Qualcomm

Qualcomm Technologies, Inc.

Telematics SDK

User Guide
v1.22.0

80-PF458-1 Rev. H

September 17, 2019

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

Revision History

Revision	Date	Description
A	Sep 2017	Initial Release
B	Dec 2017	Added subscription feature
C	Jun 2018	Added data services apps
D	Oct 2018	Update of Yocto Platform SDK instructions, Addition of Network Selection and Service System Manager
E	Nov 2018	Addition of C-V2X samples
F	Jan 2019	Addition of Audio Manager apps reference code
G	Mar 2019	Addition of Thermal Manager apps
H	May 2019	Addition of TCU Activity Management feature
I	Jun 2019	Addition of Audio play, capture apps reference code
J	Jul 2019	Updated the Location APIs apps
G	Jul 2019	Addition of Thermal shutdown manager APIs and call flows

Contents

1 Introduction

1.1 Purpose

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

1.2 Scope

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

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

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

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

2 Building Yocto Platform SDK

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

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

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

You need to have the following in order to proceed:

- Linux Ubuntu 14.04
- Install required packages

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

2.1 Get Started

- Sync the CAF build
  repo init and sync the sources
  For LE.UM.1.3.2 target

```
$ repo init -q -u git://codeaurora.org/quic/le/le/manifest.git -b release -m
       caf_AU_LINUX_EMBEDDED_LE.UM.1.3.2_TARGET_ALL.01.105.149.xml
$ repo sync -j 32
//
// Note: AU_LINUX_EMBEDDED_LE.UM.1.3.2_TARGET_ALL.01.105.149 should be replaced the AU tag
// corresponding to the desired device build.
```

  For LE.UM.1.3.r5 target

```
$ repo init -q -u git://codeaurora.org/quic/le/le/manifest.git -b release -m
       caf_AU_LINUX_EMBEDDED_LE.UM.1.3.R5_TARGET_ALL.01.66.138.xml
$ repo sync -j 32
```

  The tags are listed here - https://source.codeaurora.-org/quic/le/le/manifest/refs/?h=IMM.LE.1.0

- Update Telux and Telephony libs
  Add the following lines at the end of poky/build/conf/local.conf:
  For LE.UM.1.3.2 target

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

  For LE.UM.1.3.r5 target

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

- Setup the build environment

Source the bitbake environment

```
$ cd poky/
$ source build/conf/set_bb_env.sh
```

- Build and install the Yocto Platform SDK
  Run the following command to create a platform sdk

```
$ bitbake core-image-minimal -c do_populate_sdk
```

- Location of sdk installer script
  After successful compilation, you will have have sdk installer on this location

```
poky/build/tmp-glibc/deploy/sdk/oecore-x86_64-cortexa8hf-vfp-neon-toolchain-nodistro.0.sh
```

3 Steps to writing an App

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

You need to have the following in order to proceed:

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

3.1 Source the environment

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

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

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

3.2 To get started with a sample app

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

#include <iostream>
#include <memory>

#include <telux/tel/PhoneFactory.hpp>

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

int main() {

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

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

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

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

3.3 Compile the program

- Now compile your source code

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

3.4 Install apps on the device

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

```
user@machine:/sample_app$ adb push sample_app /data/
user@machine:/sample_app$ adb shell
$ cd data
$ chmod +x sample_app
$ ./sample_app
```

4 Sample Apps

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

4.1 Source the environment

- Source the build environment:
```
$ cd my_sdk_install; source environment-setup-cortexa8hf-vfp-neon-oe-linux-gnueabi
```

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

4.2 Compile sample apps

- Goto telux/samples directory and create a build folder
```
   $ cd telux/samples
$ mkdir -p build; cd build
```
- Compile all the samples:
```
$ cmake -DBUILD_ALL_SAMPLES:BOOL=ON -DCMAKE_INSTALL_PREFIX=./install ..
$ make clean && make install
```

Compile telsdk_console_app:

- Goto respective sample program for compiling individual programs
```
$ cd samples/telsdk_console_app
$ mkdir -p build; cd build
$ cmake -DCMAKE_INSTALL_PREFIX=./install ..
$ make clean && make install
```

4.3 Install apps on the device

- Push your binary to the /data location on the device
```
# this install all the sample apps
user@machine:/build/install/bin$ adb push . /data/
user@machine:/build/install/bin$ adb shell
# cd data
# ./telsdk_console_app

# to install telsdk_console app
user@machine:telsdk_console_app/build/install/bin$ adb push . /data/
```

5 Configuring Logs from the SDK

Please follow below steps to configure Logger settings.

