



Qualcomm Technologies, Inc.

# Telematics SDK

## Interface Specification v1.33.79

80-PF458-2 Rev. H

March 22, 2024

**Confidential and Proprietary - Qualcomm Technologies, Inc.**

**NO PUBLIC DISCLOSURE PERMITTED:** Please report postings of this document on public servers or websites to:  
[DocCtrlAgent@qualcomm.com](mailto:DocCtrlAgent@qualcomm.com)

**Restricted Distribution:** Not to be distributed to anyone who is not an employee of either Qualcomm Technologies, Inc. or its affiliated companies without the express approval of Qualcomm Configuration Management.

Not to be used, copied, reproduced, or modified in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies, Inc.

All Qualcomm products mentioned herein are products of Qualcomm Technologies, Inc. and/or its subsidiaries.

Qualcomm is a trademark of Qualcomm Incorporated, registered in the United States and other countries. Other product and brand names may be trademarks or registered trademarks of their respective owners.

This technical data may be subject to U.S. and international export, re-export, or transfer ("export") laws. Diversion contrary to U.S. and international law is strictly prohibited.

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

# Revision History

Date	Description
Sep 2017	Initial release
Dec 2017	Added subscription APIs and section on Versioning and API Status
Jun 2018	Added Cellular Connection Management APIs
Oct 2018	Addition of Network Selection and Serving System Management APIs and call flows
Nov 2018	Addition of C-V2X APIs
Jan 2019	Addition of Audio APIs and call flows
Mar 2019	Addition of Thermal manager APIs and call flows
May 2019	Addition of TCU Activity Management APIs and call flows
Jun 2019	Addition of Audio APIs for play, capture, DTMF and related call flows
Jul 2019	Updated the Location APIs and call flows
Jul 2019	Addition of Thermal shutdown manager APIs and call flows
Sep 2019	Addition of Remote SIM APIs and call flows
Sep 2019	Addition of Audio APIs for Loopback, Tone Generator and related call flows
Sep 2019	Addition of Audio APIs for compressed audio format playback and related call flows
Sep 2019	Addition of modem config APIs and related call flows
Oct 2019	Addition of Data Filter APIs and call flows
Oct 2019	Addition of Location concurrent report APIs and call flows
Oct 2019	Addition of Location constraint time uncertainty APIs and call flows
Nov 2019	Addition of Audio Format Transcoding APIs and call flows
Nov 2019	Addition of Data Networking APIs and call flows
Nov 2019	Addition of Compressed audio format playback on voice paths APIs and call flows
Jan 2020	Addition of Data software bridge management APIs and call flows
Jan 2020	Addition of Socks Proxy to Data Networking APIs and call flows
Feb 2020	Addition of Location Configurator APIs and call flows
Mar 2020	Addition of Robust Location API in Location Configurator and call flows
Apr 2020	Addition of Security section with SE-linux interface documentation
Sep 2020	Addition of Location Manager APIs, Location Configurator APIs and callflows
Apr 2021	Addition of terrestrial positioning APIs and callflows in location SDK
Feb 2022	Support for Xtra feature in location.
Aug 2022	Addition of Wlan management support and call flow

# Contents

---

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Purpose	5
1.2	Scope	5
1.3	Conventions	5
1.4	SDK Versioning	5
1.5	Public API Status	6
<b>2</b>	<b>Functional Overview</b>	<b>7</b>
2.1	Overview	7
2.2	Features	9
2.2.1	Call Management	9
2.2.2	SMS	9
2.2.3	SIM Card Services	9
2.2.4	Phone Information	10
2.2.5	Location Services	10
2.2.6	Data Services	10
2.2.7	Network Selection and Serving System Management	11
2.2.8	C-V2X	12
2.2.9	Audio	12
2.2.10	Thermal Management	15
2.2.11	Thermal Shutdown Management	15
2.2.12	TCU Activity Management	15
2.2.13	Remote SIM	15
2.2.14	Modem Config Management	16
2.2.15	Wlan Management	16
2.3	Security	16
2.3.1	SELinux	17
<b>3</b>	<b>Call Flow Diagrams</b>	<b>18</b>
3.1	Application initialization call flow	18
3.1.1	Phone manager initialization	18
3.2	Telephony	19
3.2.1	Dial call flow	19
3.2.2	ECall call flow	20
3.2.3	Signal strength call flow	22
3.2.4	Answer, Reject, RejectWithSMS call flow	23
3.2.5	Hold call flow	24
3.2.6	Hold, Conference, Swap call flow	25
3.2.7	SMS call flow	27

3.2.8	Radio and Service state call flow	28
3.2.9	Network Selection Manager call flow	29
3.2.10	Serving System Manager Call Flow	31
3.3	Card Services	32
3.3.1	Get applications call flow	33
3.3.2	Transmit APDU call flow	34
3.3.2.1	On logical channel	34
3.3.2.2	On basic channel	35
3.3.3	SAP card manager call flow	35
3.3.3.1	Request card reader status, Request ATR, Transmit APDU call flow	36
3.3.3.2	SIM Turn off, Turn on and Reset call flow	38
3.3.4	Subscription Call flow	39
3.3.4.1	Subscription initialization	40
3.3.4.2	Subscription call flow	41
3.4	Call flow for location services	42
3.4.1	Call flow to register/remove listener for generating basic reports	42
3.4.2	Call flow to register/remove listener for generating detailed reports	44
3.4.3	Call flow to register/remove listener for generating detailed engine reports	46
3.4.4	Call flow to register/remove listener for system info updates	47
3.4.5	Call flow to request energy consumed information	48
3.4.6	Call flow to get terrestrial positioning information	49
3.4.7	Call flow to cancel terrestrial positioning information	49
3.4.8	Call flow to enable/disable constraint time uncertainty	50
3.4.9	Call flow to enable/disable PACE	50
3.4.10	Call flow to delete all aiding data	51
3.4.11	Call flow to configure lever arm parameters	51
3.4.12	Call flow to configure blacklisted constellations	52
3.4.13	Call flow to configure robust location	52
3.4.14	Call flow to configure min gps week	53
3.4.15	Call flow to request min gps week	53
3.4.16	Call flow to delete specified data	54
3.4.17	Call flow to configure min sv elevation	54
3.4.18	Call flow to request min sv elevation	55
3.4.19	Call flow to request robust location	55
3.4.20	Call flow to configure dead reckoning engine	56
3.4.21	Call flow to provide user consent for terrestrial positioning	56
3.4.22	Call flow to represent Xtra Feature	57
3.5	Data	58
3.5.1	Start/Stop for data connection manager call flow	58
3.5.2	Request/Create/Delete/Modify for data profile manager call flow	60
3.5.3	Data Filter Manager Call Flow	61
3.5.3.1	Call flow to Set/Get data filter mode	62
3.5.3.2	Call flow to Add data restrict filter	63
3.5.3.3	Call flow to Remove data restrict filter	64
3.5.4	Data Networking Call Flow	65
3.5.4.1	Create VLAN and Bind it to PDN in data vlan manager call flow	66
3.5.4.2	LAN-LAN VLAN Configuration from EAP usecase call flow	67
3.5.4.3	LAN-LAN VLAN Configuration from A7 usecase call flow	68
3.5.4.4	LAN-WAN VLAN Configuration from EAP usecase call flow	69

3.5.4.5	LAN-WAN VLAN Configuration from A7 usecase call flow . . . . .	71
3.5.4.6	Create Static NAT entry in data Static NAT manager call flow . . . . .	72
3.5.4.7	Firewall Enablement in data Firewall manager call flow . . . . .	73
3.5.4.8	Add Firewall Entry in data Firewall manager call flow . . . . .	74
3.5.4.9	Set Firewall DMZ in data Firewall manager call flow . . . . .	75
3.5.4.10	Socks Enablement in data Socks manager call flow . . . . .	76
3.5.4.11	Call flow to add and enable software bridge . . . . .	77
3.5.4.12	Call flow to remove and disable software bridge . . . . .	78
3.6	C-V2X . . . . .	79
3.6.1	Start/Stop C-V2X Mode . . . . .	79
3.6.2	C-V2X Radio Manager API . . . . .	80
3.6.3	C-V2X Radio Initialization . . . . .	81
3.6.4	C-V2X Radio RX subscription . . . . .	82
3.6.5	C-V2X Radio TX event-driven flow . . . . .	83
3.6.6	C-V2X Radio TX SPS flow . . . . .	84
3.7	Audio . . . . .	85
3.7.1	Audio Manager API call flow . . . . .	86
3.7.2	Audio Voice Call Start/Stop call flow . . . . .	88
3.7.3	Audio Voice Call Device Switch call flow . . . . .	90
3.7.4	Audio Voice Call Volume/Mute control call flow . . . . .	92
3.7.5	Call flow to play DTMF tone . . . . .	94
3.7.6	Call flow to detect DTMF tones . . . . .	96
3.7.7	Audio Playback call flow . . . . .	98
3.7.8	Audio Capture call flow . . . . .	100
3.7.9	Audio Tone Generator call flow . . . . .	102
3.7.10	Audio Loopback call flow . . . . .	103
3.7.11	Compressed audio format playback call flow . . . . .	105
3.7.12	Audio Transcoding Operation Callflow . . . . .	107
3.7.13	Compressed audio format playback on Voice Paths Callflow . . . . .	109
3.8	Thermal manager call flow . . . . .	111
3.9	Thermal shutdown management . . . . .	111
3.9.1	Call flow to register/remove listener for Thermal auto-shutdown mode updates. . . . .	112
3.9.2	Call flow to set/get the Thermal auto-shutdown mode . . . . .	113
3.9.3	Call flow to manage thermal auto-shutdown from an eCall application. . . . .	114
3.10	TCU Activity Management . . . . .	115
3.10.1	Call flow to register/remove listener for TCU-activity manager . . . . .	116
3.10.2	Call flow to set the TCU-activity state . . . . .	117
3.11	Remote SIM call flow . . . . .	118
3.12	Modem Config Call Flow . . . . .	119
3.12.1	Call flow to load and activate a modem config file. . . . .	120
3.12.2	Call flow to deactivate and delete a modem config file. . . . .	121
3.12.3	Call flow to set and get config auto selection mode . . . . .	122
3.13	WLAN . . . . .	123
3.13.1	Call flow to Modify WLAN Configuration . . . . .	124
3.13.2	Call flow to Modify WLAN Station Configuration . . . . .	126
3.13.3	Call flow to Modify WLAN Access Point Configuration . . . . .	127
<b>4</b>	<b>Deprecated List . . . . .</b>	<b>129</b>
<b>5</b>	<b>Interfaces . . . . .</b>	<b>133</b>

5.1	Telematics SDK APIs	133
5.2	Phone Factory	134
5.2.1	Data Structure Documentation	134
5.2.1.1	class telux::tel::PhoneFactory	134
5.3	Phone	139
5.3.1	Data Structure Documentation	139
5.3.1.1	class telux::tel::GsmCellIdentity	139
5.3.1.2	class telux::tel::CdmaCellIdentity	140
5.3.1.3	class telux::tel::LteCellIdentity	142
5.3.1.4	class telux::tel::WcdmaCellIdentity	143
5.3.1.5	class telux::tel::TdscdmaCellIdentity	145
5.3.1.6	class telux::tel::CellInfo	146
5.3.1.7	class telux::tel::GsmCellInfo	147
5.3.1.8	class telux::tel::CdmaCellInfo	148
5.3.1.9	class telux::tel::LteCellInfo	149
5.3.1.10	class telux::tel::WcdmaCellInfo	150
5.3.1.11	class telux::tel::TdscdmaCellInfo	151
5.3.1.12	struct telux::tel::ECallMsdOptionals	152
5.3.1.13	struct telux::tel::ECallMsdControlBits	152
5.3.1.14	struct telux::tel::ECallVehicleIdentificationNumber	152
5.3.1.15	struct telux::tel::ECallVehiclePropulsionStorageType	153
5.3.1.16	struct telux::tel::ECallVehicleLocation	153
5.3.1.17	struct telux::tel::ECallVehicleLocationDelta	153
5.3.1.18	struct telux::tel::ECallOptionalPdu	154
5.3.1.19	struct telux::tel::ECallMsdData	154
5.3.1.20	struct telux::tel::ECallModelInfo	154
5.3.1.21	struct telux::tel::ECallHlapTimerStatus	155
5.3.1.22	struct telux::tel::ECallHlapTimerEvents	155
5.3.1.23	struct telux::tel::EcallConfig	156
5.3.1.24	class telux::tel::IPhone	156
5.3.1.25	class telux::tel::ISignalStrengthCallback	160
5.3.1.26	class telux::tel::IVoiceServiceStateCallback	161
5.3.1.27	struct telux::tel::SimRatCapability	161
5.3.1.28	struct telux::tel::CellularCapabilityInfo	162
5.3.1.29	class telux::tel::IPhoneListener	162
5.3.1.30	class telux::tel::IPhoneManager	165
5.3.1.31	class telux::tel::ICellularCapabilityCallback	168
5.3.1.32	class telux::tel::IOperatingModeCallback	169
5.3.1.33	class telux::tel::SignalStrength	170
5.3.1.34	class telux::tel::LteSignalStrengthInfo	172
5.3.1.35	class telux::tel::GsmSignalStrengthInfo	174
5.3.1.36	class telux::tel::CdmaSignalStrengthInfo	175
5.3.1.37	class telux::tel::WcdmaSignalStrengthInfo	177
5.3.1.38	class telux::tel::TdscdmaSignalStrengthInfo	178
5.3.1.39	class telux::tel::Nr5gSignalStrengthInfo	178
5.3.1.40	class telux::tel::VoiceServiceInfo	179
5.3.2	Enumeration Type Documentation	181
5.3.2.1	CellType	181
5.3.2.2	ECallVariant	181
5.3.2.3	EmergencyCallType	181

