA_RESULT_DATA_LENGTH; x++) {
            printf("%02x", data[x] & 0xffU);
            if ((x == 31) || (x == 63)) {
                printf("\n");
            }
        }
        printf("\n");
    }

    void onCalculationResult(uint32_t uniqueId, telux::common::ErrorCode ec,
        std::vector<uint8_t> resultData) {
    }
};

int main(int argc, char **argv) {

    telux::common::ErrorCode ec;
    std::shared_ptr<ResultListener> resultListener;
    std::shared_ptr<telux::sec::ICryptoAcceleratorManager> cryptAccelMgr;

    uint32_t uniqueId;
```

```
    telux::sec::ECCCurve curve;
    telux::sec::DataDigest digest{};
    telux::sec::ECCPoint publicKey{};
    telux::sec::Signature signature{};
    telux::sec::RequestPriority priority;

    /* Step - 1 */
    resultListener = std::make_shared<ResultListener>();

    /* Step - 2 */
    auto &secFact = telux::sec::SecurityFactory::getInstance();

    /* Step - 3 */
    cryptAccelMgr = secFact.getCryptoAcceleratorManager(ec,
        telux::sec::Mode::MODE_ASYNC_LISTENER, resultListener);
    if (!cryptAccelMgr) {
        std::cout <<
         "can't get ICryptoAcceleratorManager, err " << static_cast<int>(ec) << std::endl;
        return -ENOMEM;
    }

    /* Step - 4 */
    uniqueId = 1;
    curve = telux::sec::ECCCurve::CURVE_NISTP256;
    priority = telux::sec::RequestPriority::REQ_PRIORITY_NORMAL;
    digest.digest = dig;
    digest.digestLength = sizeof(dig)/sizeof(dig[0]);
    publicKey.x = pubKeyX;
    publicKey.xLength = sizeof(pubKeyX)/sizeof(pubKeyX[0]);
    publicKey.y = pubKeyY;
    publicKey.yLength = sizeof(pubKeyY)/sizeof(pubKeyY[0]);
    signature.rSignature = rSig;
    signature.sSignature = sSig;
    signature.rsLength = sizeof(rSig)/sizeof(rSig[0]);

    /* Step - 5 */
    ec = cryptAccelMgr->eccPostDigestForVerification(digest, publicKey, signature,
            curve, uniqueId, priority);
    if (ec != telux::common::ErrorCode::SUCCESS) {
        std::cout << "request not sent, " << static_cast<int>(ec) << std::endl;
    }

    /* Let result become available, listener invoked, before we exit the application */
    std::this_thread::sleep_for(std::chrono::milliseconds(5000));

    return 0;
}
```

## 3.13 Cellular connection security

- Receive cellular security reports

### 3.13.1 Receive cellular security reports

Steps to register listener and receive cellular connection security reports are:

1. Define listener that will receive report.
2. Get ConnectionSecurityFactory instance.
3. Get ICellularSecurityManager instance from ConnectionSecurityFactory.
4. Register listener to receive security reports.
5. Receive reports in the registered listener.

6. Optional, implement onServiceStatusChange() if SSR updates are required.
7. When the use-case is complete, deregister the listener.

```
#include <iostream>
#include <chrono>
#include <thread>

#include <telux/sec/ConnectionSecurityFactory.hpp>

class CellSecurityReportListener : public telux::sec::ICellularScanReportListener {

    /* Step - 5 */
    void onScanReportAvailable(telux::sec::CellularSecurityReport report,
            telux::sec::EnvironmentInfo envInfo) {

        std::cout << "Threat score: " << static_cast<uint32_t>(report.threatScore) << std::endl;
        std::cout << "Cell ID     : " << static_cast<uint32_t>(report.cellId) << std::endl;
        std::cout << "PID         : " << static_cast<uint32_t>(report.pid) << std::endl;
        std::cout << "MCC         : " << report.mcc << std::endl;
        std::cout << "MNC         : " << report.mnc << std::endl;
        std::cout << "Action type : " << static_cast<uint32_t>(report.actionType) << std::endl;
        std::cout << "RAT         : " << static_cast<uint32_t>(report.rat) << std::endl;

        for (size_t x = 0; x < report.threatTypesDetected.size(); x++) {
            std::cout << "Threat type : " <<
                static_cast<uint32_t>(report.threatTypesDetected[x]) << std::endl;
        }
    }