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

By default, both console logging and file logging are set to "NONE" log level, *tel.conf* will be placed under /etc location

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

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

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

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

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

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

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

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

1. Console and file level logging

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

```
# NONE - No logging.
# ERROR - Very minimal logging.Prints error messages only.
# WARNING - Prints both error and warning messages.
# INFO - Prints errors, warning and information messages.
# DEBUG - Full logging including debug messages.It is intended for debugging purposes only.

CONSOLE_LOG_LEVEL=INFO
FILE_LOG_LEVEL=DEBUG
```

2. Set Max file size

MAX_LOG_FILE_SIZE specifies the maximum allowed size(in bytes) of the log file

- When max size is reached, logger backs up the log file once, for example: tel.log will be renamed to tel.log.backup and a new log file will be created.
- Default MAX_LOG_FILE_SIZE is 5 Mega Bytes

```
MAX_LOG_FILE_SIZE=5242880
```

3. Prefix date and time for the log message

Used to prefix date and time on every log Message

```
# 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. Set log file path

Specifies the path of the log file. In an external application processor, the path needs to be in a writable partition.

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

5. Set log file name

Specifies the name of the log file to be used

```
LOG_FILE_NAME=tel.log
```

6 Sample Applications Configuration

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

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

DIAL_NUMBER = +1234512345

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

RECEIVER_NUMBER = +1234512345

MESSAGE = Text message

Below are the steps to run the sample application.

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

7 Making a Voice Call

Please follow below steps to make a voice call

1. Get the PhoneFactory and PhoneManager instances

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

2. Check if telephony subsystem is ready

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

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

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

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

3. Instantiate Phone and call manager

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

4. Initialize phoneId with default value

```
int phoneId = DEFAULT_PHONE_ID;
```

5. Instantiate dial call instance - this is optional

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

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

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

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

6. Send a dial request

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

8 Making eCall (Emergency E112)

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

1. Get the PhoneFactory and PhoneManager instances.

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

2. Check if telephony subsystem is ready

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

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

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

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

3. Instantiate Phone and call manager

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

5. Initialize phoneId with default value

```
int phoneId = DEFAULT_PHONE_ID;
```

6. Instantiate dial callback instance - this is optional

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

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

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

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

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

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

8. Send a eCall request

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

9 Request Voice Service State of the device

Please follow below steps to get voice service state notifications.

1. Get the PhoneFactory and PhoneManager instances

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

2. Check if telephony subsystem is ready

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

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

If subsystem is not ready, wait unconditionally.

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

3. Instantiate Phone

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

4. Check for radio state

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

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

5. Implement IVoiceServiceStateCallback interface

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

6. Instantiate MyVoiceServiceStateCallback

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

7. Send voice service state request.

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

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

10 Set radio power of the device

Please follow below steps to Radio Power state notifications.

1. Get the PhoneFactory and PhoneManager instances

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

2. Check if telephony subsystem is ready

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

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

If subsystem is not ready, wait unconditionally.

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

3. Instantiate Phone

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

4. Implement IPhoneListener interface to receive service state change notifications

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

4.1 Instantiate MyPhoneListener

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

5. Register for phone info updates

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

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

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

7. Instantiate MyPhoneCommandResponseCallback

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

8. Set the Radio power ON/OFF.

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

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

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

11 Using Subscription Manager APIs

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

1. Get the PhoneFactory and SubscriptionManager instances

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

2. Wait for the Subscription subsystem to ready

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

3. Get the Subscription information

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

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

12 Listening to Incoming SMS

Please follow below steps to listen for incoming SMS

1. Implement ISmsListener interface to receive incoming SMS

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

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

2. Get the PhoneFactory and PhoneManager instances

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

3. Check if telephony subsystem is ready

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

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

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

5. Instantiate global ISmsListener

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

6. Get default SMS Manager instance

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

7. Register for incoming SMS

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

8. Wait for incoming SMS

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

13 Sending SMS

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

1. Get the PhoneFactory and PhoneManager instances.

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

2. Check if telephony subsystem is ready

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

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

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

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

3. Instantiate SMS sent and delivery callback

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

3.1 Implement ICommandResponseCallback interface to know SMS sent and Delivery status

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

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

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

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

4. Get default SMS Manager instance

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

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

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

6. Receive responses for sendSms request

14 Using Card Service APIs

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

1. Get the PhoneFactory and CardManager instances.

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

2. Wait for the telephony subsystem initialization.

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

3. Get SlotCount, SlotIds and Card instance

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

4. Get supported applications from the card

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

5. Instantiate optional IOpenLogicalChannelCallback, ICommandResponseCallback and ITransmitApduResponseCallback

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

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

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

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

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

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

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

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

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

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

6. Open Logical Channel and wait for request to complete

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

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

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

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

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

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

9. Transmit Apdu on Basic Channel and wait for completion

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

15 Using SAP APIs

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

1. Get the PhoneFactory and PhoneManager instances.

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

2. Wait for the telephony subsystem initialization.

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

3. Get default Sap Card Manager instance

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

4. Instantiate ICommandResponseCallback, IAtrResponseCallback and ISapCardCommandCallback

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

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

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

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

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

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

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

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

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

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

5. Open Sap connection and wait for request to complete

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

6. request sap ATR and wait for complete

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

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

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

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

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

16 Using Location Service APIs

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

1. Implement a command response function

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

2. Implement ILocationListener interface

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

3. Get the LocationFactory instance

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

4. Get LocationManager instance