	5.3.2.4	ECallMsdTransmissionStatus	181
	5.3.2.5	ECallCategory	182
	5.3.2.6	ECallVehicleType	182
	5.3.2.7	ECallOptionalDataType	182
	5.3.2.8	ECallMode	182
	5.3.2.9	ECallModeReason	182
	5.3.2.10	HlapTimerStatus	183
	5.3.2.11	HlapTimerEvent	183
	5.3.2.12	ECallNumType	183
	5.3.2.13	EcallConfigType	183
	5.3.2.14	RadioState	184
	5.3.2.15	ServiceState	184
	5.3.2.16	RadioTechnology	184
	5.3.2.17	RATCapability	185
	5.3.2.18	VoiceServiceTechnology	185
	5.3.2.19	OperatingMode	185
	5.3.2.20	SignalStrengthLevel	186
	5.3.2.21	VoiceServiceState	186
	5.3.2.22	VoiceServiceDenialCause	186
5.4	Call		188
	5.4.1	Data Structure Documentation	188
	5.4.1.1	class telux::tel::ICall	188
	5.4.1.2	class telux::tel::ICallListener	194
	5.4.1.3	class telux::tel::ICallManager	196
	5.4.1.4	class telux::tel::IMakeCallCallback	206
	5.4.2	Enumeration Type Documentation	206
	5.4.2.1	CallDirection	206
	5.4.2.2	CallState	207
	5.4.2.3	CallEndCause	207
5.5	SMS		209
	5.5.1	Data Structure Documentation	209
	5.5.1.1	struct telux::tel::MessageAttributes	209
	5.5.1.2	struct telux::tel::MessagePartInfo	209
	5.5.1.3	class telux::tel::SmsMessage	209
	5.5.1.4	class telux::tel::ISmsManager	212
	5.5.1.5	class telux::tel::ISmsListener	216
	5.5.1.6	class telux::tel::ISmscAddressCallback	218
	5.5.2	Enumeration Type Documentation	218
	5.5.2.1	SmsEncoding	218
5.6	SIM Card Services		219
	5.6.1	Data Structure Documentation	219
	5.6.1.1	class telux::tel::ICardApp	219
	5.6.1.2	struct telux::tel::lccResult	221
	5.6.1.3	class telux::tel::ICardManager	222
	5.6.1.4	class telux::tel::ICard	224
	5.6.1.5	class telux::tel::ICardChannelCallback	228
	5.6.1.6	class telux::tel::ICardCommandCallback	229
	5.6.1.7	class telux::tel::ICardListener	229
	5.6.1.8	struct telux::tel::CardReaderStatus	230
	5.6.1.9	class telux::tel::ISapCardManager	230

5.6.1.10	class telux::tel::IAtrResponseCallback	236
5.6.1.11	class telux::tel::ISapCardCommandCallback	236
5.6.1.12	class telux::tel::ICardReaderCallback	237
5.6.1.13	class telux::tel::ISapCardListener	237
5.6.2	Enumeration Type Documentation	238
5.6.2.1	CardState	238
5.6.2.2	CardError	238
5.6.2.3	CardLockType	238
5.6.2.4	AppType	238
5.6.2.5	AppState	239
5.6.2.6	SapState	239
5.6.2.7	SapCondition	239
5.7	Location Services	240
5.7.1	Data Structure Documentation	240
5.7.1.1	class telux::loc::IDgnssStatusListener	240
5.7.1.2	class telux::loc::IDgnssManager	240
5.7.1.3	class telux::loc::ILocationConfigurator	243
5.7.1.4	struct telux::loc::GnssKinematicsData	257
5.7.1.5	struct telux::loc::LLAInfo	257
5.7.1.6	struct telux::loc::TimeInfo	258
5.7.1.7	struct telux::loc::GlonassTimeInfo	258
5.7.1.8	union telux::loc::SystemTimeInfo	259
5.7.1.9	struct telux::loc::SystemTime	259
5.7.1.10	struct telux::loc::GnssMeasurementInfo	260
5.7.1.11	struct telux::loc::SvUsedInPosition	260
5.7.1.12	struct telux::loc::GnssData	260
5.7.1.13	struct telux::loc::SvBlackListInfo	261
5.7.1.14	struct telux::loc::LeverArmParams	261
5.7.1.15	struct telux::loc::ExternalLocationInfo	262
5.7.1.16	struct telux::loc::GnssMeasurementsData	262
5.7.1.17	struct telux::loc::GnssMeasurementsClock	263
5.7.1.18	struct telux::loc::GnssMeasurements	264
5.7.1.19	struct telux::loc::LeapSecondChangeInfo	264
5.7.1.20	struct telux::loc::LeapSecondInfo	265
5.7.1.21	struct telux::loc::LocationSystemInfo	265
5.7.1.22	struct telux::loc::GnssEnergyConsumedInfo	265
5.7.1.23	struct telux::loc::RobustLocationVersion	265
5.7.1.24	struct telux::loc::RobustLocationConfiguration	266
5.7.1.25	struct telux::loc::NmeaConfig	266
5.7.1.26	struct telux::loc::BodyToSensorMountParams	266
5.7.1.27	struct telux::loc::DREngineConfiguration	267
5.7.1.28	struct telux::loc::XtraConfig	267
5.7.1.29	struct telux::loc::XtraStatus	269
5.7.1.30	class telux::loc::ILocationInfoBase	269
5.7.1.31	class telux::loc::ILocationInfoEx	273
5.7.1.32	class telux::loc::ISVInfo	282
5.7.1.33	class telux::loc::IGnssSVInfo	285
5.7.1.34	class telux::loc::IGnssSignalInfo	285
5.7.1.35	class telux::loc::LocationFactory	286
5.7.1.36	class telux::loc::ILocationListener	287



5.7.1.37	class telux::loc::ILocationSystemInfoListener	289
5.7.1.38	class telux::loc::ILocationInjectionListener	290
5.7.1.39	class telux::loc::ILocationConfigListener	290
5.7.1.40	class telux::loc::ILocationManager	291
5.7.2	Enumeration Type Documentation	299
5.7.2.1	DgnssDataFormat	299
5.7.2.2	DgnssStatus	299
5.7.2.3	HorizontalAccuracyLevel	299
5.7.2.4	LocationReliability	299
5.7.2.5	SbasCorrectionType	300
5.7.2.6	AltitudeType	300
5.7.2.7	GnssConstellationType	300
5.7.2.8	SVHealthStatus	301
5.7.2.9	SVStatus	301
5.7.2.10	SVInfoAvailability	301
5.7.2.11	GnssPositionTechType	301
5.7.2.12	KinematicDataValidityType	302
5.7.2.13	GnssSystem	302
5.7.2.14	GnssTimeValidityType	302
5.7.2.15	GlonassTimeValidity	303
5.7.2.16	GnssSignalType	303
5.7.2.17	LocationTechnologyType	303
5.7.2.18	LocationValidityType	304
5.7.2.19	LocationInfoExValidityType	304
5.7.2.20	GnssDataSignalTypes	305
5.7.2.21	GnssDataValidityType	306
5.7.2.22	DrCalibrationStatusType	306
5.7.2.23	LocReqEngineType	306
5.7.2.24	LocationAggregationType	306
5.7.2.25	PositioningEngineType	307
5.7.2.26	LeverArmType	307
5.7.2.27	GnssMeasurementsDataValidityType	307
5.7.2.28	GnssMeasurementsStateValidityType	308
5.7.2.29	GnssMeasurementsAdrStateValidityType	308
5.7.2.30	GnssMeasurementsMultipathIndicator	308
5.7.2.31	GnssMeasurementsClockValidityType	308
5.7.2.32	LeapSecondInfoValidityType	309
5.7.2.33	LocationSystemInfoValidityType	309
5.7.2.34	GnssEnergyConsumedInfoValidityType	309
5.7.2.35	AidingDataType	309
5.7.2.36	TerrestrialTechnologyType	310
5.7.2.37	RobustLocationConfigType	310
5.7.2.38	NmeaSentenceType	310
5.7.2.39	GeodeticDatumType	310
5.7.2.40	DRConfigValidityType	311
5.7.2.41	GnssReportType	311
5.7.2.42	EngineType	311
5.7.2.43	ReportStatus	311
5.7.2.44	DebugLogLevel	312
5.7.2.45	XtraDataStatus	312

	5.7.2.46	LocConfigIndicationsType	312
5.7.3		Variable Documentation	312
	5.7.3.1	UNKNOWN_CARRIER_FREQ	312
	5.7.3.2	UNKNOWN_SIGNAL_MASK	313
	5.7.3.3	UNKNOWN_TIMESTAMP	313
	5.7.3.4	DEFAULT_TUNC_THRESHOLD	313
	5.7.3.5	DEFAULT_TUNC_ENERGY_THRESHOLD	313
	5.7.3.6	DEFAULT_GNSS_REPORT	313
5.8		Data Services	314
5.8.1		Define Documentation	314
	5.8.1.1	PROFILE_ID_MAX	314
	5.8.1.2	MAX_QOS_FILTERS	314
	5.8.1.3	IP_PROT_UNKNOWN	314
5.8.2		Data Structure Documentation	314
	5.8.2.1	class telux::data::IDataConnectionManager	314
	5.8.2.2	class telux::data::IDataCall	319
	5.8.2.3	class telux::data::IDataConnectionListener	323
	5.8.2.4	struct telux::data::DataRestrictMode	324
	5.8.2.5	struct telux::data::PortInfo	324
	5.8.2.6	struct telux::data::ProfileParams	325
	5.8.2.7	struct telux::data::DataCallStats	325
	5.8.2.8	struct telux::data::IpAddrInfo	325
	5.8.2.9	struct telux::data::DataCallEndReason	326
	5.8.2.10	struct telux::data::LatencyConfig	326
	5.8.2.11	struct telux::data::BackhaulInfo	326
	5.8.2.12	struct telux::data::VlanConfig	326
	5.8.2.13	struct telux::data::FlowDataRate	326
	5.8.2.14	struct telux::data::QosIPFlowInfo	327
	5.8.2.15	class telux::data::DataFactory	327
	5.8.2.16	class telux::data::IDataFilterListener	332
	5.8.2.17	class telux::data::IDataFilterManager	333
	5.8.2.18	class telux::data::DataProfile	339
	5.8.2.19	class telux::data::IDataProfileListener	341
	5.8.2.20	class telux::data::IDataProfileManager	342
	5.8.2.21	class telux::data::IDataCreateProfileCallback	346
	5.8.2.22	class telux::data::IDataProfileListCallback	347
	5.8.2.23	class telux::data::IDataProfileCallback	347
	5.8.2.24	class telux::data::IDataSettingsManager	348
	5.8.2.25	class telux::data::IDataSettingsListener	351
	5.8.2.26	struct telux::data::RoamingStatus	351
	5.8.2.27	struct telux::data::ServiceStatus	352
	5.8.2.28	class telux::data::IServingSystemManager	352
	5.8.2.29	class telux::data::IServingSystemListener	355
	5.8.2.30	union telux::data::DataCallEndReason.__unnamed__	356
5.8.3		Enumeration Type Documentation	357
	5.8.3.1	IpFamilyType	357
	5.8.3.2	TechPreference	357
	5.8.3.3	AuthProtocolType	357
	5.8.3.4	DataRestrictModeType	357
	5.8.3.5	DataCallStatus	357

5.8.3.6	DataBearerTechnology	358
5.8.3.7	EndReasonType	358
5.8.3.8	MobileIpReasonCode	359
5.8.3.9	InternalReasonCode	359
5.8.3.10	CallManagerReasonCode	361
5.8.3.11	SpecReasonCode	364
5.8.3.12	PPPReasonCode	366
5.8.3.13	EHRPDReasonCode	366
5.8.3.14	Ipv6ReasonCode	366
5.8.3.15	HandoffReasonCode	367
5.8.3.16	ProfileChangeEvent	367
5.8.3.17	OperationType	367
5.8.3.18	Direction	367
5.8.3.19	InterfaceType	367
5.8.3.20	BackhaulType	368
5.8.3.21	LatencyLevel	368
5.8.3.22	ServiceState	368
5.8.3.23	QosFlowStateChangeEvent	368
5.8.3.24	IpTrafficClassType	368
5.8.3.25	QosIPFlowMaskType	369
5.8.3.26	QosFlowMaskType	369
5.8.3.27	RoamingType	369
5.8.3.28	DataServiceState	369
5.8.3.29	NetworkRat	369
5.9	Subscription Management	371
5.9.1	Data Structure Documentation	371
5.9.1.1	class telux::tel::ISubscription	371
5.9.1.2	class telux::tel::ISubscriptionListener	374
5.9.1.3	class telux::tel::ISubscriptionManager	375
5.10	Network Selection	377
5.10.1	Data Structure Documentation	377
5.10.1.1	struct telux::tel::PreferredNetworkInfo	377
5.10.1.2	struct telux::tel::OperatorStatus	377
5.10.1.3	class telux::tel::INetworkSelectionManager	377
5.10.1.4	class telux::tel::OperatorInfo	381
5.10.1.5	class telux::tel::INetworkSelectionListener	382
5.10.2	Enumeration Type Documentation	383
5.10.2.1	RatType	383
5.10.2.2	NetworkSelectionMode	383
5.10.2.3	InUseStatus	383
5.10.2.4	RoamingStatus	383
5.10.2.5	ForbiddenStatus	383
5.10.2.6	PreferredStatus	384
5.11	Serving System	385
5.11.1	Data Structure Documentation	385
5.11.1.1	struct telux::tel::DcStatus	385
5.11.1.2	class telux::tel::IServingSystemManager	385
5.11.1.3	class telux::tel::IServingSystemListener	389
5.11.2	Enumeration Type Documentation	390
5.11.2.1	ServiceDomainPreference	390