    /* Step - 6 */
    void onServiceStatusChange(telux::common::ServiceStatus newStatus) {

        std::cout << "New status: " << static_cast<uint32_t>(newStatus) << std::endl;
    }
};

int main(int argc, char **argv) {

    telux::common::ErrorCode ec;
    std::shared_ptr<CellSecurityReportListener> reportListener;
    std::shared_ptr<telux::sec::ICellularSecurityManager> cellConSecMgr;

    /* Step - 1 */
    try {
        reportListener = std::make_shared<CellSecurityReportListener>();
    } catch (const std::exception& e) {
        std::cout << "can't allocate CellSecurityReportListener" << std::endl;
        return -ENOMEM;
    }

    /* Step - 2 */
    auto &cellConSecFact = telux::sec::ConnectionSecurityFactory::getInstance();

    /* Step - 3 */
    cellConSecMgr = cellConSecFact.getCellularSecurityManager(ec);
    if (!cellConSecMgr) {
        std::cout <<
         "can't get ICellularSecurityManager, err " << static_cast<int>(ec) << std::endl;
        return -ENOMEM;
    }

    /* Step - 4 */
    ec = cellConSecMgr->registerListener(reportListener);
    if (ec != telux::common::ErrorCode::SUCCESS) {
        std::cout << "can't register listener, err " << static_cast<int>(ec) << std::endl;
        return -EIO;
    }

    /* Add application specific business logic here */
    std::this_thread::sleep_for(std::chrono::milliseconds(10000));
```

```
    /* Step - 7 */
    ec = cellConSecMgr->deRegisterListener(reportListener);
    if (ec != telux::common::ErrorCode::SUCCESS) {
        std::cout << "can't deregister listener, err " << static_cast<int>(ec) << std::endl;
        return -EIO;
    }

    return 0;
}
```

## 3.14 WiFi connection security

- Receive wifi security reports

### 3.14.1 Receive wifi security reports

Steps to register listener and receive wifi connection security reports are:

1. Define listener that will receive report and receive invocation for consent to trusting an AP.
2. Get ConnectionSecurityFactory instance.
3. Get IWiFiSecurityManager instance from ConnectionSecurityFactory.
4. Register listener to receive security reports.
5. Receive reports in the registered listener.
6. When the use-case is complete, deregister the listener.

```
#include <iostream>
#include <chrono>
#include <thread>

#include <telux/sec/ConnectionSecurityFactory.hpp>

class WiFiSecurityReportListener : public telux::sec::IWiFiReportListener {

    /* Step - 5 */
    void onReportAvailable(telux::sec::WiFiSecurityReport report) {
        std::cout << "ssid             : " << report.ssid << std::endl;
        std::cout << "bssid            : " << report.bssid << std::endl;
        std::cout << "is connected     : " << report.isConnectedToAP << std::endl;
        std::cout << "is open          : " << report.isOpenAP << std::endl;
        std::cout << "ml threat score  : " <<
            report.mlAlgorithmAnalysis.threatScore << std::endl;
        std::cout << "ml result        : " <<
            static_cast<int>(report.mlAlgorithmAnalysis.result) << std::endl;
        std::cout << "summoning result : " <<
            static_cast<int>(report.summoningAnalysis.result) << std::endl;
    }

    void onDeauthenticationAttack(telux::sec::DeauthenticationInfo deauthenticationInfo) {
        std::cout << "disconnect reason : " <<
            deauthenticationInfo.deauthenticationReason << std::endl;
        std::cout << "did AP initiated  : " <<
            deauthenticationInfo.didAPInitiateDisconnect << std::endl;
        std::cout << "threat score      : " <<
            deauthenticationInfo.threatScore << std::endl;
    }

    void isTrustedAP(std::string ssid, bool& isTrusted) {
        /* In this example we always trust the AP */
        isTrusted = true;
```

```
    }
};

int main(int argc, char **argv) {

    telux::common::ErrorCode ec;
    std::shared_ptr<WiFiSecurityReportListener> reportListener;
    std::shared_ptr<telux::sec::IWiFiSecurityManager> wifiConSecMgr;