```
auto locationManager_ = locationFactory.getLocationManager();
```

5. Wait for the location subsystem initialization

```
bool subSystemsStatus = locationManager_->isSubsystemReady();
if (!subSystemsStatus) {
    std::cout << "Location subsystem is not ready, Please wait!!!... "<< std::endl;
    std::future<bool> f = locationManager_->onSubsystemReady();
    subSystemsStatus = f.get();
}
```

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

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

7. Instantiate MyLocationListener

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

8. Register for Location, SV and Jammer info updates

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

9. Start Location reports with Detailed information

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

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

11. Wait for Location fix, SV and Jammer info

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

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

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

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

17 How to get data profile list

Please follow below steps to request list of available modem profiles

1. Get the DataFactory and DataProfileManager instances

```
auto &dataFactory = DataFactory::getInstance();
auto dataProfileMgr = dataFactory.getDataProfileManager();
```

2. Instantiate requestProfileList callback

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

2.1 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;
  }
};
```

3. Send a requestProfileList along with required callback function

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

4. Receive DataProfileListCallback responses for requestProfileList request

18 Cellular Data Call - Start/Stop

Please follow below steps to start or stop cellular data call

1. Get the DataFactory and DataConnectionManager instances

```
auto &dataFactory = DataFactory::getInstance();
auto dataConnectionManager = dataFactory.getDataConnectionManager();
```

2. Check if data subsystem is ready

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

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

Data subsystems is to make sure that device is ready for services like bring-up or tear-down cellular data call. if subsystems were not ready, wait for unconditionally.

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

3. 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;
   }
}
```

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

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

5. Response callback will be called for the startDataCall response

6. 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;
}
```

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

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

8. Response callback will be called for the stopDataCall response

19 Using Request Network Selection Mode API

Please follow below steps to request network selection mode

1. Get phone factory and network selection manager instances

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

2. Wait for the network selection subsystem initialization

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

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

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

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

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

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

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

5. Send requestNetworkSelectionMode along with required function object

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

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

20 Using Request Service Domain Preference API

Please follow below steps to request service domain preference

1. Get phone factory and serving system manager instances

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

2. Wait for the serving subsystem initialization

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

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

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

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

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

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

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

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

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

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

21 C-V2X Get Status Sample App

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

1. Create a RequestCv2xStatusCallback function

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

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

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

2. Get a handle to the ICv2xRadioManager object

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

3. Request the C-V2X status

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

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

    return EXIT_SUCCESS;

} // main
```

22 C-V2X RX Sample App

This Document walks through the cv2x_rx_app sample application.

1. Create Callback functions for ICv2xRadio and ICv2xRadioManager methods

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

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

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

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

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

2. Get a handle to the ICv2xRadioManager object

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

3. Request the C-V2X status

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

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

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

```
}
```

4. Get handle to C-V2X Radio

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

5. Wait for C-V2X Radio to be ready

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

6. Create RX Subscription and receive data using RX socket

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

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

7. Close RX Subcription

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

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

    return EXIT_SUCCESS;
} // main
```

23 C-V2X TX Sample App

This Document walks through the cv2x_tx_app sample application.

1. Create Callback functions for ICv2xRadio and ICv2xRadioManager methods

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

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

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

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

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

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

2. Get a handle to the ICv2xRadioManager object

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

3. Request the C-V2X status

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

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

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

4. Get handle to C-V2X Radio

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

5. Wait for C-V2X Radio to be ready

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

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

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

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

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

7. Close TX SPS flow

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

    return EXIT_SUCCESS;
} // main
```

24 Audio Manager API Sample Reference

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

1. Get the AudioFactory and AudioManager instances

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

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

// Globals
static std::shared_ptr<IAudioManager> audioManager;
static std::shared_ptr<IAudioVoiceStream> audioVoiceStream;
static unsigned int timeoutSec = 5;
Status status;

auto &audioFactory = audioFactory::getInstance();
audioManager = audioFactory.getAudioManager();
```

2. Check if Audio subsystem is ready

```
bool subSystemsStatus = audioManager->isSubsystemReady();
if (subSystemsStatus) {
    std::cout << "Audio Subsystem is ready." << std::endl;
} else {
    std::cout << "Audio Subsystem is NOT ready." << std::endl;
}
```

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

Make sure that Audio subsystem is ready for services like voice call. if subsystems were not ready, unconditionally wait or Timeout based wait.

```
std::future<bool> f = audioManager->onSubsystemReady();
#if  //Timeout based wait
     if (f.wait_for(std::chrono::seconds(timeoutSec)) == std::future_status::timeout) {
         std::cout << "operation timed out." << std::endl;
     } else {
         std::cout << "Audio Subsystem is ready." << std::endl;
     }
#else //Unconditional wait
     bool subSystemsStatus = f.get();
     if (subSystemsStatus) {
       std::cout << "Audio Subsystem is ready." << std::endl;
     } else {
       std::cout << "Audio Subsystem is NOT ready." << std::endl;
     }
#endif
```