	5.11.2.2	RatPrefType . . . . .	390
	5.11.2.3	EndcAvailability . . . . .	391
	5.11.2.4	DcnrRestriction . . . . .	391
5.12	Common . . . . .		392
	5.12.1	Data Structure Documentation . . . . .	392
	5.12.1.1	class telux::common::ICommandCallback . . . . .	392
	5.12.1.2	class telux::common::ICommandResponseCallback . . . . .	392
	5.12.1.3	class telux::common::DeviceConfig . . . . .	393
	5.12.1.4	class telux::common::Log . . . . .	393
	5.12.1.5	struct telux::common::SdkVersion . . . . .	393
	5.12.1.6	class telux::common::Version . . . . .	393
	5.12.2	Enumeration Type Documentation . . . . .	394
	5.12.2.1	Status . . . . .	394
	5.12.2.2	ErrorCode . . . . .	395
	5.12.2.3	ServiceStatus . . . . .	399
	5.12.2.4	LogLevel . . . . .	399
5.13	C-V2X . . . . .		401
	5.13.1	Data Structure Documentation . . . . .	401
	5.13.1.1	class telux::cv2x::ICv2xConfigListener . . . . .	401
	5.13.1.2	class telux::cv2x::ICv2xConfig . . . . .	401
	5.13.1.3	class telux::cv2x::Cv2xFactory . . . . .	403
	5.13.1.4	class telux::cv2x::ICv2xRadio . . . . .	404
	5.13.1.5	class telux::cv2x::ICv2xRadioListener . . . . .	415
	5.13.1.6	class telux::cv2x::ICv2xRadioManager . . . . .	416
	5.13.1.7	struct telux::cv2x::SocketInfo . . . . .	419
	5.13.1.8	struct telux::cv2x::Cv2xStatus . . . . .	420
	5.13.1.9	struct telux::cv2x::Cv2xPoolStatus . . . . .	420
	5.13.1.10	struct telux::cv2x::Cv2xStatusEx . . . . .	420
	5.13.1.11	struct telux::cv2x::TxPoolIdInfo . . . . .	421
	5.13.1.12	struct telux::cv2x::EventFlowInfo . . . . .	421
	5.13.1.13	struct telux::cv2x::SpsFlowInfo . . . . .	421
	5.13.1.14	struct telux::cv2x::Cv2xRadioCapabilities . . . . .	422
	5.13.1.15	struct telux::cv2x::MacDetails . . . . .	423
	5.13.1.16	struct telux::cv2x::SpsSchedulingInfo . . . . .	424
	5.13.1.17	struct telux::cv2x::TrustedUEInfo . . . . .	424
	5.13.1.18	struct telux::cv2x::TrustedUEInfoList . . . . .	424
	5.13.1.19	struct telux::cv2x::IPv6Address . . . . .	425
	5.13.1.20	struct telux::cv2x::DataSessionSettings . . . . .	425
	5.13.1.21	struct telux::cv2x::ConfigEventInfo . . . . .	425
	5.13.1.22	struct telux::cv2x::RFTxInfo . . . . .	426
	5.13.1.23	struct telux::cv2x::TxStatusReport . . . . .	426
	5.13.1.24	class telux::cv2x::ICv2xRxSubscription . . . . .	427
	5.13.1.25	class telux::cv2x::ICv2xTxFlow . . . . .	429
	5.13.1.26	class telux::cv2x::ICv2xTxRxSocket . . . . .	430
	5.13.2	Enumeration Type Documentation . . . . .	431
	5.13.2.1	TrafficCategory . . . . .	432
	5.13.2.2	Cv2xStatusType . . . . .	432
	5.13.2.3	Cv2xCauseType . . . . .	432
	5.13.2.4	TrafficIpType . . . . .	432
	5.13.2.5	RadioConcurrencyMode . . . . .	433

	5.13.2.6	Cv2xEvent	433
	5.13.2.7	Priority	433
	5.13.2.8	Periodicity	433
	5.13.2.9	ConfigSourceType	434
	5.13.2.10	ConfigEvent	434
	5.13.2.11	RFTxStatus	434
	5.13.2.12	SegmentType	434
	5.13.2.13	TxType	435
5.13.3		Variable Documentation	435
	5.13.3.1	MAX_ANTENNAS_SUPPORTED	435
5.14		Audio	436
	5.14.1	Data Structure Documentation	436
	5.14.1.1	struct telux::audio::FormatParams	436
	5.14.1.2	struct telux::audio::AmrwbpParams	436
	5.14.1.3	struct telux::audio::StreamConfig	436
	5.14.1.4	struct telux::audio::FormatInfo	437
	5.14.1.5	struct telux::audio::ChannelVolume	437
	5.14.1.6	struct telux::audio::StreamVolume	437
	5.14.1.7	struct telux::audio::StreamMute	437
	5.14.1.8	struct telux::audio::StreamBuffer	438
	5.14.1.9	struct telux::audio::DtmfTone	438
	5.14.1.10	class telux::audio::AudioFactory	438
	5.14.1.11	class telux::audio::IVoiceListener	439
	5.14.1.12	class telux::audio::IPlayListener	439
	5.14.1.13	class telux::audio::ITranscodeListener	440
	5.14.1.14	class telux::audio::IAudioListener	440
	5.14.1.15	class telux::audio::IAudioBuffer	441
	5.14.1.16	class telux::audio::IStreamBuffer	442
	5.14.1.17	class telux::audio::IAudioManager	443
	5.14.1.18	class telux::audio::IAudioDevice	446
	5.14.1.19	class telux::audio::IAudioStream	447
	5.14.1.20	class telux::audio::IAudioVoiceStream	449
	5.14.1.21	class telux::audio::IAudioPlayStream	452
	5.14.1.22	class telux::audio::IAudioCaptureStream	454
	5.14.1.23	class telux::audio::IAudioLoopbackStream	455
	5.14.1.24	class telux::audio::IAudioToneGeneratorStream	456
	5.14.1.25	class telux::audio::ITranscoder	457
5.14.2		Enumeration Type Documentation	460
	5.14.2.1	DeviceType	460
	5.14.2.2	DeviceDirection	461
	5.14.2.3	Direction	461
	5.14.2.4	StreamType	461
	5.14.2.5	StreamDirection	461
	5.14.2.6	ChannelType	461
	5.14.2.7	AudioFormat	462
	5.14.2.8	DtmfLowFreq	462
	5.14.2.9	DtmfHighFreq	462
	5.14.2.10	AmrwbpFrameFormat	462
	5.14.2.11	StopType	462
5.15		Thermal Management	464

5.15.1	Data Structure Documentation	464
5.15.1.1	class telux::therm::ThermalFactory	464
5.15.1.2	struct telux::therm::BoundCoolingDevice	465
5.15.1.3	class telux::therm::IThermalManager	465
5.15.1.4	class telux::therm::ITripPoint	466
5.15.1.5	class telux::therm::IThermalZone	467
5.15.1.6	class telux::therm::ICoolingDevice	469
5.15.1.7	class telux::therm::IThermalShutdownListener	470
5.15.1.8	class telux::therm::IThermalShutdownManager	471
5.15.2	Enumeration Type Documentation	474
5.15.2.1	AutoShutdownMode	474
5.15.2.2	TripType	474
5.15.3	Variable Documentation	474
5.15.3.1	DEFAULT_TIMEOUT	474
5.16	TCU Activity Manager	475
5.16.1	Data Structure Documentation	475
5.16.1.1	class telux::power::PowerFactory	475
5.16.1.2	class telux::power::ITcuActivityListener	475
5.16.1.3	class telux::power::ITcuActivityManager	476
5.16.2	Enumeration Type Documentation	480
5.16.2.1	TcuActivityState	480
5.16.2.2	TcuActivityStateAck	480
5.16.2.3	ClientType	480
5.17	Remote SIM Provisioning	481
5.17.1	Data Structure Documentation	481
5.17.1.1	class telux::tel::SimProfile	481
5.17.1.2	class telux::tel::ISimProfileListener	484
5.17.1.3	class telux::tel::ISimProfileManager	485
5.17.2	Enumeration Type Documentation	490
5.17.2.1	ProfileType	490
5.17.2.2	IconType	490
5.17.2.3	ProfileClass	490
5.17.2.4	DownloadStatus	490
5.17.2.5	DownloadErrorCause	491
5.17.2.6	PolicyRuleType	491
5.18	Remote SIM	492
5.18.1	Data Structure Documentation	492
5.18.1.1	class telux::tel::IRemoteSimListener	492
5.18.1.2	class telux::tel::IRemoteSimManager	493
5.18.2	Enumeration Type Documentation	498
5.18.2.1	CardErrorCause	498
5.19	Modem Config	499
5.19.1	Data Structure Documentation	499
5.19.1.1	class telux::config::ConfigFactory	499
5.19.1.2	struct telux::config::ConfigInfo	499
5.19.1.3	class telux::config::IModemConfigListener	500
5.19.1.4	class telux::config::IModemConfigManager	500
5.19.2	Enumeration Type Documentation	504
5.19.2.1	ConfigType	504
5.19.2.2	AutoSelectionMode	505

	5.19.2.3	ConfigUpdateStatus	505
5.20	Wlan		506
	5.20.1	Define Documentation	506
	5.20.1.1	INVALID_AP_ID	506
	5.20.2	Data Structure Documentation	506
	5.20.2.1	struct telux::wlan::ApNetConfig	506
	5.20.2.2	struct telux::wlan::ApConfig	506
	5.20.2.3	struct telux::wlan::DeviceInfo	506
	5.20.2.4	struct telux::wlan::DeviceStats	507
	5.20.2.5	class telux::wlan::IStaInterfaceManager	507
	5.20.2.6	class telux::wlan::IStaListener	507
	5.20.2.7	struct telux::wlan::ApInfo	507
	5.20.2.8	struct telux::wlan::ApNetInfo	508
	5.20.2.9	struct telux::wlan::ApStatus	508
	5.20.2.10	struct telux::wlan::StaStatus	508
	5.20.2.11	struct telux::wlan::InterfaceStatus	508
	5.20.2.12	class telux::wlan::IWlanDeviceManager	509
	5.20.2.13	class telux::wlan::IWlanListener	509
	5.20.2.14	class telux::wlan::WlanFactory	509
	5.20.3	Enumeration Type Documentation	511
	5.20.3.1	ApInterworking	511
	5.20.3.2	ApDeviceConnectionEvent	511
	5.20.3.3	BandType	511
	5.20.3.4	ConnectionStatus	512
	5.20.3.5	Id	512
	5.20.3.6	ApType	512
	5.20.3.7	StaInterfaceStatus	512
	5.20.3.8	OperationType	512
	5.20.3.9	IpFamilyType	513
	5.20.3.10	ServiceOperation	513
	5.20.3.11	InterfaceState	513
	5.20.3.12	HwDeviceType	513
	5.20.4	Function Documentation	513
	5.20.4.1	setConfig(ApConfig config)=0	514
	5.20.4.2	getConfig(std::vector< ApConfig > &config)=0	514
	5.20.4.3	getStatus(std::vector< ApStatus > &status)=0	514
	5.20.4.4	getConnectedDevices(std::vector< DeviceInfo > &clientsInfo)=0	515
	5.20.4.5	getConnectedDevicesStats(std::vector< DeviceStats > &clientsStats)=0	515
	5.20.4.6	manageApService(Id apId, ServiceOperation opr)=0	516
	5.20.4.7	registerListener(std::weak_ptr< IApListener > listener)=0	516
	5.20.4.8	deregisterListener(std::weak_ptr< IApListener > listener)=0	516
	5.20.4.9	~IApInterfaceManager()	517
	5.20.4.10	onApDeviceStatusChanged(ApDeviceConnectionEvent event, std::vector< DeviceInfo > info)	517
	5.20.4.11	onApBandChanged(BandType radio)	517
	5.20.4.12	~IApListener()	517
	5.20.4.13	getServiceStatus()=0	517
	5.20.4.14	enable(bool enable)=0	518
	5.20.4.15	setMode(int numOfAp, int numOfSta)=0	518