    /* Step - 1 */
    try {
        reportListener = std::make_shared<WiFiSecurityReportListener>();
    } catch (const std::exception& e) {
        std::cout << "can't allocate WiFiSecurityReportListener" << std::endl;
        return -ENOMEM;
    }

    /* Step - 2 */
    auto &wifiConSecFact = telux::sec::ConnectionSecurityFactory::getInstance();

    /* Step - 3 */
    wifiConSecMgr = wifiConSecFact.getWiFiSecurityManager(ec);
    if (!wifiConSecMgr) {
        std::cout <<
         "can't get IWiFiSecurityManager, err " << static_cast<int>(ec) << std::endl;
        return -ENOMEM;
    }

    /* Step - 4 */
    ec = wifiConSecMgr->registerListener(reportListener);
    if (ec != telux::common::ErrorCode::SUCCESS) {
        std::cout << "can't register listener, err " << static_cast<int>(ec) << std::endl;
        return -EIO;
    }

    /* Add application specific business logic here */
    std::this_thread::sleep_for(std::chrono::milliseconds(10000));

    /* Step - 6 */
    ec = wifiConSecMgr->deRegisterListener(reportListener);
    if (ec != telux::common::ErrorCode::SUCCESS) {
        std::cout << "can't deregister listener, err " << static_cast<int>(ec) << std::endl;
        return -EIO;
    }

    return 0;
}
```

## 3.15 WLAN

- wlan_config
- wlan_sta_config

## 3.16 Diagnostics

## The List of sample apps related to diagnostics:

- # How to configure, start and stop logging

### 3.16.1 # How to configure, start and stop logging

This sample application demonstrates how to configure, start and stop diagnostics logging.

#### 1. Create the initialization callback object to be notified when initialization is complete and the object is ready to be used.

```
auto initCb = [&](telux::common::ServiceStatus status) {
   initPromise.set_value(status);
};
```

#### 2. Get the DiagnosticsFactory and DiagLogManager instances.

```
auto &diagFactory = telux::platform::diag::DiagnosticsFactory::getInstance();
diagMgr  = diagFactory.getDiagLogManager(initCb);
```

#### 3. Ensure subsystem is ready.

```
subSystemStatus = initPromise.get_future().get();
```

#### 4. Ensure logging is not currently in progress.

```
if(diagStatus.isLoggingInProgress) {
   std::cout << " *** Logging already in progress - quitting application *** " << std::endl;
   break;
}
```

#### 5. Configure diagnostics logging.

```
telux::platform::diag::DiagConfig fileMethodCfg {};
fileMethodCfg.method = telux::platform::diag::LogMethod::FILE;
fileMethodCfg.srcType = telux::platform::diag::SourceType::DEVICE;
fileMethodCfg.srcInfo.device = static_cast<uint8_t>(2);
fileMethodCfg.mdmLogMaskFile = maskFile;
fileMethodCfg.modeType = telux::platform::diag::DiagLogMode::STREAMING;
fileMethodCfg.methodConfig.fileConfig.maxSize = 0;
fileMethodCfg.methodConfig.fileConfig.maxNumber = 0;
errCode = diagMgr->setConfig(fileMethodCfg);
```

#### 6. Start logging.

```
errCode = diagMgr->startLogCollection();
if(telux::common::ErrorCode::SUCCESS != errCode) {
   std::cout << "Failed to start logging... Error Code: "
             << static_cast<int>(errCode) << ". quitting application" <<std::endl;
   break;
}
```

#### 7. Wait for 10 seconds.

```
std::this_thread::sleep_for(std::chrono::milliseconds(10000));
```

#### 8. Stop logging.

```
errCode = diagMgr->stopLogCollection();
if(telux::common::ErrorCode::SUCCESS != errCode) {
   std::cout << "Failed to stop logging... Error Code: "
             << static_cast<int>(errCode) << ". quitting application" <<std::endl;
   break;
}
```

### 9. Cleanup.