3. Query Supported Devices and Stream Types of Audio subsystem

Below methods provides details on supported Device Types and Stream Types

3.1 Query Supported Devices Types of Audio subsystem

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

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

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

3.2 Query Supported Stream Types of Audio subsystem

```
//Callback to get supported stream type details.
void getStreamTypesCallback(std::vector<StreamType> streams, ErrorCode error)
{
    if (error != ErrorCode::SUCCESS) {
        std::cout << "getStreamTypes() returned with error " << static_cast<unsigned int>(error)
            << std::endl;
        return;
    }

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

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

4. Create an Audio Stream (Voice Call Session)

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

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

//Create an Audio Stream (Voice Call Session)
StreamConfig config;
config.type = StreamType::VOICE_CALL;
config.modemSubId = 1;
config.sampleRate = 16000;
config.format = AudioFormat::PCM_16BIT_SIGNED;
config.channelTypeMask = ChannelType::LEFT;
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
status = audioManager->createStream(config, createStreamCallback);
```

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

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

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

25 Audio Manager API Sample Reference for audio capture session

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

1. Get the AudioFactory and AudioManager instances

```
auto &audioFactory = audioFactory::getInstance();
auto audioManager = audioFactory.getAudioManager();
```

2. Check if Audio subsystem is ready

```
bool subSystemsStatus = audioManager->isSubsystemReady();
if (subSystemsStatus) {
    std::cout << "Audio Subsystem is ready." << std::endl;
} else {
    std::cout << "Audio Subsystem is NOT ready." << std::endl;
}
```

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

Make sure that Audio subsystem is ready for services like audio capture. If subsystem is not ready, wait unconditionally (or) until a timeout.

```
std::future<bool> f = audioManager->onSubsystemReady();
#if  //Timeout based wait
    if (f.wait_for(std::chrono::seconds(timeoutSec)) == std::future_status::timeout) {
        std::cout << "operation timed out." << std::endl;
    } else {
        subSystemsStatus = f.get();
        if (subSystemsStatus) {
            std::cout << "Audio Subsystem is ready." << std::endl;
        }
    }
#else //Unconditional wait
    subSystemsStatus = f.get();
    if (subSystemsStatus) {
        std::cout << "Audio Subsystem is ready." << std::endl;
    } else {
        std::cout << "Audio Subsystem is NOT ready." << std::endl;
    }
#endif
```

3. Create an Audio Stream (Audio Capture Session)

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

```
    std::cout << "createStream() succeeded." << std::endl;
    audioCaptureStream = std::dynamic_pointer_cast<IAudioCaptureStream>(stream);
}

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

4. Allocate Stream Buffers for Capture Operation

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

5. Start read operation for the capture to Start

```
//Callback which provides response to read operation
void readCallback(std::shared_ptr<IStreamBuffer> buffer, ErrorCode error)
{
    uint32_t bytesWrittenToFile = 0;
    if (error != ErrorCode::SUCCESS) {
        std::cout << "read() returned with error " << static_cast<int>(error) << std::endl;
    } else {
        uint32_t size = buffer->getDataSize();
        std::cout << "Successfully read " << size << " bytes" << std::endl;
    }
    buffer->reset();
    return;
}
//Read from Capture
//First read starts Capture Session.
auto status = audioCaptureStream->read(streamBuffer, bytesToRead, readCallback);
if(status != telux::common::Status::SUCCESS) {
    std::cout << "read() failed with error" << static_cast<int>(status) << std::endl;
} else {
    std::cout << "Request to read stream sent" << std::endl;
}
```

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

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

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

26 Audio Manager API Sample Reference for audio playback session

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

1. Get the AudioFactory and AudioManager instances

```
auto &audioFactory = audioFactory::getInstance();
auto audioManager = audioFactory.getAudioManager();
```

2. Check if Audio subsystem is ready

```
bool subSystemsStatus = audioManager->isSubsystemReady();
if (subSystemsStatus) {
    std::cout << "Audio Subsystem is ready." << std::endl;
} else {
    std::cout << "Audio Subsystem is NOT ready." << std::endl;
}
```

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

Make sure that Audio subsystem is ready for services like audio play. If subsystem is not ready, wait unconditionally (or) until a timeout.

```
std::future<bool> f = audioManager->onSubsystemReady();
#if  //Timeout based wait
    if (f.wait_for(std::chrono::seconds(timeoutSec)) == std::future_status::timeout) {
        std::cout << "operation timed out." << std::endl;
    } else {
        subSystemsStatus = f.get();
        if (subSystemsStatus) {
            std::cout << "Audio Subsystem is ready." << std::endl;
        }
    }
#else //Unconditional wait
    subSystemsStatus = f.get();
    if (subSystemsStatus) {
        std::cout << "Audio Subsystem is ready." << std::endl;
    } else {
        std::cout << "Audio Subsystem is NOT ready." << std::endl;
    }
#endif
```

3. Create an Audio Stream (Audio Playback Session)

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