5.20.4.16	getConfig(int &numAp, int &numSta)=0	519
5.20.4.17	getStatus(bool &isEnabled, std::vector< InterfaceStatus > &status)=0	519
5.20.4.18	registerListener(std::weak_ptr< IWlanListener > listener)=0	520
5.20.4.19	deregisterListener(std::weak_ptr< IWlanListener > listener)=0	520
5.20.4.20	~IWlanDeviceManager()	520
5.20.4.21	onServiceStatusChange(telux::common::ServiceStatus status)	520
5.20.4.22	onEnableChanged(bool enable)	520
5.20.4.23	~IWlanListener()	521
5.20.5	Variable Documentation	521
5.20.5.1	info	521
5.20.5.2	interworking	521
5.20.5.3	id	521
5.20.5.4	network	521
5.20.5.5	id	521
5.20.5.6	name	521
5.20.5.7	ipv4Address	521
5.20.5.8	ipv6Address	521
5.20.5.9	macAddress	521
5.20.5.10	id	521
5.20.5.11	macAddress	521
5.20.5.12	bytesTx	522
5.20.5.13	bytesRx	522
5.20.5.14	device	522
5.20.5.15	apStatus	522
5.20.5.16	staStatus	522
5.21	Wlan Device Management	523
5.22	Access Point Management	524
5.22.1	Data Structure Documentation	524
5.22.1.1	class telux::wlan::IApInterfaceManager	524
5.22.1.2	class telux::wlan::IApListener	524
5.23	Station Management	525
5.23.1	Data Structure Documentation	525
5.23.1.1	struct telux::wlan::StaStaticIpConfig	525
5.23.1.2	struct telux::wlan::StaConfig	525
5.23.2	Enumeration Type Documentation	525
5.23.2.1	StaIpConfig	525
5.23.2.2	StaBridgeMode	525
5.23.3	Function Documentation	526
5.23.3.1	setIpConfig(Id stald, StaIpConfig ipConfig, StaStaticIpConfig staticIpConfig)=0	526
5.23.3.2	setBridgeMode(Id stald, StaBridgeMode bridgeMode)=0	526
5.23.3.3	getConfig(std::vector< StaConfig > &config)=0	527
5.23.3.4	getStatus(std::vector< StaStatus > &status)=0	527
5.23.3.5	manageStaService(Id stald, ServiceOperation opr)=0	527
5.23.3.6	registerListener(std::weak_ptr< IStaListener > listener)=0	528
5.23.3.7	deregisterListener(std::weak_ptr< IStaListener > listener)=0	528
5.23.3.8	~IStaInterfaceManager()	528
5.23.3.9	onStationStatusChanged(std::vector< StaStatus > staStatus)	529
5.23.3.10	onStationBandChanged(BandType radio)	529



	5.23.3.11	~IStaListener()	529
5.23.4		Variable Documentation	529
	5.23.4.1	ipAddr	529
	5.23.4.2	gwIpAddr	529
	5.23.4.3	netMask	529
	5.23.4.4	dnsAddr	529
	5.23.4.5	stald	529
	5.23.4.6	ipConfig	529
	5.23.4.7	staticIpConfig	530
	5.23.4.8	bridgeMode	530
5.24		Telematics_cv2x_cpp	531
	5.24.1	Data Structure Documentation	531
		5.24.1.1 class telux::cv2x::ICv2xThrottleManagerListener	531
		5.24.1.2 class telux::cv2x::ICv2xThrottleManager	532
		5.24.1.3 class telux::cv2x::ICv2xTxStatusReportListener	533
5.25		Telematics_net	535
	5.25.1	Data Structure Documentation	535
		5.25.1.1 class telux::data::net::IBridgeManager	535
		5.25.1.2 class telux::data::net::IBridgeListener	538
		5.25.1.3 class telux::data::net::IFirewallManager	539
		5.25.1.4 class telux::data::net::IFirewallEntry	548
		5.25.1.5 class telux::data::net::IFirewallListener	549
		5.25.1.6 struct telux::data::net::L2tpSessionConfig	550
		5.25.1.7 struct telux::data::net::L2tpTunnelConfig	550
		5.25.1.8 struct telux::data::net::L2tpSysConfig	550
		5.25.1.9 class telux::data::net::IL2tpManager	551
		5.25.1.10 class telux::data::net::IL2tpListener	554
		5.25.1.11 struct telux::data::net::NatConfig	555
		5.25.1.12 class telux::data::net::INatManager	555
		5.25.1.13 class telux::data::net::INatListener	558
		5.25.1.14 class telux::data::net::ISocksManager	559
		5.25.1.15 class telux::data::net::ISocksListener	561
		5.25.1.16 class telux::data::net::IVlanManager	562
		5.25.1.17 class telux::data::net::IVlanListener	568
	5.25.2	Enumeration Type Documentation	569
		5.25.2.1 L2tpProtocol	569
5.26		Telematics_platform_deviceinfo	570
	5.26.1	Data Structure Documentation	570
		5.26.1.1 class telux::platform::IDeviceInfoListener	570
		5.26.1.2 struct telux::platform::PlatformVersion	570
		5.26.1.3 class telux::platform::IDeviceInfoManager	570
5.27		Telematics_platform	573
	5.27.1	Data Structure Documentation	573
		5.27.1.1 class telux::platform::PlatformFactory	573
5.28		Telematics_cellbroadcast	575
	5.28.1	Data Structure Documentation	575
		5.28.1.1 struct telux::tel::CellBroadcastFilter	575
	5.28.2	Enumeration Type Documentation	575
		5.28.2.1 GeographicalScope	575
		5.28.2.2 MessagePriority	575

5.28.2.3	MessageType	575
5.28.2.4	EtwsWarningType	576
5.28.2.5	CmasMessageClass	576
5.28.2.6	CmasSeverity	576
5.28.2.7	CmasUrgency	576
5.28.2.8	CmasCertainty	577
5.28.2.9	GeometryType	577
5.29	Telematics_ecall	578
5.29.1	Data Structure Documentation	578
5.29.1.1	class telux::tel::IEcallManager	578
5.29.1.2	class telux::tel::IEcallListener	580
5.30	Telematics_ims_settings	581
5.30.1	Data Structure Documentation	581
5.30.1.1	struct telux::tel::ImsServiceConfig	581
5.30.1.2	class telux::tel::IImSettingsManager	581
5.30.1.3	class telux::tel::IImSettingsListener	583
5.30.2	Enumeration Type Documentation	584
5.30.2.1	ImsServiceConfigType	584
5.31	Telematics_multi_sim	585
5.31.1	Data Structure Documentation	585
5.31.1.1	struct telux::tel::SlotStatus	585
5.31.1.2	class telux::tel::IMultiSimManager	585
5.31.1.3	class telux::tel::IMultiSimListener	589
5.31.2	Enumeration Type Documentation	590
5.31.2.1	SlotState	590
<b>6</b>	<b>Namespace Documentation</b>	<b>591</b>
6.1	telux Namespace Reference	591
6.2	telux::audio Namespace Reference	591
6.2.1	Variable Documentation	593
6.2.1.1	INFINITE_DTMF_DURATION	593
6.2.1.2	INFINITE_TONE_DURATION	593
6.3	telux::common Namespace Reference	593
6.3.1	Typedef Documentation	593
6.3.1.1	ResponseCallback	593
6.3.1.2	InitResponseCb	593
6.4	telux::config Namespace Reference	594
6.5	telux::cv2x Namespace Reference	594
6.5.1	Typedef Documentation	595
6.5.1.1	CreateRxSubscriptionCallback	595
6.5.1.2	CreateTxSpsFlowCallback	596
6.5.1.3	CreateTxEventFlowCallback	596
6.5.1.4	CloseTxFlowCallback	596
6.5.1.5	CloseRxSubscriptionCallback	597
6.5.1.6	ChangeSpsFlowInfoCallback	597
6.5.1.7	RequestSpsFlowInfoCallback	597
6.5.1.8	ChangeEventFlowInfoCallback	598
6.5.1.9	RequestCapabilitiesCallback	598
6.5.1.10	RequestDataSessionSettingsCallback	598
6.5.1.11	UpdateTrustedUEListCallback	598

	6.5.1.12	UpdateSrcL2InfoCallback	599
	6.5.1.13	CreateTcpSocketCallback	599
	6.5.1.14	CloseTcpSocketCallback	599
	6.5.1.15	StartCv2xCallback	600
	6.5.1.16	StopCv2xCallback	600
	6.5.1.17	RequestCv2xStatusCallback	600
	6.5.1.18	RequestCv2xStatusCallbackEx	600
	6.5.1.19	UpdateConfigurationCallback	601
6.6		telux::data Namespace Reference	601
	6.6.1	Data Structure Documentation	603
		6.6.1.1 struct telux::data::EsplInfo	603
		6.6.1.2 struct telux::data::IcmlInfo	604
		6.6.1.3 struct telux::data::IpFamilyInfo	604
		6.6.1.4 struct telux::data::IPv4Info	604
		6.6.1.5 struct telux::data::IPv6Info	604
		6.6.1.6 struct telux::data::QosFilterRule	605
		6.6.1.7 struct telux::data::TcpInfo	605
		6.6.1.8 struct telux::data::TftChangeInfo	605
		6.6.1.9 struct telux::data::TrafficFlowTemplate	606
		6.6.1.10 struct telux::data::UdpInfo	606
	6.6.2	Typedef Documentation	606
		6.6.2.1 DataCallResponseCb	606
		6.6.2.2 StatisticsResponseCb	607
		6.6.2.3 DataCallListResponseCb	607
		6.6.2.4 TrafficFlowTemplateCb	608
		6.6.2.5 DataRestrictModeCb	608
		6.6.2.6 TypeOfService	608
		6.6.2.7 TrafficClass	608
		6.6.2.8 FlowLabel	608
6.7		telux::data::net Namespace Reference	608
	6.7.1	Data Structure Documentation	610
		6.7.1.1 struct telux::data::net::BridgeInfo	610
		6.7.1.2 struct telux::data::net::DmzConfig	610
		6.7.1.3 struct telux::data::net::FirewallConfig	610
		6.7.1.4 struct telux::data::net::FirewallEntryInfo	610
		6.7.1.5 struct telux::data::net::VlanBindConfig	611
	6.7.2	Typedef Documentation	611
		6.7.2.1 BridgeInfoResponseCb	611
		6.7.2.2 FirewallConfigCb	611
		6.7.2.3 FirewallEntryInfoCb	611
		6.7.2.4 DmzEntryInfoCb	612
		6.7.2.5 FirewallStatusCb	612
		6.7.2.6 FirewallEntriesCb	612
		6.7.2.7 DmzEntriesCb	613
		6.7.2.8 L2tpConfigCb	613
		6.7.2.9 StaticNatEntriesCb	613
		6.7.2.10 CreateVlanCb	614
		6.7.2.11 QueryVlanResponseCb	614
		6.7.2.12 VlanMappingResponseCb	614
	6.7.3	Enumeration Type Documentation	615

6.7.3.1	BridgelFaceType	615
6.8	telux::loc Namespace Reference	615
6.9	telux::platform Namespace Reference	617
6.10	telux::power Namespace Reference	617
6.11	telux::tel Namespace Reference	617
6.11.1	Data Structure Documentation	623
6.11.1.1	struct telux::tel::ImsRegistrationInfo	623
6.11.1.2	struct telux::tel::NetworkTimelInfo	623
6.11.1.3	struct telux::tel::Point	623
6.11.2	Typedef Documentation	624
6.11.2.1	MakeCallCallback	624
6.11.2.2	ECallHlapTimerStatusCallback	624
6.11.2.3	PinOperationResponseCb	624
6.11.2.4	QueryFdnLockResponseCb	625
6.11.2.5	QueryPin1LockResponseCb	625
6.11.2.6	EidResponseCallback	625
6.11.2.7	RequestFiltersResponseCallback	625
6.11.2.8	RequestActivationStatusResponseCallback	626
6.11.2.9	ImsRegistrationInfoCb	626
6.11.2.10	ImsServiceConfigValidity	626
6.11.2.11	ImsServiceConfigCb	626
6.11.2.12	HighCapabilityCallback	627
6.11.2.13	SlotStatusCallback	627
6.11.2.14	RatMask	627
6.11.2.15	SelectionModeResponseCallback	627
6.11.2.16	PreferredNetworksCallback	628
6.11.2.17	NetworkScanCallback	628
6.11.2.18	VoiceRadioTechResponseCb	628
6.11.2.19	CellInfoCallback	629
6.11.2.20	ECallGetOperatingModeCallback	629
6.11.2.21	RATCapabilitiesMask	629
6.11.2.22	VoiceServiceTechnologiesMask	629
6.11.2.23	SapStateResponseCallback	629
6.11.2.24	RatPreference	629
6.11.2.25	RatPreferenceCallback	630
6.11.2.26	ServiceDomainPreferenceCallback	630
6.11.2.27	ProfileListResponseCb	630
6.11.2.28	EidResponseCb	630
6.11.3	Enumeration Type Documentation	631
6.11.3.1	RegistrationStatus	631
6.12	telux::therm Namespace Reference	631
6.13	telux::wlan Namespace Reference	632
<b>7</b>	<b>Data Structure Documentation</b>	<b>633</b>
7.1	telux::tel::CellBroadcastMessage Class Reference	633
7.1.1	Constructors and Destructors	633
7.1.1.1	CellBroadcastMessage(std::shared_ptr< EtwsInfo > etwsInfo)	633
7.1.1.2	CellBroadcastMessage(std::shared_ptr< CmasInfo > cmasInfo)	633
7.1.2	Member Function Documentation	634
7.1.2.1	getMessageType() const	634

