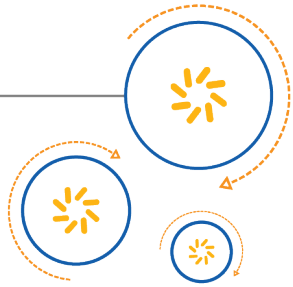
Qualcomm Technologies, Inc.



Telematics SDK

User Guide
v1.10.0

80-PF458-1

October 24, 2018

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

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 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

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 -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

```
LOG_FILE_PATH=/data
```

5. Set log file name

Specifies the name of the log file to be used

```
LOG_FILE_NAME=tel.log
```

6 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;
}
```

7 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;
}
```

8 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.

9 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);
   }
}
```

10 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;
}
```

11 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;
```

12 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

13 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;
```

14 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);
```

15 Using Location Service APIs

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

1. Implement ICommandResponseCallback interface for receiving notifications

```
class MyLocationCommandResponseCallback : public ICommandResponseCallback {
public:
   MyLocationCommandResponseCallback() {
   }

   void commandResponse(ErrorCode error) override;
};
```

2. Implement ILocationListener interface

```
class MyLocationListener : public ILocationListener {
public:
   void onLocationUpdate(const std::shared_ptr<ILocationInfo> &locationInfo) override;

   void onGnssSVInfo(const std::shared_ptr<IGnssSVInfo> &gnssSVInfo) 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" << std::endl;
   std::cout << "wait uncondotionally for it to be ready " << 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 << " *** Sub Systems 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 and SV info updates

```
locationManager->registerListener(myLocationListener);
```

9. Wait for Location fix and SV info

Following configuration values will be used by default which can be changed through configuration APIs

- HORIZONTAL_ACCURACY_LEVEL = 1 // Low
- FINAL_REPORT_INTERVAL = 1000 // 1 second
- POSITION_REPORT_TIMEOUT = 255000 // 255 seconds

```
void MyLocationListener::onLocationUpdate(const std::shared_ptr<ILocationInfo> &locationInfo) {
   std::cout << std::endl;
   std::cout << "********************** Location Report ********************" << std::endl;
   time_t realtime;
   realtime = (time_t)(locationInfo->getTimeStamp());
   std::cout << "Timestamp : " << ctime(&realtime) << std::endl;
   std::cout << "Latitude : " << locationInfo->getLatitude() << std::endl;
   std::cout << "Longitude : " << locationInfo->getLongitude() << std::endl;
}

void MyLocationListener::onGnssSVInfo(const std::shared_ptr<IGnssSVInfo> &gnssSVInfo) {
   std::cout << std::endl;
   std::cout << "**************** Satellite Vehicle Infomation ***************" << std::endl;
   std::cout << "Altitude type : " << static_cast<int>(gnssSVInfo->getAltitudeType())
                     << std::endl;
   for(auto svInfo : gnssSVInfo->getSVInfoList()) {
      std::cout << "Constellation : " << static_cast<int>(svInfo->getConstellation())
                        << std::endl;
      std::cout << "Gnss SV Id : " << svInfo->getId() << std::endl;
      std::cout << "SV health htatus : " << static_cast<int>(svInfo->getSVHealthStatus())
                        << std::endl;
      std::cout << "SV status : " << static_cast<int>(svInfo->getStatus()) << std::endl;
      std::cout << "Has ephemeris : " << static_cast<int>(svInfo->getHasEphemeris())
                        << std::endl;
      std::cout << "Has almanac : " << static_cast<int>(svInfo->getHasAlmanac())
                        << std::endl;
      std::cout << "Elevation : " << svInfo->getElevation() << std::endl;
      std::cout << "Azimuth : " << svInfo->getAzimuth() << std::endl;
      std::cout << "SNR : " << svInfo->getSnr() << std::endl;
   }

   std::cout << "************************************************************" << std::endl;
}
```

10. Instantiate MyLocationCommandResponseCallback

```
auto myLocationCommandCb = std::make_shared<MyLocationCommandResponseCallback>();
```

11. Set minimum interval for reports

```
uint32_t minIntervalForReports = 1500; // Default is 1000 milli seconds.
locationManager->setMinIntervalForReports(minIntervalForReports, myLocationCommandCb);
```

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

```
MyLocationCommandResponseCallback::commandResponse(ErrorCode error) {
   if(error == ErrorCode::SUCCESS) {
      std::cout << "Set min interval for reports request is successful ";
   } else {
      std::cout << "Set min interval for reports request failed with error " << static_cast<int>(error);
   }
}
```

13. Set position timeout

```
uint32_t timeout = 200000;
locationManager->setPositionReportTimeout(timeout, myLocationCommandCb);
```

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

```
MyLocationCommandResponseCallback::commandResponse(ErrorCode error) {
   if(error == ErrorCode::SUCCESS) {
      std::cout << "Set position report timeout request is successful ";
   } else {
      std::cout << "Set position report timeout reports request failed with error " << static_cast<int>(
      error);
   }
}
```

15. Configuring horizontal accuracy level

```
HorizontalAccuracyLevel accuracy = HorizontalAccuracyLevel::MEDIUM;
locationManager->setHorizontalAccuracyLevel(accuracy, myLocationCommandCb);
```

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

```
MyLocationCommandResponseCallback::commandResponse(ErrorCode error) {
   if(error == ErrorCode::SUCCESS) {
      std::cout << "Set horizontal accuracy level request is successful ";
   } else {
      std::cout << "Set horizontal accuracy request failed with error " << static_cast<int>(error);
   }
}
```

17. Observe that changes in the configuration parameters have corresponding effect on how the Location and SV Info reports are received.

16 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

17 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

18 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

19 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