```
     std::cout << "createStream() succeeded." << std::endl;
     audioPlayStream = std::dynamic_pointer_cast<IAudioPlayStream>(stream);
 }
 //Create an Audio Stream (Audio Playback Session)
 StreamConfig config;
 config.type = StreamType::PLAY;
 config.sampleRate = 48000;
 config.format = AudioFormat::PCM_16BIT_SIGNED;
 config.channelTypeMask = ChannelType::LEFT;
 config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
 status = audioManager->createStream(config, createStreamCallback);
```

4. Allocate Stream buffers for Playback operation

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

5. Start write operation for playback to start

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

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

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

```
    std::cout << "deleteStream failed with error" << static_cast<int>(status) << std::endl;
}
```

27 Audio Manager API Sample Reference for voice session device switch

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

1. Get the AudioFactory and AudioManager instances

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

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

// Globals
static std::shared_ptr<IAudioManager> audioManager;
static std::shared_ptr<IAudioVoiceStream> audioVoiceStream;
static unsigned int timeoutSec = 5;
Status status;

auto &audioFactory = audioFactory::getInstance();
audioManager = audioFactory.getAudioManager();
```

2. Check if Audio subsystem is ready

```
bool subSystemsStatus = audioManager->isSubsystemReady();
if (subSystemsStatus) {
    std::cout << "Audio Subsystem is ready." << std::endl;
} else {
    std::cout << "Audio Subsystem is NOT ready." << std::endl;
}
```

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

Make sure that Audio subsystem is ready for services like voice call. if subsystems were not ready, unconditionally wait or Timeout based wait.

```
std::future<bool> f = audioManager->onSubsystemReady();
#if  //Timeout based wait
     if (f.wait_for(std::chrono::seconds(timeoutSec)) == std::future_status::timeout) {
         std::cout << "operation timed out." << std::endl;
     } else {
         std::cout << "Audio Subsystem is ready." << std::endl;
     }
#else //Unconditional wait
     bool subSystemsStatus = f.get();
     if (subSystemsStatus) {
       std::cout << "Audio Subsystem is ready." << std::endl;
     } else {
       std::cout << "Audio Subsystem is NOT ready." << std::endl;
     }
#endif
```

3. Create an Audio Stream (Voice Call Session)

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

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

//Create an Audio Stream (Voice Call Session)
StreamConfig config;
config.type = StreamType::VOICE_CALL;
config.modemSubId = 1;
config.sampleRate = 16000;
config.format = AudioFormat::PCM_16BIT_SIGNED;
config.channelTypeMask = ChannelType::LEFT;
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
status = audioManager->createStream(config, createStreamCallback);
```

4. Start Created Audio Stream (Voice Call Session)

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

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

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

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

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

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

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

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

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

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

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

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

7. Stop Created Audio Stream (Voice Call Session)

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

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

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

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

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

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

28 Audio Manager API Sample Reference for voice session start and stop

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

1. Get the AudioFactory and AudioManager instances

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

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

// Globals
static std::shared_ptr<IAudioManager> audioManager;
static std::shared_ptr<IAudioVoiceStream> audioVoiceStream;
static unsigned int timeoutSec = 5;
Status status;

auto &audioFactory = audioFactory::getInstance();
audioManager = audioFactory.getAudioManager();
```

2. Check if Audio subsystem is ready

```
bool subSystemsStatus = audioManager->isSubsystemReady();
if (subSystemsStatus) {
    std::cout << "Audio Subsystem is ready." << std::endl;
} else {
    std::cout << "Audio Subsystem is NOT ready." << std::endl;
}
```

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

Make sure that Audio subsystem is ready for services like voice call. if subsystems were not ready, unconditionally wait or Timeout based wait.

```
std::future<bool> f = audioManager->onSubsystemReady();
#if  //Timeout based wait
     if (f.wait_for(std::chrono::seconds(timeoutSec)) == std::future_status::timeout) {
         std::cout << "operation timed out." << std::endl;
     } else {
         std::cout << "Audio Subsystem is ready." << std::endl;
     }
#else //Unconditional wait
     bool subSystemsStatus = f.get();
     if (subSystemsStatus) {
       std::cout << "Audio Subsystem is ready." << std::endl;
     } else {
       std::cout << "Audio Subsystem is NOT ready." << std::endl;
     }
#endif
```

3. Create an Audio Stream (Voice Call Session)

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

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

//Create an Audio Stream (Voice Call Session)
StreamConfig config;
config.type = StreamType::VOICE_CALL;
config.modemSubId = 1;
config.sampleRate = 16000;
config.format = AudioFormat::PCM_16BIT_SIGNED;
config.channelTypeMask = ChannelType::LEFT;
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
status = audioManager->createStream(config, createStreamCallback);
```

4. Start Created Audio Stream (Voice Call Session)

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

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

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

5. Stop Created Audio Stream (Voice Call Session)

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

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

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

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

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

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

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

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

1. Get the AudioFactory and AudioManager instances

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

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

// Globals
static std::shared_ptr<IAudioManager> audioManager;
static std::shared_ptr<IAudioVoiceStream> audioVoiceStream;
static unsigned int timeoutSec = 5;
Status status;

auto &audioFactory = audioFactory::getInstance();
audioManager = audioFactory.getAudioManager();
```