	7.1.2.2	getEtwsInfo() const	634
	7.1.2.3	getCmasInfo() const	634
7.2		telux::tel::Circle Class Reference	634
7.2.1		Constructors and Destructors	635
	7.2.1.1	Circle(Point center, double radius)	635
7.2.2		Member Function Documentation	635
	7.2.2.1	getCenter()	635
	7.2.2.2	getRadius()	635
7.3		telux::tel::CmasInfo Class Reference	635
7.3.1		Constructors and Destructors	636
	7.3.1.1	CmasInfo(GeographicalScope geographicalScope, int msgId, int serialNumber, std::string languageCode, std::string message↵ Text, MessagePriority priority, CmasMessageClass message↵ Class, CmasSeverity severity, CmasUrgency urgency, Cmas↵ Certainty certainty, std::shared_ptr< WarningAreaInfo > warning↵ AreaInfo)	636
7.3.2		Member Function Documentation	636
	7.3.2.1	getGeographicalScope() const	637
	7.3.2.2	getMessageId() const	637
	7.3.2.3	getSerialNumber() const	637
	7.3.2.4	getLanguageCode() const	637
	7.3.2.5	getMessageBody() const	638
	7.3.2.6	getPriority() const	638
	7.3.2.7	getMessageCode() const	638
	7.3.2.8	getUpdateNumber() const	639
	7.3.2.9	getMessageClass()	639
	7.3.2.10	getSeverity()	639
	7.3.2.11	getUrgency()	639
	7.3.2.12	getCertainty()	640
	7.3.2.13	getWarningAreaInfo()	640
7.4		telux::tel::EtwsInfo Class Reference	640
7.4.1		Constructors and Destructors	641
	7.4.1.1	EtwsInfo(GeographicalScope geographicalScope, int msgId, int serialNumber, std::string languageCode, std::string message↵ Text, MessagePriority priority, EtwsWarningType warningType, bool emergencyUserAlert, bool activatePopup, bool primary, std↵ ::vector< uint8_t > warningSecurityInformation)	641
7.4.2		Member Function Documentation	641
	7.4.2.1	getGeographicalScope() const	641
	7.4.2.2	getMessageId() const	642
	7.4.2.3	getSerialNumber() const	642
	7.4.2.4	getLanguageCode() const	642
	7.4.2.5	getMessageBody() const	643
	7.4.2.6	getPriority() const	643
	7.4.2.7	getMessageCode() const	643
	7.4.2.8	getUpdateNumber() const	643
	7.4.2.9	getEtwsWarningType()	644
	7.4.2.10	isEmergencyUserAlert()	644
	7.4.2.11	isPopupAlert()	644
	7.4.2.12	isPrimary()	644

	7.4.2.13	getWarningSecurityInformation()	645
7.5		telux::tel::Geometry Class Reference	645
	7.5.1	Constructors and Destructors	645
		7.5.1.1 Geometry(std::shared_ptr< Polygon > polygon)	645
		7.5.1.2 Geometry(std::shared_ptr< Circle > circle)	645
	7.5.2	Member Function Documentation	646
		7.5.2.1 getType() const	646
		7.5.2.2 getPolygon() const	646
		7.5.2.3 getCircle() const	646
7.6		telux::tel::ICellBroadcastListener Class Reference	646
	7.6.1	Constructors and Destructors	647
		7.6.1.1 ~ICellBroadcastListener()	647
	7.6.2	Member Function Documentation	647
		7.6.2.1 onIncomingMessage(SlotId slotId, const std::shared_ptr< CellBroadcast← Message > cbMessage)	647
		7.6.2.2 onMessageFilterChange(SlotId slotId, std::vector< CellBroadcast← Filter > filters)	647
7.7		telux::tel::ICellBroadcastManager Class Reference	648
	7.7.1	Constructors and Destructors	648
		7.7.1.1 ~ICellBroadcastManager()	648
	7.7.2	Member Function Documentation	648
		7.7.2.1 isSubsystemReady()=0	648
		7.7.2.2 onSubsystemReady()=0	649
		7.7.2.3 getSlotId()=0	649
		7.7.2.4 updateMessageFilters(std::vector< CellBroadcastFilter > filters, telux::common::ResponseCallback callback=nullptr)=0	649
		7.7.2.5 requestMessageFilters(RequestFiltersResponseCallback callback)=0650	
		7.7.2.6 setActivationStatus(bool activate, telux::common::Response← Callback callback=nullptr)=0	650
		7.7.2.7 requestActivationStatus(RequestActivationStatusResponseCallback callback)=0	650
		7.7.2.8 registerListener(std::weak_ptr< ICellBroadcastListener > listener)=0651	
		7.7.2.9 deregisterListener(std::weak_ptr< ICellBroadcastListener > lis- tener)=0	651
7.8		telux::cv2x::ICv2xListener Class Reference	652
	7.8.1	Constructors and Destructors	652
		7.8.1.1 ~ICv2xListener()	652
	7.8.2	Member Function Documentation	652
		7.8.2.1 onStatusChanged(Cv2xStatus status)	652
		7.8.2.2 onStatusChanged(Cv2xStatusEx status)	652
7.9		telux::data::IEspFilter Class Reference	652
	7.9.1	Constructors and Destructors	653
		7.9.1.1 ~IEspFilter()	653
	7.9.2	Member Function Documentation	653
		7.9.2.1 getEspInfo()=0	653
		7.9.2.2 setEspInfo(const EspInfo &espInfo)=0	653
7.10		telux::data::IlcmpFilter Class Reference	654
	7.10.1	Constructors and Destructors	654
		7.10.1.1 ~IlcmpFilter()	654
	7.10.2	Member Function Documentation	654

	7.10.2.1	getIcmpInfo()=0	654
	7.10.2.2	setIcmpInfo(const IcmpInfo &icmpInfo)=0	654
7.11		telux::tel::IImServingSystemListener Class Reference	655
	7.11.1	Constructors and Destructors	655
	7.11.1.1	~IImServingSystemListener()	655
	7.11.2	Member Function Documentation	655
	7.11.2.1	onServiceStatusChange(telux::common::ServiceStatus status)	655
	7.11.2.2	onImSRegStatusChange(ImSRegistrationInfo status)	656
7.12		telux::tel::IImServingSystemManager Class Reference	656
	7.12.1	Constructors and Destructors	656
	7.12.1.1	~IImServingSystemManager()	656
	7.12.2	Member Function Documentation	656
	7.12.2.1	getServiceStatus()=0	656
	7.12.2.2	requestRegistrationInfo(ImSRegistrationInfoCb callback)=0	657
	7.12.2.3	registerListener(std::weak_ptr< telux::tel::IImServingSystem← Listener > listener)=0	657
	7.12.2.4	deregisterListener(std::weak_ptr< telux::tel::IImServingSystem← Listener > listener)=0	658
7.13		telux::data::IPFilter Class Reference	658
	7.13.1	Constructors and Destructors	658
	7.13.1.1	~IPFilter()	658
	7.13.2	Member Function Documentation	659
	7.13.2.1	getIPv4Info()=0	659
	7.13.2.2	setIPv4Info(const IPv4Info &ipv4Info)=0	659
	7.13.2.3	getIPv6Info()=0	659
	7.13.2.4	setIPv6Info(const IPv6Info &ipv6Info)=0	659
	7.13.2.5	getIpProtocol()=0	660
	7.13.2.6	getIpFamily()=0	660
7.14		telux::common::IServiceStatusListener Class Reference	660
	7.14.1	Constructors and Destructors	661
	7.14.1.1	~IServiceStatusListener()	661
	7.14.2	Member Function Documentation	661
	7.14.2.1	onServiceStatusChange(ServiceStatus status)	661
7.15		telux::data::ITcpFilter Class Reference	661
	7.15.1	Constructors and Destructors	661
	7.15.1.1	~ITcpFilter()	661
	7.15.2	Member Function Documentation	661
	7.15.2.1	getTcpInfo()=0	661
	7.15.2.2	setTcpInfo(const TcpInfo &tcpInfo)=0	662
7.16		telux::data::IUDPFilter Class Reference	662
	7.16.1	Constructors and Destructors	662
	7.16.1.1	~IUDPFilter()	662
	7.16.2	Member Function Documentation	662
	7.16.2.1	getUdpInfo()=0	662
	7.16.2.2	setUdpInfo(const UdpInfo &udpInfo)=0	663
7.17		telux::tel::Polygon Class Reference	663
	7.17.1	Constructors and Destructors	663
	7.17.1.1	Polygon(std::vector< Point > vertices)	663
	7.17.2	Member Function Documentation	663
	7.17.2.1	getVertices()	664

7.18	telux::tel::WarningAreaInfo Class Reference	664
7.18.1	Constructors and Destructors	664
7.18.1.1	WarningAreaInfo(int maxWaitTime, std::vector< Geometry > geometries)	664
7.18.2	Member Function Documentation	664
7.18.2.1	getGeoFenceMaxWaitTime()	664
7.18.2.2	getGeometries()	665



## List of Figures

2-1	SDK-Overview	7
3-1	Phone manager initialization	18
3-2	Dial call flow	19
3-3	ECall call flow	20
3-4	Signal strength call flow	22
3-5	Answer, Reject, RejectWithSMS call flow	23
3-6	Hold call flow	24
3-7	Hold, Conference, Swap call flow	25
3-8	SMS call flow	27
3-9	Radio and Service state call flow	28
3-10	Network selection manager call flow	29
3-11	Serving System Manager Call Flow	31
3-12	Get applications call flow	33
3-13	On logical channel	34
3-14	On basic channel	35
3-15	Request card reader status, Request ATR, Transmit APDU call flow	36
3-16	SIM Turn off, Turn on and Reset call flow	38
3-17	Subscription initialization call flow	40
3-18	Subscription call flow	41
3-19	Call flow to register/remove listener for generating basic reports	42
3-20	Call flow to register/remove listener for generating detailed reports	44
3-21	Call flow to register/remove listener for generating detailed engine reports	46
3-22	Call flow to register/remove listener for system info updates	47
3-23	Call flow to request energy consumed information	48
3-24	Call flow to get terrestrial positioning information	49
3-25	Call flow to cancel terrestrial positioning information	49
3-26	Call flow to enable/disable constraint time uncertainty	50
3-27	Call flow to enable/disable PACE	50
3-28	Call flow to delete all aiding data	51
3-29	Call flow to configure lever arm parameters	51
3-30	Call flow to configure blacklisted constellations	52
3-31	Call flow to configure robust location	52
3-32	Call flow to configure min gps week	53
3-33	Call flow to request min gps week	53
3-34	Call flow to delete specified data	54
3-35	Call flow to configure min sv elevation	54
3-36	Call flow to request min sv elevation	55
3-37	Call flow to request robust location	55
3-38	Call flow to configure dead reckoning engine	56
3-39	Call flow to provide user consent for terrestrial positioning	56
3-40	Call flow to represent Xtra Feature	57
3-41	Start/Stop for data connection manager call flow	58
3-42	Request/Create/Delete/Modify for data profile manager call flow	60
3-43	Get/Set data filter mode call flow	62
3-44	Add data restrict filter call flow	63
3-45	Remove data restrict filter call flow	64

3-46 Create VLAN and bind it to PDN for data VLAN manager call flow . . . . .	66
3-47 LAN-LAN VLAN Configuration Usecase from EAP call flow . . . . .	67
3-48 LAN-LAN VLAN Configuration Usecase from A7 call flow . . . . .	68
3-49 LAN-WAN VLAN Configuration Usecase from EAP call flow . . . . .	69
3-50 LAN-WAN VLAN Configuration Usecase from A7 call flow . . . . .	71
3-51 Create Static NAT entry for data NAT manager call flow . . . . .	72
3-52 Firewall enablement in data Firewall manager call flow . . . . .	73
3-53 Add Firewall entry in data Firewall manager call flow . . . . .	74
3-54 Set Firewall DMZ in data Firewall manager call flow . . . . .	75
3-55 Socks enablement in data Socks manager call flow . . . . .	76
3-56 Call flow to add and enable a software bridge . . . . .	77
3-57 Call flow to remove and disable a software bridge . . . . .	78
3-58 Start/Stop C-V2X Mode call flow . . . . .	79
3-59 C-V2X Radio Manager call flow . . . . .	80
3-60 C-V2X Radio Initialization call flow . . . . .	81
3-61 C-V2X Radio RX subscription call flow . . . . .	82
3-62 C-V2X Radio TX event-driven flow call flow . . . . .	83
3-63 C-V2X Radio SPS flow call flow . . . . .	84
3-64 Audio Manager API call flow . . . . .	86
3-65 Audio Voice Call Start/Stop call flow . . . . .	88
3-66 Audio Voice Call Device Switch call flow . . . . .	90
3-67 Audio Voice Call Volume/Mute control call flow . . . . .	92
3-68 Call flow to play DTMF tone . . . . .	94
3-69 Call flow to detect DTMF tone . . . . .	96
3-70 Audio Playback call flow . . . . .	98
3-71 Audio Capture call flow . . . . .	100
3-72 Call flow to play/stop tone on a sink device . . . . .	102
3-73 Call flow to start/stop loopback between source and sink devices . . . . .	103
3-74 Call flow to play Compressed audio format . . . . .	105
3-75 Audio Transcoding Operation Callflow . . . . .	107
3-76 Compressed audio format playback on Voice Paths Callflow . . . . .	109
3-77 Thermal manager call flow . . . . .	111
3-78 Call flow to register/remove listener for Thermal shutdown manager . . . . .	112
3-79 Call flow to set/get the Thermal auto-shutdown mode . . . . .	113
3-80 Call flow to manage thermal auto-shutdown from an eCall application . . . . .	114
3-81 Call flow to register/remove listener for TCU-activity manager . . . . .	116
3-82 Call flow to set the TCU-activity state . . . . .	117
3-83 Remote SIM call flow . . . . .	118
3-84 Modem Config load and activate call flow . . . . .	120
3-85 Modem Config deactivate and delete Call Flow . . . . .	121
3-86 Modem Config get and set Auto Selection Mode Call Flow . . . . .	122
3-87 Modify WLAN Configuration Call Flow . . . . .	124
3-88 Modify WLAN Station Configuration Call Flow . . . . .	126
3-89 Modify WLAN Access Point Configuration Call Flow . . . . .	127

# 1 Introduction

---

## 1.1 Purpose

This document describes interface specification of Telematics Software Development Kit (SDK) for applications on Linux based automotive platforms.

## 1.2 Scope

Telematics SDK exposes a rich set of APIs conforming to C++11 standard which form the building blocks to create stand-alone Telematics applications.