```
if(diagMgr) {
   diagMgr = nullptr;
}
```

# 4 TelSDK logging

TelSDK provides a configurable logging facility that can be used by application to log messages. This also makes it easier for application to include its messages along with messages from TelSDK library itself.

## 4.1 Logging API

Use the LOG() API to log any message.

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

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

## 4.2 Logging configuration

TelSDK has a default configuration for logging which defines behaviour of logging. To override this configuration, a configuration file should be provided to the TelSDK. This file is searched by TelSDK logging mechanism in the following order.

- app_name.conf in /etc directory (for example, /etc/telsdk_console_app.conf)
- app_name.conf in the directory that contains the application
- tel.conf in /etc
- tel.conf in the directory that contains the application

Advantage of using different configuration files by applications is, it allows flexibility to either share the same log file or log in separate files. In QC provided builds, /etc/tel.conf has been used as an example configuration by default for all applications.

### 4.2.1 File name

Specify name of the log file if log messages are printed on file.

```
LOG_FILE_NAME=tel.log
```

### 4.2.2 File path

Specify an absolute writable directory where log file(s) will be created. The application must be part of Linux "system" group to access /data/vendor/telsdk directory.

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

### 4.2.3 Destination

The log messages can be routed to device's console, DIAG and file. This is specified when defining logging level.

```
# NONE means do not print on console

CONSOLE_LOG_LEVEL=NONE
```

### 4.2.4 Levels

Specifies threshold for emitting messages. The messages which are above this threshold appears on log destination.

#### Console and File logging

```
# NONE - No logging
# PERF - Prints messages with nanoseconds precision timestamp
# ERROR - Very minimal logging, prints perf and error messages only
# WARNING - Prints perf, error and warning messages
# INFO - Prints perf, errors, warning and information messages
# DEBUG - Full logging including debug messages

CONSOLE_LOG_LEVEL=ERROR
FILE_LOG_LEVEL=DEBUG
```

#### DIAG logging

The log messages appears in QXDM tool when DIAG is used as destination. The DIAG defines its own log levels and the mapping of them with TelSDK is given below.

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

DIAG_LOG_LEVEL=ERROR
```

### 4.2.5 File size

Specify the maximum allowed size(in bytes) of the log file. When the maximum limit is reached, it is saved as tel.log.backup.

```
MAX_LOG_FILE_SIZE=5242880
```

### 4.2.6 Adding date and time

Specify whether date and time should be prefixed to log message or not.

```
# FALSE - logs with filename and line number, this is default option
# TRUE - logs with date, time, filename and line number

LOG_PREFIX_DATE_TIME=TRUE
```

### 4.2.7 Filtering

Logs can be emitted selectively based on the technology domain by specifying TELUX_LOG_COMPONENT_FILTER.

```
# 0 - All logs are printed
# 1 - Audio logs are printed
# 2 - CV2X logs are printed
# 3 - Data logs are printed
# 4 - Location logs are printed
# 5 - Power logs are printed
# 6 - Telephony logs are printed
# 7 - Thermal logs are printed

# For logging all component
# use TELUX_LOG_COMPONENT_FILTER= 0

# For logging more than one component like cv2x and audio (comma separated)
# use TELUX_LOG_COMPONENT_FILTER= 2,1
```

# 5 Linux groups

Certain APIs of the SDK require the caller to be part of one or more Linux groups. Given below is a summary of APIs and the corresponding group IDs required by the caller. This mapping of API and group Ids might differ based on the software product family. The table below also lists the applicable software product.

**Common**

| Linux group | Usage | Applicable software products |
|---|---|---|
| system | Accessing telsdk configuration files (for ex; /etc/tel.conf) | All software products with 4.14 or older Linux kernel |
| diag | Logging using diag (qxdm logs) | All software products with 4.14 or older Linux kernel |
| system | Logging on file on file system | All software products with 4.14 or older Linux kernel |
| telux | Logging on file on file system | All software products with 5.4 or newer Linux kernel |

**Location**

| Linux group | Usage | Applicable software products |
|---|---|---|
| locclient | Needed by APIs in telux::loc::∗ namespace. | All software products |

**Sensor**

| Linux group | Usage | Applicable software products |
|---|---|---|
| sensors | Needed by APIs in telux::sensor::ISensorManager, telux::sensor::ISensorFeatureManager, telux-::sensor::ISensor/telux::sensor::ISensorClient namespace. | All software products except SPs for SA415M target |