2. Check if Audio subsystem is ready

```
bool subSystemsStatus = audioManager->isSubsystemReady();
if (subSystemsStatus) {
    std::cout << "Audio Subsystem is ready." << std::endl;
} else {
    std::cout << "Audio Subsystem is NOT ready." << std::endl;
}
```

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

Make sure that Audio subsystem is ready for services like voice call. if subsystems were not ready, unconditionally wait or Timeout based wait.

```
std::future<bool> f = audioManager->onSubsystemReady();
#if  //Timeout based wait
     if (f.wait_for(std::chrono::seconds(timeoutSec)) == std::future_status::timeout) {
         std::cout << "operation timed out." << std::endl;
     } else {
         std::cout << "Audio Subsystem is ready." << std::endl;
     }
#else //Unconditional wait
     bool subSystemsStatus = f.get();
     if (subSystemsStatus) {
       std::cout << "Audio Subsystem is ready." << std::endl;
     } else {
       std::cout << "Audio Subsystem is NOT ready." << std::endl;
     }
#endif
```

3. Create an Audio Stream (Voice Call Session)

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

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

//Create an Audio Stream (Voice Call Session)
StreamConfig config;
config.type = StreamType::VOICE_CALL;
config.modemSubId = 1;
config.sampleRate = 16000;
config.format = AudioFormat::PCM_16BIT_SIGNED;
config.channelTypeMask = ChannelType::LEFT;
config.deviceTypes.emplace_back(DeviceType::DEVICE_TYPE_SPEAKER);
status = audioManager->createStream(config, createStreamCallback);
```

4. Start Created Audio Stream (Voice Call Session)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

}

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

9. Stop Created Audio Stream (Voice Call Session)

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

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

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

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

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

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

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

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

1. Implement IVoiceListener interface

```
class MyVoiceListener : public IVoiceListener {
public:
    void onDtmfToneDetection(DtmfTone dtmfTone) override;
};
```

2. Get the Audio Factory and Audio Manager instances

```
auto &audioFactory = AudioFactory::getInstance();
auto audioManager = audioFactory.getAudioManager();
```

3. Wait for the Audio subsystem to be initialized and ready

```
bool isReady = audioManager->isSubsystemReady();
if(!isReady) {
    std::cout << "Audio subsystem is not ready, waiting for it to be ready " << std::endl;
    std::future<bool> f = audioManager->onSubsystemReady();
    isReady = f.get();
}
```

4. 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;
}
```

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

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

6. Start the Voice call session

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

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

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

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

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

9. Stop the Voice call session

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

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

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

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

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

1. Get the Audio Factory and Audio Manager instances

```
auto &audioFactory = AudioFactory::getInstance();
auto audioManager = audioFactory.getAudioManager();
```

2. Wait for the Audio subsystem to be initialized and ready

```
bool isReady = audioManager->isSubsystemReady();
if(!isReady) {
    std::cout << "Audio subsystem is not ready, waiting for it to be ready " << std::endl;
    std::future<bool> f = audioManager->onSubsystemReady();
    isReady = f.get();
}
```

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

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

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

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

5. Start the Voice call session

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

```
    std::cout << "startAudio() succeeded." << std::endl;
}
// Start the Voice call session
status = audioVoiceStream->startAudio(startAudioCallback);
```

6. 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);
```

7. Stop the Voice call session

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

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

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

32 Using Thermal Shutdown Manager APIs to get Thermal autoshutdown mode updates

The below steps need to be followed by applications to listen to thermal auto-shutdown mode updates.

1. Implement IThermalShutdownListener interface

```
class MyThermalShutdownModeListener : public IThermalShutdownListener {
public:
    void onShutdownEnabled() override;
    void onShutdownDisabled() override;
    void onImminentShutdownEnablement(uint32_t imminentDuration) override;
    void onServiceStatusChange(ServiceStatus status) override;
};
```

2. Get thermal factory and thermal shutdown manager instances

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

3. Check if thermal shutdown management service is ready.

```
bool subSystemsStatus = thermShutdownMgr_->isReady();
 if(subSystemsStatus) {
     std::cout << " Thermal-Shutdown management service is ready."<< std::endl;
 } else {
     std::cout << " Thermal-Shutdown management service is NOT ready."<< std::endl;
 }
```

4. If Thermal shutdown management service is not ready, wait for it to be ready

```
std::future<bool> f = thermShutdownMgr_->onSubsystemReady();
#if  //Timeout based wait
    if (f.wait_for(std::chrono::seconds(timeoutSec)) == std::future_status::timeout) {
        std::cout << "operation timed out." << std::endl;
    } else {
        subSystemsStatus = f.get();
        if (subSystemsStatus) {
            std::cout << " Thermal-Shutdown management service is ready." << std::endl;
        }
    }
#else //Unconditional wait
    subSystemsStatus = f.get();
    if (subSystemsStatus) {
        std::cout << " Thermal-Shutdown management service is ready." << std::endl;
    } else {
        std::cout << " Thermal-Shutdown management service is NOT ready." << std::endl;
        return -1;
    }
#endif
```

5. Instantiate MyThermalStateListener

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

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

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

7. Wait for the Thermal auto shutdown mode updates