This document provides high level overview of Telematics SDK architecture and its APIs. Call flow diagrams are provided for various APIs such as a Phone, Call, SMS, SIM Card Services, Audio, Location, Remote Sim, Modem configuration etc.

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

## 1.3 Conventions

Function declarations, function names, type declarations, and code samples appear in a different font. For example,

```
#include
```

Parameter directions are indicated as follows:

### Parameters

in	<i>paramname</i>	indicates an input parameter.
out	<i>paramname</i>	indicates an output parameter.
in, out	<i>paramname</i>	indicates a parameter used for both input and output.

Most APIs are asynchronous as underlying sub-systems such as telephony are asynchronous. API names follow the convention of using prefix " get " for synchronous calls and " request " for asynchronous calls. Asynchronous responses such as listener callbacks come on a different thread than the application thread.

## 1.4 SDK Versioning

The following convention is used for versioning the SDK releases

**SDK version (major.minor.patch)**

SDK\_VERSION = 1.0.0

**Major version:** This number will be incremented whenever significant changes or features are introduced.

**Minor version:** This number will be incremented when smaller features with some new APIs are introduced.

**Patch version:** If the release only contains bug fixes, but no API change then the patch version would be incremented, No change in the actual API interface.

**NOTE:** Use `telux::common::Version::getSdkVersion()` API to query the current version of SDK or refer the VERSION file in the repository

## 1.5 Public API Status

Public APIs are introduced and removed (if necessary) in phases. This allows users of an existing API that is being Deprecated to migrate. APIs will be marked with note indicating whether the API is subject to change or if it is not recommended to use the API.

as follows:

- **Eval:** This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.
- **Obsolete:** API is not preferred and a better alternative is available.
- **Deprecated:** API is not going to be supported in future releases. Clients should stop using this API. Once an API has been marked as Deprecated, the API could be removed in future releases.

## 2 Functional Overview

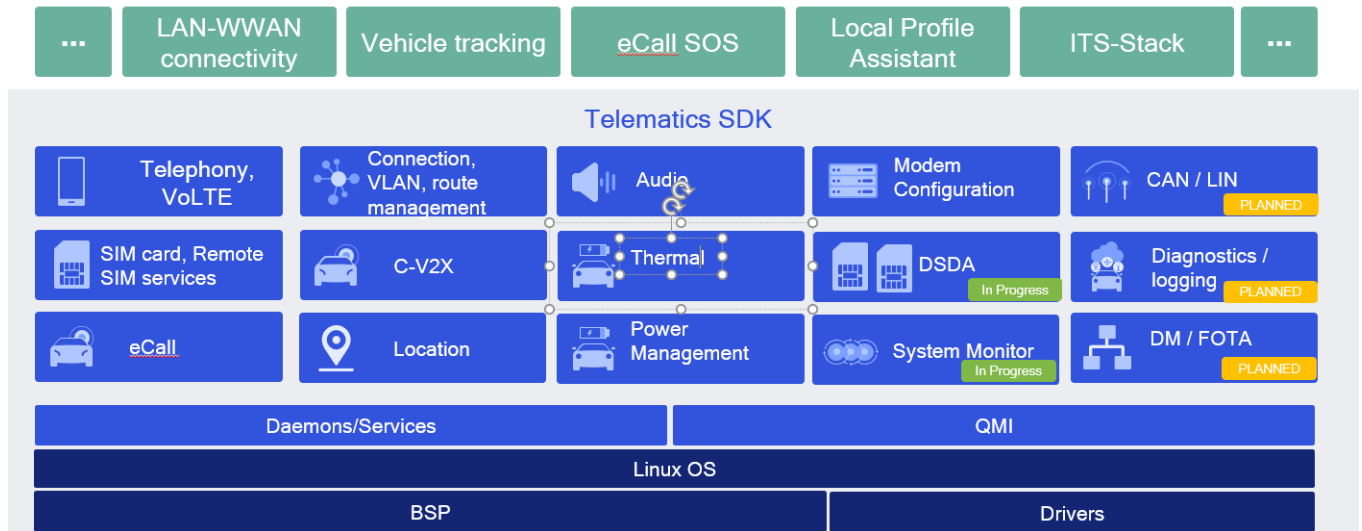


Figure 2-1 SDK-Overview

### 2.1 Overview

The Telematics library runs in the user space of the Linux system. It interacts with Telephony services and other sub-systems to provide various services like phone calls, SMS etc. These services are exposed by the SDK through fixed public APIs that are available on all Telematics platforms that support SDK. The Telematics APIs are grouped into the following functional modules:

#### Telephony

Telephony sub-system consists of APIs for functions related to Phone, Call, SMS and Signal Strength, Network Selection and Serving System Management.

#### SIM Card Services

SIM Card services sub-system consists of APIs to perform SIM card operations such as Send APDU messages to SIM card applications, SIM Access Profile(SAP) operations etc.

#### Location Services

Location Services sub-system consists of APIs to receive location details such as GNSS Positions, Satellite Vehicle information, Jammer Data signals, nmea and measurements information. The location manager sub-system also consists of APIs to get location system info, request energy consumed, get year of hardware information, get terrestrial position information and cancel terrestrial position information. LocationConfigurator allows general engine configurations (example: TUNC, PACE etc),configuration of

specific engines like SPE (example: minSVElevation, minGPSWeek etc) or DRE, deletion of warm and cold aiding data, NMEA configuration and support for XTRA feature.

### **Connection Management**

Connection Management sub-system consists of APIs for establishing Cellular WAN/ Backhaul connection sessions and for Connection Profile Management etc.

### **Audio Management**

Audio Management sub-system consists of APIs for Audio management such as setting up audio streams, switching devices on streams, apply volume/mute etc

### **Thermal Management**

Thermal Management sub-system consists of APIs to get list of thermal zones, cooling devices and binding information.

### **Thermal Shutdown Management**

Thermal shutdown management sub-system consists of APIs to get/set the thermal auto-shutdown mode and listen to its updates.

### **TCU Activity Management**

TCU Activity Management sub-system consists of APIs to get TCU-activity state updates, set the TCU-activity state, etc.

### **Remote SIM Services**

Remote SIM sub-system consists of APIs that allow a device that does not have a SIM physically connected to it to access a SIM remotely (e.g. over BT, Ethernet, etc.) and perform card operations on that SIM, such as requesting reset, transmitting APDU messages, etc.

### **Modem Config Services**

Modem Config sub-system consists of APIs that allow to request modem config files, load/delete a modem config file from modem's storage, activate/deactivate a modem config file, get/set auto selection mode for config files.

### **Data Network Management**

Data Network Management sub-system consists of APIs to setup VLAN, static NAT, Firewall, Socks, etc.

### **Wlan Management**

Wlan Management sub-system consists of APIs to configure, enable, set access point, station configurations, etc.

Telematics SDK classes can be broadly divided into the following types:

- Factory - Factory classes are central classes such as PhoneFactory which can be used to create Manager classes corresponding to their sub-systems such as PhoneManager.
- Manager - Manager classes such as PhoneManager to manage multiple Phone instances, CardManager to manage multiple SIM Card instances etc.
- Observer/ Listener - Listener for unsolicited responses.
- Command Callback - Single-shot response callback for asynchronous API requests.

- Logger - APIs to log messages, control the log levels.

## 2.2 Features

Telematics SDK provides APIs for the following features:

### 2.2.1 Call Management

CallManager, Phone and PhoneManager APIs of Telematics SDK provides call related control operations such as

- Initiate a voice call
- Answer the incoming call
- Hold the call
- Hangup waiting, held or active call

CallManager and PhoneManager also provides additional functionality such as

- Allowing conference, and switch between waiting or holding call and active call
- Emergency Call (dial 112)
- Notifications about call state change

### 2.2.2 SMS

SMS Manager APIs of Telematics SDK provides SMS related functionality such as

- Sends and receives SMS messages of type GSM7, GSM8 and UCS2

### 2.2.3 SIM Card Services

The SIM Card operations are performed by CardManager and SapCardManager.

CardManager APIs of Telematics SDK perform operations on UICC card such as

- Open or close logical/basic channel to ICC card
- Transmit Application Protocol Data Unit (APDU) to the ICC Card over logical/basic channel
- Receive response APDU from the ICC Card with the status
- Notify about ICC card information change

SapCardManager APIs provides SIM Access Profile(SAP) related functionality such as

- Open or close SIM Access Profile(SAP) connection
- Transmit Application Protocol Data Unit (APDU) over SAP connection
- Receive response APDU over SAP connection
- Perform SAP operations such as Answer to Reset(ATR), SIM Power off, SIM Power On, SIM Reset and fetch Card Reader status.

## 2.2.4 Phone Information

Phone APIs of Telematics SDK provides phone related information such as

- Get Service state of phone i.e. EMERGENCY\_ONLY, IN\_SERVICE and OUT\_OF\_SERVICE
- Get Radio state of device i.e RADIO\_STATE\_OFF, RADIO\_STATE\_ON and RADIO\_STATE\_UNAVAILABLE
- Retrieve the signal strength corresponding to the technology supported by SIM
- Device Identity
- Set or Request Operating Mode
- Subscription Information

## 2.2.5 Location Services

Location Services APIs of Telematics SDK provide the mechanism to register listener and to receive location updates, satellite vehicle information, jammer signals, nmea and measurement updates. The location manager sub-system also consists of APIs to get terrestrial position information and cancel terrestrial position information. Following parameters are configurable through the APIs.

- Minimum time interval between two consecutive location reports.
- Minimum distance travelled after which the period between two consecutive location reports depends on the interval specified.

LocationConfigurator allows general engine configurations (example: TUNC, PACE etc), configuration of specific engines like SPE (example: minSVElevation, minGPSWeek etc) or DRE, deletion of warm and cold aiding data, NMEA configuration and support for XTRA feature.

## 2.2.6 Data Services

Data Services APIs in the Telematics SDK used for cellular connectivity, modem profile management, filters management, and networking.

Data Connection Manager APIs provide functionality such as

- start / stop data call
- listen for data call state changes

Data Profile Manager APIs provide functionality such as

- List available profiles in the modem
- Create / modify / delete / modify modem profiles
- Query for the selected profile

Data Filter Manager APIs provide functionality such as

- get / set data filter mode per data call
- get / set data filter mode for all data call in up state
- add / remove data filter per data call



- add / remove data filter for all data call in up state

Data VLAN Manager APIs provide functionality such as

- Create / remove VLAN
- Query VLAN info
- Bind / unbind VLAN to PDN
- Query current VLAN to PDN mapping

Data Static LAN Manager APIs provide functionality such as

- Add / remove static LAN entry
- Request current static NAT entries

Data Firewall Manager APIs provide functionality such as

- Add / remove DMZ entry
- Query current DMZ entries
- Set Firewall configuration to enable / disable Firewall
- Query current status of Firewall
- Add / remove Firewall entry
- Query Firewall entry rules

Data Software Bridge Manager provides interface to enable packet acceleration for non-standard WLAN and Ethernet physical interfaces. It facilitates to configure a software bridge between the interface and Hardware accelerator. Its APIs provide functionality such as

- Add / remove a software bridge
- Query the software bridges configured in the system
- Enable / Disable the software bridge management

## 2.2.7 Network Selection and Serving System Management

Network Selection and Service System Management APIs in the Telematics SDK used for configuring the networks and preferences

Network Selection Manager APIs provide functionality such as

- request or set network selection mode (Manual or Automatic)
- scan for available networks
- request or set preferred networks list

Serving System Manager APIs provide functionality such as

- request and set service domain preference and radio access technology mode preference for searching and registering (CS/PS domain, RAT and operation mode)

## 2.2.8 C-V2X

The C-V2X sub-system contains APIs that support Cellular-V2X operation.

Cellular-V2X APIs in the Telematics SDK include Cv2xRadioManager and Cv2xRadio interfaces.

Cv2xRadioManager provides an interface to a C-V2X capable modem. The API includes methods for

- Enabling C-V2X mode
- Disabling C-V2X mode
- Querying the status of C-V2X
- Updating the C-V2X configuration via a config XML file

Cv2xRadio abstracts a C-V2X radio channel. The API includes methods for

- Obtaining the current capabilities of the radio
- Listen for radio state changes
- Creating and Closing an RX subscription
- Creating and Closing a TX event-driven flow
- Creating and Closing a TX semi-persistent-scheduling (SPS) flow
- Updating TX SPS flow parameters
- Update Source L2 Info

## 2.2.9 Audio

The Audio subSystem contains of APIs that support Audio operation.

Audio APIs in the Telematics SDK include AudioManager, AudioStream, AudioVoiceStream, AudioPlayStream, AudioCaptureStream interfaces.

AudioManager provides an interface for creation/deletion of audio stream. The API includes methods for

- Query readiness of subSystem
- Query supported "Device Types"
- Query supported "Stream Types"
- Creating Audio Stream
- Deleting Audio Stream

AudioStream abstracts the properties common to different stream types. The API includes methods for

- Query stream type
- Query routed device
- Set device
- Query volume details
- Set volume
- Query mute details

- Set mute

AudioVoiceStream along with inheriting AudioStream, provides additional APIs to manage voice call session as stated below.

- Start Voice Audio Operation
- Stop Voice Audio Operation
- Play DTMF tone
- Detect DTMF tone

AudioPlayStream along with inheriting AudioStream, provides additional APIs to manage audio play session as stated below.

- write audio samples
- Write audio samples for compressed audio format
- Stop Audio for compressed audio format
- Play compressed format audio on voice paths

AudioCaptureStream along with inheriting AudioStream, provides additional APIs to manage audio capture session as stated below.