**Power**

| Linux group | Usage | Applicable software products |
|---|---|---|
| net_raw | Needed by APIs in telux::power::ITcuActivityManager. | All software products |

**Cv2x**

| Linux group | Usage | Applicable software products |
|---|---|---|
| radio | Required for communication with modem firmware. | All software products |

**Data**

| Linux group | Usage | Applicable software products |
|---|---|---|
| inet | Required for accessing data layers. | All software products |

**Platform**

| Linux group | Usage | Applicable software products |
|---|---|---|
| system | Required for QMI access to fs-hms QMI service (/etc/sec_config) | All software products based on sa415m target |

**Telephony**

| Linux group | Usage | Applicable software products |
|---|---|---|
| net_raw or radio | Required by APIs in telux::tel::IPhoneManager, telux::tel::IPhone, telux::tel::ICallManager, telux::tel::ICall, telux::tel::INetworkSelectionManager and telux::tel::IServingSystemManager. | All software products based on mdm9650, mdm9607 and sa415m targets |
| radio | Required by APIs in telux::tel::ISubscription-Manager, telux::tel::ISubscription, telux::tel::ICardManager, telux::tel::ICard, telux::tel::ISapCardManager, telux::tel::ISimProfileManager, telux::tel::ICellBroadcastManager, telux::tel::IRemoteSimManager and telux::tel::IMultiSimManager. | All software products based on mdm9650, mdm9607 and sa415m targets |

**Security**

| Linux group | Usage | Applicable software products |
|---|---|---|
| mvm | Required for accessing crypto accelerator. | All software products |

**Audio**

Does not require application to join any particular group.

**Thermal**

Does not require application to join any particular group.

**Config**

Does not require application to join any particular group.

# 6 Request and set operating mode

This sample application demonstrates how to request and set Operating mode of device

### 1. Implement ResponseCallback interface to receive subsystem initialization status

```
std::promise<telux::common::ServiceStatus> cbProm = std::promise<telux::common::ServiceStatus>();
void initResponseCb(telux::common::ServiceStatus status) {
   if(subSystemsStatus == SERVICE_AVAILABLE) {
      std::cout << Phone Manager subsystem is ready << std::endl;
   } else if(subSystemsStatus == SERVICE_FAILED) {
      std::cout << Phone Manager subsystem initialization failed << std::endl;
   }
   cbProm.set_value(status);
}
```

### 2. Get the PhoneFactory and PhoneManager instance

```
auto &phoneFactory = PhoneFactory::getInstance();
auto phoneManager = phoneFactory.getPhoneManager(initResponseCb);
if(phoneManager == NULL) {
   std::cout << " Failed to get Phone Manager instance" << std::endl;
   return -1;
}
```

### 3. Wait for Phone Manager subsystem to be ready

```
telux::common::ServiceStatus status = cbProm.get_future().get();
if(status != SERVICE_AVAILABLE) {
   std::cout << Unable to initialize Phone Manager subsystem << std::endl;
   return -1;
}
```

### 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 IOperatingModeCallback interface and instantiate MyOperatingModeCallback

```
std::promise<bool> callbackPromise;
telux::tel::OperatingMode operatingMode;
class MyOperatingModeCallback : public telux::tel::IOperatingModeCallback {
public:
   void operatingModeResponse(OperatingMode mode, telux::common::ErrorCode error) {
     if(error == ErrorCode::SUCCESS) {
         std::cout << "requestOperatingMode response successful" << std::endl;
         std::cout << "Operating Mode: " << mode << std::endl;
         operatingMode = static_cast<telux::tel::OperatingMode>(mode);
     } else {
         std::cout << "requestOperatingMode is failed, errorCode: " << static_cast<int>(error) << std::endl
      ;
     }
   }
   callbackPromise.set_value(true);
};

auto myOperatingModeCallback = std::make_shared<MyOperatingModeCallback>();

void setOperatingModeResponse(telux::common::ErrorCode error) {
     std::cout << "Set Operating Mode is :, errorCode: " << static_cast<int>(error)
         << std::endl;
}
```

### 7. Request the operating mode of device and set the operating mode to ONLINE, if the operating mode is OFFLINE to perform any operations on the phone

```
phoneManager->requestOperatingMode(myOperatingModeCallback);
if ((callbackPromise.get_future().get()) &&
    (telux::tel::operatingMode == telux::tel::OperatingMode::OFFLINE )) {
     phoneManager->setOperatingMode(telux::tel::OperatingMode::ONLINE, &setOperatingModeResponse);
}
```

# 7 How to configure and enable WLAN

The following steps illustrate how to configure WLAN.