```
// Avoid long blocking calls when handling notifications
void MyThermalShutdownModeListener::onShutdownEnabled() {
    std::cout << std::endl << "**** Thermal auto shutdown mode enabled ****" << std::endl;
}
void MyThermalShutdownModeListener::onShutdownDisabled() {
    std::cout << std::endl << "**** Thermal auto shutdown mode disabled ****" << std::endl;
}
void MyThermalShutdownModeListener::onImminentShutdownEnablement(uint32_t imminentDuration) {
    std::cout << std::endl << "**** Thermal auto shutdown mode will be enabled in "
        << imminentDuration << " seconds ****" << std::endl;
}
```

8. When the Thermal shutdown management service goes down, this API is invoked with status UNAVAILABLE. All Thermal auto shutdown mode notifications will be stopped until the status becomes AVAILABLE again

```
void MyThermalShutdownModeListener::onServiceStatusChange(ServiceStatus status) {
    std::cout << std::endl << "**** Thermal-Shutdown management service status update ****" << std::endl;
    // Avoid long blocking calls when handling notifications
}
```

33 Using Thermal Shutdown Manager APIs to set Autoshutdown modes

Please follow below steps to set thermal autoshutdown modes

1. Get thermal factory and thermal shutdown manager instances

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

2. Check if thermal shutdown management service is ready.

```
bool subSystemsStatus = thermShutdownMgr_->isReady();
 if(subSystemsStatus) {
     std::cout << APP_NAME << " Thermal-Shutdown management service is ready."<< std::endl;
 } else {
     std::cout << APP_NAME << " Thermal-Shutdown management service is NOT ready."<< std::endl;
 }
```

3 If Thermal shutdown management service is not ready, wait for it to be ready

```
std::future<bool> f = thermShutdownMgr_->onSubsystemReady();
#if  //Timeout based wait
    if (f.wait_for(std::chrono::seconds(timeoutSec)) == std::future_status::timeout) {
        std::cout << "operation timed out." << std::endl;
    } else {
        subSystemsStatus = f.get();
        if (subSystemsStatus) {
            std::cout << " Thermal-Shutdown management service is ready." << std::endl;
        }
    }
#else //Unconditional wait
    subSystemsStatus = f.get();
    if (subSystemsStatus) {
        std::cout << " Thermal-Shutdown management service is ready." << std::endl;
    } else {
        std::cout << " Thermal-Shutdown management service is NOT ready" << std::endl;
        return -1;
    }
#endif
```

3. Query the current thermal auto shutdown mode

```
// Callback which provides response to query operation
void getStatusCallback(AutoShutdownMode mode)
{
    if(mode == AutoShutdownMode::ENABLE) {
        std::cout << " Current auto shutdown mode is: Enable" << std::endl;
    } else if(mode == AutoShutdownMode::DISABLE) {
        std::cout << " Current auto shutdown mode is: Disable" << std::endl;
    } else {
        std::cout << " *** ERROR - Failed to send get auto-shutdown mode " << std::endl;
    }
}
```

```
// Send get themal auto Shutdown mode command
auto status = thermShutdownMgr_->getAutoShutdownMode(getStatusCallback);
if(status != telux::common::Status::SUCCESS) {
    std::cout << "getShutdownMode command failed with error" << static_cast<int>(status) << std::endl;
} else {
    std::cout << "Request to query thermal shutdown status sent" << std::endl;
}
```

4. Set thermal auto shutdown mode

```
// Callback which provides response to set thermal auto shutdown mode command
void commandResponse(ErrorCode error)
{
    if(error == ErrorCode::SUCCESS) {
    std::cout << " sent successfully" << std::endl;
    } else {
        std::cout << " failed\n errorCode: " << static_cast<int>(error) << std::endl;
    }
}
// Send set themal auto Shutdown mode command
auto status = thermShutdownMgr_->setAutoShutdownMode(state, commandResponse);
if(status != telux::common::Status::SUCCESS) {
    std::cout << "setShutdownMode command failed with error" << static_cast<int>(status) << std::endl;
} else {
    std::cout << "Request to set thermal shutdown status sent" << std::endl;
}
```

34 Using Thermal Manager APIs

Please follow below steps to get thermal zones and cooling devices

1. Get thermal factory and thermal manager instances

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

2. Send get thermal zones request using thermal manager object

```
if(thermalMgr != nullptr) {
   std::vector<std::shared_ptr<telux::therm::IThermalZone>> zoneInfo
      = thermalMgr->getThermalZones();
   if(zoneInfo.size() > 0) {
      for(auto index = 0; index < zoneInfo.size(); index++) {
         std::cout << "Thermal zone Id: " << zoneInfo(index)->getId() << "Description: "
                   << zoneInfo(index)->getDescription() << "Current temp: "
                   << zoneInfo(index)->getCurrentTemp() << std::endl;
         std::cout << std::endl;
      }
   } else {
      std::cout << "No thermal zones found!" << std::endl;
   }
} else {
   std::cout << "Thermal manager is nullptr" << std::endl;
}
```

3. Send get cooling devices request using thermal manager object