- read audio samples

AudioLoopbackStream along with inheriting AudioStream, provides additional APIs to manage audio loopback session as stated below.

- Start loopback
- Stop loopback

AudioToneGeneratorStream along with inheriting AudioStream, provides additional APIs to manage audio tone generator session as stated below.

- Play tone
- Stop tone

Transcoder provides APIs to manage audio transcoder which is able to perform below operations.

- Convert one audio format to another

Audio SDK provides details of Supported "Device Types" and "Stream Types" in the Audio subSystem of Reference Telematics platform.

“Device Type” encapsulates the type of device supported in Reference Telematics platform. The representation of these devices would be made available via public header file <usr/include/telux/audio/AudioDefines.hpp>.

Example: DEVICE\_TYPE\_XXXX

Internally SDK Device Type mapped to Audio HAL devices as per <usr/include/system/audio.h>.

In current release it is mapped per below table.

### **Current Device Mapping Table:**

SDK Audio Device Representation	Audio HAL Representation
DEVICE_TYPE_SPEAKER	AUDIO_DEVICE_OUT_SPEAKER
DEVICE_TYPE_MIC	AUDIO_DEVICE_IN_BACK_MIC

However Device Mapping configurable as stated below. This configurability provides flexibility to map different Audio HAL devices to SDK representation.

Update tel.conf file with below details before boot of system.

NUM\_DEVICES specifies the number of device types supported

DEVICE\_TYPE specifies the SDK type of each device (in comma separated values)

DEVICE\_DIR specifies the device direction for each device in order above (in comma separated values)

AHAL\_DEVICE\_TYPE specifies the mapped Audio HAL type of each device (in comma separated values)

Example:

Note: The default values provided here are based on QTI's reference design.

NUM\_DEVICES=6

DEVICE\_TYPE=1,2,3,257,258,259

DEVICE\_DIR=1,1,1,2,2,2

AHAL\_DEVICE\_TYPE=2,1,4,2147483776,2147483652,2147483664

For any stream types, maximum device supported would be one. Single stream per multiple devices not supported. For voice stream Rx Device would decide corresponding Tx Device pair as decided by Audio HAL.

#### NOTE FOR SYSTEM INTEGRATORS:

The mapping of Audio devices to Audio HAL devices is static currently based on QTI's Reference Telematics platform. For custom platforms this mapping need to be updated.

“Stream Type” encapsulates the type of stream supported in Reference Telematics Platform. The representation of these stream types made available via public header file <usr/include/telux/audio/AudioDefines.hpp>.

Example: VOICE\_CALL, PLAY, CAPTURE etc

#### Volume Support Table:

This table captures scenarios where the volume could be modified.

Stream Type	Stream Direction (RX)	Stream Direction (Tx)
VOICE_CALL	Applicable	Not Applicable
PLAY	Applicable	Not Applicable
CAPTURE	Not Applicable	Applicable
LOOPBACK	Not Applicable	Not Applicable
TONE_GENERATOR	Not Applicable	Not Applicable

In case QTI's reference design does not support volume for specific stream category, API responds with error.

**Mute Support Table:**

This table captures when stream could be muted and in which direction.

Stream Type	Stream Direction (RX)	Stream Direction (Tx)
VOICE_CALL	Applicable	Applicable
PLAY	Applicable	Not Applicable
CAPTURE	Not Applicable	Applicable
LOOPBACK	Not Applicable	Not Applicable
TONE_GENERATOR	Not Applicable	Not Applicable

In case QTI's reference design does not support mute for specific stream category, API responds with error.

**2.2.10 Thermal Management**

Thermal Management APIs in the Telematics SDK are used for reading thermal zone, cooling device and binding information.

Thermal Management APIs provide functionality such as

- get thermal zones with thermal zone description, current temperature, trip points and binding info
- get cooling devices with cooling device type, maximum and current cooling level
- get thermal zone by Id
- get cooling device by Id

**2.2.11 Thermal Shutdown Management**

Thermal Shutdown Management APIs provide functionality such as

- Query auto-shutdown mode.
- Set auto-shutdown mode.
- Get notifications on auto-shutdown mode updates.

**2.2.12 TCU Activity Management**

TCU-activity Manager APIs in Telematics SDK provides TCU-activity state related operations such as

- Query the current TCU-activity state
- Get notifications about the TCU-activity state changes
- Set the system to a desired activity state

**2.2.13 Remote SIM**

Remote SIM APIs in the Telematics SDK allow a device to use the WWAN capabilities of a SIM on another device.

Remote SIM APIs provide functionality such as

- Sending card events (reset, power up, errors) to the modem
- Sending/receiving APDU messages from/to the modem and remote SIM.
- Receiving operations from the modem (disconnect, power up, reset) to the remote SIM.

### 2.2.14 Modem Config Management

Modem Config APIs in the Telematics SDK provides modem config related functionalities such as

- Request modem config files from modem's storage.
- Load a modem config file to modem's storage.
- Activate/Deactivate a modem config file from modem's storage.
- Get Active config info details.
- Get/Set config auto selection mode.
- Delete a modem config file from modem's storage.
- Ability to get notified whenever a SW config file is activated.

### 2.2.15 Wlan Management

Wlan managemnt APIs in the Telematics SDK provides services related to WiFi functionalities such as

- Enable/Disable Wlan.
- Set/Request Wlan mode: number of access points and number of stations to be enabled.
- Request current Wlan status.
- Set/Request access points configuration.
- Request access point status
- Set/Request station configuration.
- Request station status
- Request list of devices connected to any access point.
- Restart hostapd and wpa\_supplicant damemons

## 2.3 Security

## 2.3.1 SELinux

SELinux is an access control framework provided by the Linux kernel. It provides a mechanism to restrict/control access to system resources such as file nodes and sockets. SELinux framework expects any process running in userspace to declare all its interactions with the system resources in the form of SELinux policies. On platforms enabled with SELinux, an app that uses an SDK API would also need to declare its usage through SELinux policies to ensure that it has all the required permissions. In order for an application to use the SDK's APIs on such a platform, SELinux interfaces have been defined that the app needs to declare in its policies.

Listed below are the interfaces to be used when an app uses APIs belonging to a given namespace/class of the SDK.

**Note** For the below list, let us consider the application's security context to be `app_t` (also called domain context).

SDK Namespace/Class	SELinux interface	Arguments	Usage
tel	<code>telux_allow_tel()</code>	domain context	<code>telux_allow_tel(app_t)</code>
data	<code>telux_allow_data()</code>	domain context	<code>telux_allow_data(app_t)</code>
audio	<code>telux_allow_audio()</code>	domain context	<code>telux_allow_audio(app_t)</code>
loc	<code>telux_allow_loc()</code>	domain context	<code>telux_allow_loc(app_t)</code>
IThermalManager	<code>telux_allow_↔ thermalmanager()</code>	domain context	<code>telux_allow_↔ thermalmanager(app_t)</code>
IThermalShutdown↔ Manager	<code>telux_allow_↔ thermalshutdown()</code>	domain context	<code>telux_allow_↔ thermalshutdown(app_t)</code>
power	<code>telux_allow_power()</code>	domain context	<code>telux_allow_power(app↔ _t)</code>
config	<code>telux_allow_↔ modemconfig()</code>	domain context	<code>telux_allow_↔ modemconfig(app_t)</code>

The following example illustrates how an application can incorporate the SELinux interfaces exposed by SDK in its SELinux policies. Below code snippet is part of a Type Enforcement (TE) file of the application which grants required permissions to perform all SDK data operations.

```
policy_module(application, 1.0)
type app_t;

#Granting SDK data client permissions to the application

telux_allow_data(app_t);
```

# 3 Call Flow Diagrams

## 3.1 Application initialization call flow

TelSDK initializes various sub-systems during start-up. It marks each sub-system as ready once the initialization procedures are completed for that sub-system. The application has to wait until the corresponding sub-system is ready on which it needs to make API requests. TelSDK provides APIs to check whether sub-system is ready or not.

Example:

### 3.1.1 Phone manager initialization

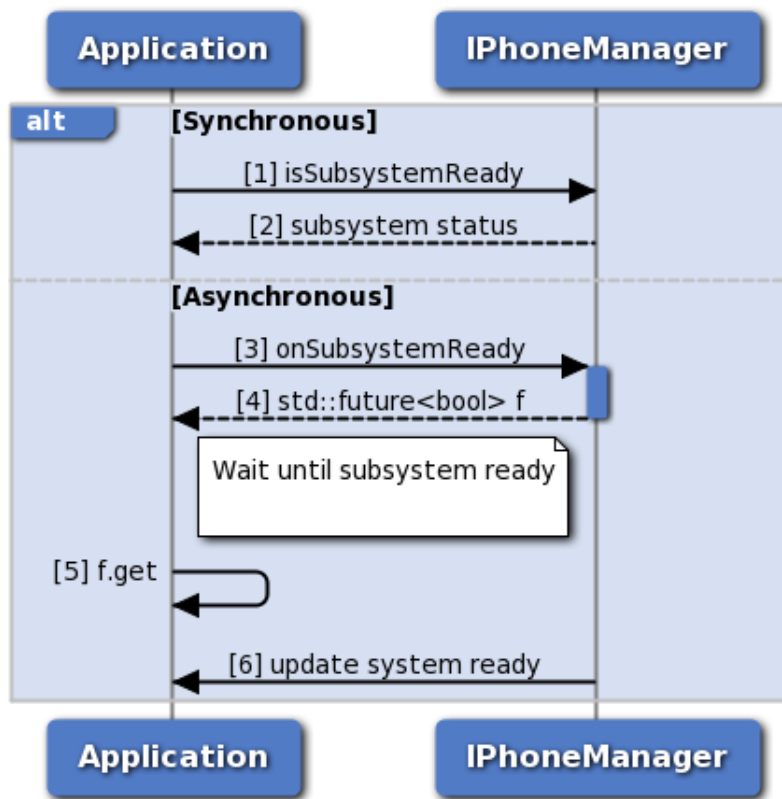


Figure 3-1 Phone manager initialization

1. Application can use iPhoneManager::isSubsystemReady to determine if the system is ready.
2. The application receives the status i.e. either true or false whether sub-system is ready or not.



3. If it is not ready, then the application could use onSubsystemReady which returns std::future.
4. PhoneManager notifies the application when the subsystem is ready through the std::future object.
5. The application waits until the asynchronous operation i.e onSubsystemReady completes.
6. PhoneManager updates the application once sub-system initialization completes.

## 3.2 Telephony

### 3.2.1 Dial call flow

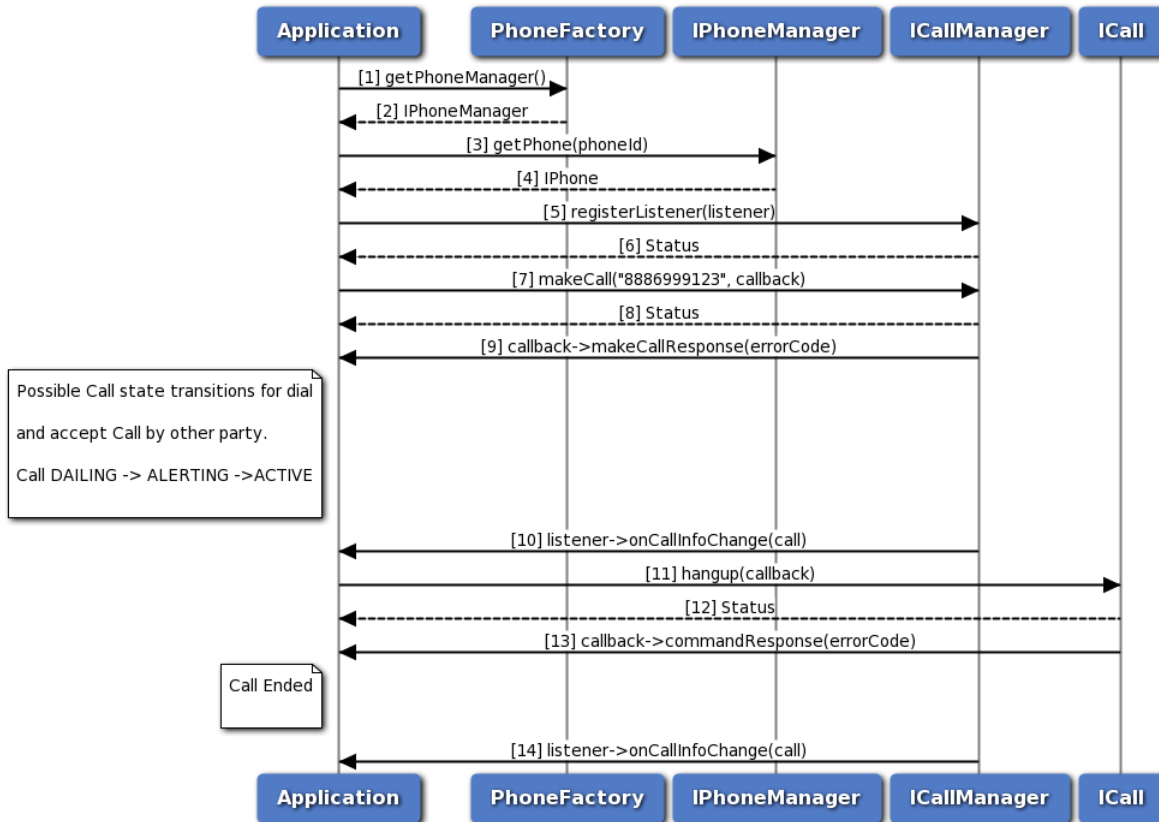


Figure 3-2 Dial call flow

1. The application gets the PhoneManager object using PhoneFactory.
2. The application receives the PhoneManager object in order to get Phone.
3. The application gets the Phone object for given phone identifier using PhoneManager.
4. PhoneManager returns Phone object to application. In case phone identifier is not specified, it returns default phone.
5. The application registers a listener with CallManager to listen to the call info change notifications like DIALING, ALERTING, ACTIVE and ENDED.
6. The application receives the status like SUCCESS or INVALIDPARAM based on registration of listener to CallManager.