**1. Create the IWlanListener class.**

```
class NotificationListener: public telux::wlan::IWlanListener {
public:
void onEnableChanged(bool enable) {
    promise_.set_value(enable);
}

bool getEnableStatus() {
    return promise_.get_future().get();
}

void resetPromise() {
    promise_ = std::promise<bool>();
}
private:
std::promise<bool> promise_;
};
```

**2. Create the initialization callback object to be notified when initialization is complete and the object is ready to be used.**

```
auto initCb = [&](telux::common::ServiceStatus status) {
    initPromise.set_value(status);
};
```

**3. Get the WlanFactory and WlanDeviceManager instances.**

```
auto &wlanFactory = telux::wlan::WlanFactory::getInstance();
wlanDevMgr  = wlanFactory.getWlanDeviceManager(initCb);
```

**4. Wait for the object to complete initialization.**

```
telux::common::ServiceStatus = subSystemStatus = initPromise.get_future().get();
if (subSystemStatus == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << " *** Wlan SubSystem is Ready *** " << std::endl;
}
else {
    std::cout << " *** Unable to initialize Wlan subsystem *** " << std::endl;
}
```

**5. If WLAN is enabled, disable it.**

```
bool enableStat = false;
std::vector<telux::wlan::InterfaceStatus> ifStatus;
if(telux::common::ErrorCode::SUCCESS != wlanDevMgr->getStatus(enableStat, ifStatus)) {
    std::cout << "Failed to retrieve Wlan status" <<std::endl;
}
if(enableStat) {
    //Disable Wlan before changing configuration...
    wlanDevMgr->registerListener(listener);
```

```
    //Ensure callback returns SUCCESS and indication for Wlan disablement is received
    if((telux::common::ErrorCode::SUCCESS != wlanDevMgr->enable(false)) ||
        (true == listener->getEnableStatus())) {
        std::cout << "Failed to disable Wlan " <<std::endl;
        break;
    }
}
```

### 6. Set the desired WLAN configurations.

```
if(telux::common::ErrorCode::SUCCESS != wlanDevMgr->setMode(numAp, numSta)) {
    std::cout << "Failed to set Wlan configuration" <<std::endl;
}
```

### 7. Enable WLAN for new configurations to take effect

```
listener->resetPromise();
//Ensure callback returns SUCCESS and indication for Wlan enablement is received
if((telux::common::ErrorCode::SUCCESS != wlanDevMgr->enable(true)) ||
   (false == listener->getEnableStatus())) {
    std::cout << "Failed to enable Wlan " <<std::endl;
}
```

# 8 How to configure and use WLAN STA operations

The following steps illustrate how to configure and use WLAN STA operation.

**1. Create the IWlanListener, IStaListener class.**

```
class NotificationListener: public telux::wlan::IWlanListener,
                            public telux::wlan::IStaListener,
                            public std::enable_shared_from_this<NotificationListener> {
public:
void onEnableChanged(bool enable) {
    promise_.set_value(enable);
}

bool getEnableStatus() {
    return promise_.get_future().get();
}

bool isScanCompleted() {
    return scanCompletePromise_.get_future().get();
}

void resetPromise() {
    promise_ = std::promise<bool>();
}

void resetScanCompletePromise() {
    scanCompletePromise_ = std::promise<bool>();
}

void onScanResultUpdated(const telux::wlan::StaScanResult &staScanResult) {
    // Use staScanResult constituents as needed
    if(staScanResult.isScanComplete) {
        scanCompletePromise_.set_value(true);
    }
}

void onStationStatusChanged(std::vector<telux::wlan::StaStatus> status) {
     std::cout << "length of Active Stations:" << status.size() << std::endl;
}

private:
std::promise<bool> promise_;
std::promise<bool> scanCompletePromise_;
};
```