```
if(thermalMgr != nullptr) {
   std::vector<std::shared_ptr<telux::therm::ICoolingDevice>> coolingDevice
      = thermalMgr->getCoolingDevices();
   if(coolingDevice.size() > 0) {
      for(auto index = 0; index < coolingDevice.size(); index++) {
         std::cout << "Cooling device Id: " << coolingDevice(index)->getId()
                   << "Description: " << coolingDevice(index)->getDescription()
                   << "Max cooling level: " << coolingDevice(index)->getMaxCoolingLevel()
                   << "Current cooling level: " << coolingDevice(index)->getCurrentCoolingLevel()
                   << std::endl;
         std::cout << std::endl;
      }
   } else {
      std::cout << "No cooling devices found!" << std::endl;
   }
} else {
   std::cout << "Thermal manager is nullptr" << std::endl;
}
```

35 Using TCU Activity Manager APIs to get TCU activity state updates

The below steps need to be followed by applications to listen to TCU-activity state notifications, for performing any tasks before the state transition.

1. Implement ITcuActivityListener and IServiceStatusListener interface

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

2. Get the Power-Factory instance

```
auto &powerFactory = PowerFactory::getInstance();
```

3. Get TCU-activity manager instance

```
auto tcuActivityManager = powerFactory.getTcuActivityManager();
```

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

```
bool isReady = tcuActivityManager->isReady();
if(!isReady) {
    std::cout << "TCU-activity management service is not ready" << std::endl;
    std::cout << "Waiting uncondotionally for it to be ready " << std::endl;
    std::future<bool> f = tcuActivityManager->onReady();
    isReady = f.get();
}
```

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

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

6. Instantiate MyTcuActivityStateListener

```
auto myTcuStateListener = std::make_shared<MyTcuActivityStateListener>();
```

7. Register for updates on TCU-activity state and its management service status

```
tcuActivityManager->registerListener(myTcuStateListener);
tcuActivityManager->registerServiceStateListener(myTcuStateListener);
```

8. Wait for the TCU-activity state updates

```
void MyTcuActivityStateListener::onTcuActivityStateUpdate(TcuActivityState state) {
    std::cout << std::endl << "********* TCU-activity state update *********" << std::endl;
    // Avoid long blocking calls when handling notifications
}
```

9. On SUSPEND/SHUTDOWN notification, save any required information and send one(despite multiple listeners) acknowledgement

```
tcuActivityManager->sendActivityStateAck(TcuActivityStateAck);
```

10. When the TCU-activity management service goes down, this API is invoked with status UNAVAILABLE. All TCU-activity state notifications will be stopped until the status becomes AVAILABLE again

```
void MyTcuActivityStateListener::onServiceStatusChange(ServiceStatus status) {
    std::cout << std::endl << "****** TCU-activity management service status update ******" << std::endl;
    // Avoid long blocking calls when handling notifications
}
```

36 Using TCU Activity Manager APIs to set the TCU activity state

The below steps need to be followed by the applications that control the TCU-activity state, to change the TCU-activity state.

1. Implement a command response function

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

2. Implement ITcuActivityListener interface.

```
class MyTcuActivityStateListener : public ITcuActivityListener {
public:
    void onTcuActivityStateUpdate(TcuActivityState state) override;
};
```

3. Get the Power-Factory instance

```
auto &powerFactory = PowerFactory::getInstance();
```

4. Get TCU-activity manager instance

```
auto tcuActivityManager = powerFactory.getTcuActivityManager();
```

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

```
bool isReady = tcuActivityManager->isReady();
if(!isReady) {
    std::cout << "TCU-activity management service is not ready" << std::endl;
    std::cout << "Waiting uncondotionally for it to be ready " << std::endl;
    std::future<bool> f = tcuActivityManager->onReady();
    isReady = f.get();
}
```

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

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

7. Instantiate MyTcuActivityStateListener

```
auto myTcuStateListener = std::make_shared<MyTcuActivityStateListener>();
```

8. Register for updates on TCU-activity state and its management service status

```
tcuActivityManager->registerListener(myTcuStateListener);
```

9. Set the TCU-activity state

```
tcuActivityManager->setActivityState(state,&commandResponse);
```

10. Command response callback function is invoked with error code indicating whether the request was SUCCESS or FAILURE

11. This API on the listener is invoked to notify that the TCU-activity state is changing to the desired state

```
void MyTcuActivityStateListener::onTcuActivityStateUpdate(TcuActivityState state) {
    std::cout << std::endl << "********* TCU-activity state update *********" << std::endl;
    // Avoid long blocking calls when handling notifications
}
```

12. On SUSPEND/SHUTDOWN notification, save any required information and send one(despite multiple listeners) acknowledgement

```
tcuActivityManager->sendActivityStateAck(TcuActivityStateAck);
```