**2. Create the initialization callback object to be notified when initialization is complete and the object is ready to be used.**

```
std::promise<telux::common::ServiceStatus> initPromise{};

auto initCb = [&](telux::common::ServiceStatus status) {
    initPromise.set_value(status);
};
```

### 3. Get the WlanFactory, WlanDeviceManager and StaInterfaceManager instances.

```
auto &wlanFactory = telux::wlan::WlanFactory::getInstance();
wlanDevMgr_  = wlanFactory.getWlanDeviceManager(initCb);
wlanStaMgr_  = wlanFactory.getStaInterfaceManager();
```

### 4. Wait for the object to complete initialization.

```
telux::common::ServiceStatus subSystemStatus = initPromise.get_future().get();
if (subSystemStatus == telux::common::ServiceStatus::SERVICE_AVAILABLE) {
    std::cout << " *** Wlan SubSystem is Ready *** " << std::endl;
}
else {
    std::cerr << " *** Unable to initialize Wlan subsystem *** " << std::endl;
}
```

### 5. If WLAN is enabled, disable it (optional).

```
bool enableStat = false;
std::vector<telux::wlan::InterfaceStatus> ifStatus;
if(telux::common::ErrorCode::SUCCESS != wlanDevMgr_->getStatus(enableStat, ifStatus)) {
    std::cerr << "Failed to retrieve Wlan status" <<std::endl;
}
if(enableStat) {
    //Ensure callback returns SUCCESS and indication for Wlan disablement is received
    if((telux::common::ErrorCode::SUCCESS != wlanDevMgr_->enable(false)) ||
        (true == listener->getEnableStatus())) {
        std::cerr << "Failed to disable Wlan " <<std::endl;
    }
}
```

### 6. Register listeners to receive notifications.

```
telux::common::ErrorCode ec;
ec = wlanDevMgr_->registerListener(listener);
if (ec != telux::common::ErrorCode::SUCCESS) {
    std::cerr << "Can't register to IWlanListener" << std::endl;
}

ec = wlanStaMgr_->registerListener(listener);
if (ec != telux::common::ErrorCode::SUCCESS) {
    std::cerr << "Can't register to IStaListener" << std::endl;
}
```

### 7. Set the desired WLAN configurations.

```
//Ensure that atleast one station is configured, numSta > 1
 if(telux::common::ErrorCode::SUCCESS != wlanDevMgr_->setMode(numAp, numSta)) {
     std::cerr << "Failed to set Wlan configuration" <<std::endl;
 }
```

### 8. Enable WLAN for new configurations to take effect.

```
listener->resetPromise();
//Ensure callback returns SUCCESS and indication for Wlan enablement is received
if((telux::common::ErrorCode::SUCCESS != wlanDevMgr_->enable(true)) ||
   (false == listener->getEnableStatus())) {
    std::cerr << "Failed to enable Wlan " << std::endl;
}
```

### 9. Initiate STA scan process.

```
listener->resetScanCompletePromise();

if(telux::common::ErrorCode::SUCCESS != wlanStaMgr_->startScan(telux::wlan::Id::PRIMARY)) {
    std::cerr << "Failed to enable Wlan STA Scan " << std::endl;
}
```

### 10. Add the network configuration once the scan is complete.

```
telux::common::ErrorCode ec;

if(listener->isScanCompleted()) {
    std::cout << "Adding Network Config" << std::endl;
    ec = wlanStaMgr_->addNetworkConfig(telux::wlan::Id::PRIMARY, staNetworkConfigEntry);
    if (ec != telux::common::ErrorCode::SUCCESS) {
        std::cerr << "Can't add NetworkConfig entry, err " << static_cast<int>(ec)
                  << std::endl;
    }
    // expect scan connection status details via onStationStatusChanged() notification
}
```

# LEGAL INFORMATION

**Your access to and use of this material, along with any documents, software, specifications, reference board files, drawings, diagnostics and other information contained herein (collectively this "Material"), is subject to your (including the corporation or other legal entity you represent, collectively "You" or "Your") acceptance of the terms and conditions ("Terms of Use") set forth below. If You do not agree to these Terms of Use, you may not use this Material and shall immediately destroy any copy thereof.**

**1) Legal Notice.**

This Material is being made available to You solely for Your internal use with those products and service offerings of Qualcomm Technologies, Inc. ("**Qualcomm Technologies**"), its affiliates and/or licensors described in this Material, and shall not be used for any other purposes. If this Material is marked as "**Qualcomm Internal Use Only**", no license is granted to You herein, and You must immediately (a) destroy or return this Material to Qualcomm Technologies, and (b) report Your receipt of this Material to qualcomm.support@qti.qualcomm.com. This Material may not be altered, edited, or modified in any way without Qualcomm Technologies' prior written approval, nor may it be used for any machine learning or artificial intelligence development purpose which results, whether directly or indirectly, in the creation or development of an automated device, program, tool, algorithm, process, methodology, product and/or other output. Unauthorized use or disclosure of this Material or the information contained herein is strictly prohibited, and You agree to indemnify Qualcomm Technologies, its affiliates and licensors for any damages or losses suffered by Qualcomm Technologies, its affiliates and/or licensors for any such unauthorized uses or disclosures of this Material, in whole or part.

Qualcomm Technologies, its affiliates and/or licensors retain all rights and ownership in and to this Material. No license to any trademark, patent, copyright, mask work protection right or any other intellectual property right is either granted or implied by this Material or any information disclosed herein, including, but not limited to, any license to make, use, import or sell any product, service or technology offering embodying any of the information in this Material.

THIS MATERIAL IS BEING PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE. TO THE MAXIMUM EXTENT PERMITTED BY LAW, QUALCOMM TECHNOLOGIES, ITS AFFILIATES AND/OR LICENSORS SPECIFICALLY DISCLAIM ALL WARRANTIES OF TITLE, MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, COMPLETENESS OR ACCURACY, AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MOREOVER, NEITHER QUALCOMM TECHNOLOGIES, NOR ANY OF ITS AFFILIATES AND/OR LICENSORS, SHALL BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY EXPENSES, LOSSES, USE, OR ACTIONS HOWSOEVER INCURRED OR UNDERTAKEN BY YOU IN RELIANCE ON THIS MATERIAL.

Certain product kits, tools and other items referenced in this Material may require You to accept additional terms and conditions before accessing or using those items.

Technical data specified in this Material may be subject to U.S. and other applicable export control laws. Transmission contrary to U.S. and any other applicable law is strictly prohibited.

Nothing in this Material is an offer to sell any of the components or devices referenced herein.

This Material is subject to change without further notification.

In the event of a conflict between these Terms of Use and the Website Terms of Use on www.qualcomm.com, the *Qualcomm Privacy Policy* referenced on www.qualcomm.com, or other legal statements or notices found on prior pages of the Material, these Terms of Use will control. In the event of a conflict between these Terms of Use and any other agreement (written or click-through, including, without limitation any non-disclosure agreement) executed by You and Qualcomm Technologies or a Qualcomm Technologies affiliate and/or licensor with respect to Your access to and use of this Material, the other agreement will control.

These Terms of Use shall be governed by and construed and enforced in accordance with the laws of the State of California, excluding the U.N. Convention on International Sale of Goods, without regard to conflict of laws principles. Any dispute, claim or controversy arising out of or relating to these Terms of Use, or the breach or validity hereof, shall be adjudicated only by a court of competent jurisdiction in the county of San Diego, State of California, and You hereby consent to the personal jurisdiction of such courts for that purpose.

**2) Trademark and Product Attribution Statements.**

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the U.S. and/or elsewhere. The Bluetooth® word mark is a registered trademark owned by Bluetooth SIG, Inc. Other product and brand names referenced in this Material may be trademarks or registered trademarks of their respective owners.

Snapdragon and Qualcomm branded products referenced in this Material are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.

**THE DOCUMENTATION ACCOMPANYING THE MATERIALS AND/OR RELEVANT PRODUCTS AND SERVICE OFFERINGS MAY INCLUDE IMPORTANT USE LIMITATIONS. ANY DEVIATIONS FROM APPLICABLE USE LIMITATIONS MAY ADVERSELY IMPACT PERFORMANCE, DURABILITY, QUALITY OR SAFETY. YOU ASSUME ALL RISKS AND LIABILITIES ASSOCIATED WITH ANY DEVIATIONS.**