7. The application dials a number by using makeCall API, optionally specifying callback to get asynchronous response.
8. The application receives the status like SUCCESS, INVALIDPARAM and FAILED etc based on the execution of makeCall API.
9. Optionally, the application gets asynchronous response for makeCall using makeCallResponseCallback.
10. The application receives callback on call info change like DIALING/ALERTING/ACTIVE when other party accepts the call.
11. The application sends hangup command to hangup the call, optionally specifying callback to get asynchronous response.
12. The application receives the status like SUCCESS, FAILED etc based on the execution of hangup API.
13. Optionally, the application gets asynchronous response for hangup using CommandResponseCallback.
14. The application receives callback on call info change i.e Call Ended from CallManager.

### 3.2.2 ECall call flow

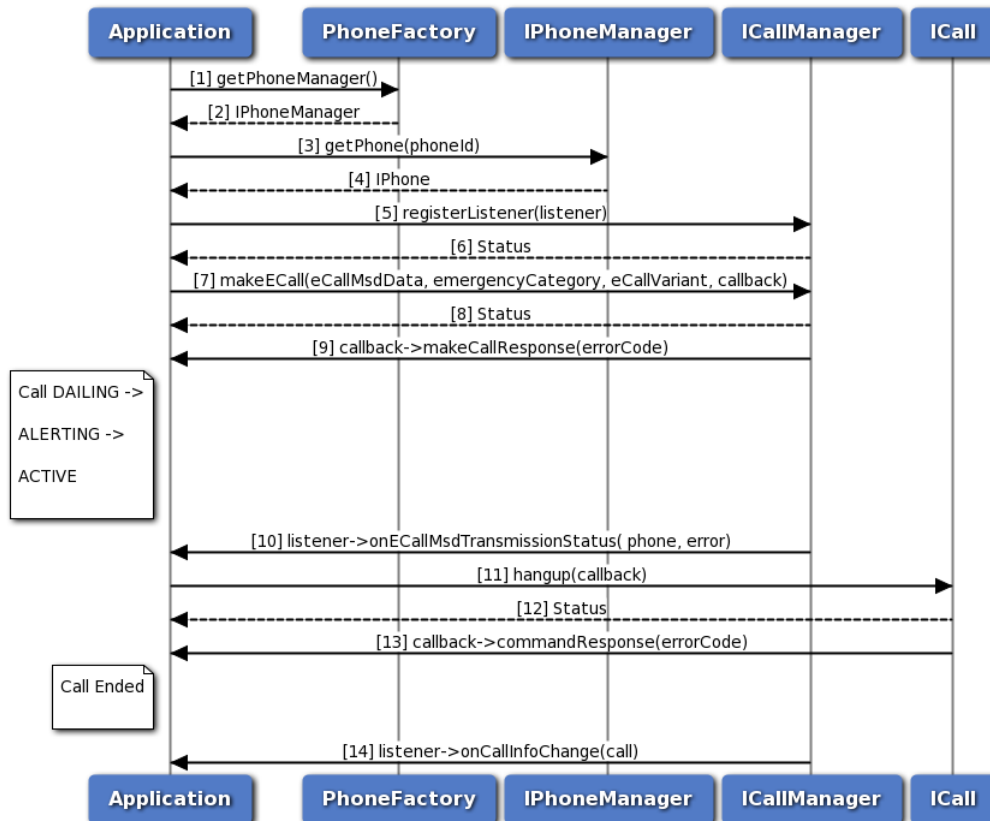


Figure 3-3 ECall call flow

1. The application gets the PhoneManager object using PhoneFactory.

2. The application receives the PhoneManager object in order to get Phone.
3. The application gets the Phone object optionally specifying phone identifier using PhoneManager.
4. PhoneManager returns Phone object to application, returns default phone in case phone identifier is not specified.
5. The application registers a listener with CallManager to listen to the call info change notifications like DIALING, ALERTING, ACTIVE etc.
6. The application receives the status like SUCCESS, FAILED etc based on registration of listener from CallManager.
7. The application dials an emergency call by using makeECall API, optionally specifying callback to get asynchronous response.
8. The application receives the status like SUCCESS, FAILED etc based on the execution of makeECall API.
9. Optionally, the application gets asynchronous response for makeECall using makeCallResponseCallback.
10. CallManager sends ecall msd transmission status to the application by using onECallMsdTransmissionStatus API.
11. The application sends hangup command to hangup the call, optionally gets asynchronous response using callback.
12. The application receives the status like SUCCESS, FAILED etc based on the execution of hangup API.
13. Optionally, the application gets asynchronous response for hangup using CommandResponseCallback.
14. CallManager sends call info change i.e Call Ended to the application by using onCallInfoChange callback function.

### 3.2.3 Signal strength call flow

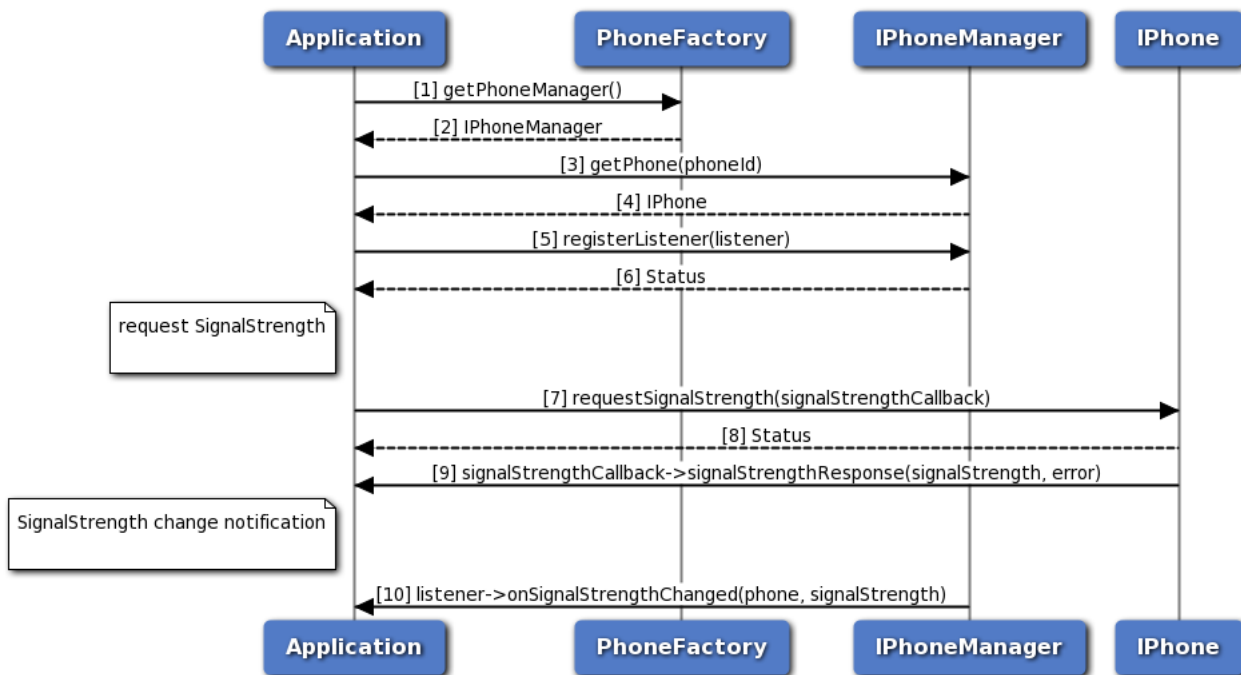


Figure 3-4 Signal strength call flow

1. The application gets PhoneManager object using PhoneFactory.
2. The application receives the PhoneManager object in order to get Phone instance.
3. The application gets phone instance for a given phone identifier using PhoneManager object.
4. PhoneManager returns IPhone object to the application.
5. Application registers the listener to get notification for signal strength change.
6. The application receives the status i.e. either SUCCESS or FAILED based on the registration of the listener.
7. The application requests for signal strength and optionally, gets asynchronous response using ISignalStrengthCallback.
8. The application receives the status i.e. either SUCCESS or FAILED based on execution of requestSignalStrength API in SapCardManager.
9. Optionally, the response for signal strength request is received by the application.
10. Application receives a notification when there is a change in signal strength.

### 3.2.4 Answer, Reject, RejectWithSMS call flow

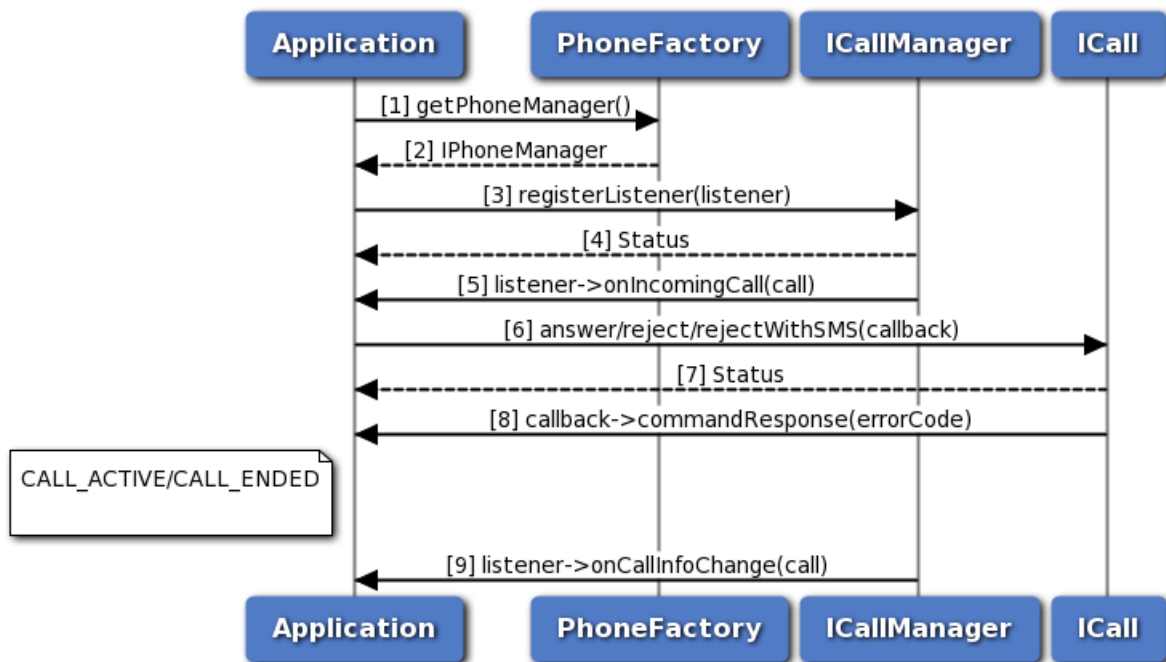
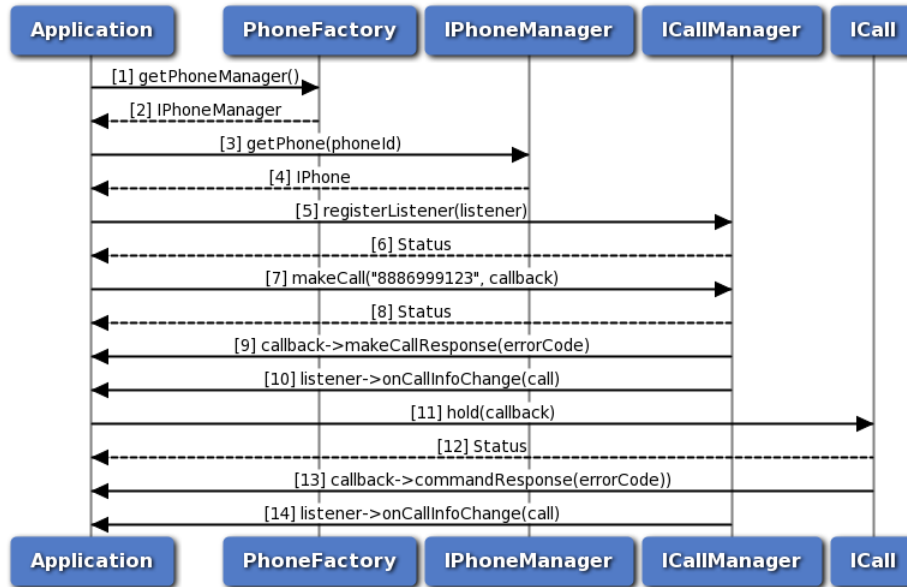


Figure 3-5 Answer, Reject, RejectWithSMS call flow

1. The application gets the PhoneManager object using PhoneFactory.
2. The application receives the PhoneManager object in order to register listener.
3. The application registers a listener with CallManager to listen incoming call notifications.
4. The application receives the status like SUCCESS, FAILED etc based on registration of listener from CallManager.
5. The application receives onIncomingCall notification when there is an incoming call.
6. The application performs answer/reject/rejectWithSMS operation using ICall.
7. The application receives the status like SUCCESS, FAILED etc based on execution of answer/reject/rejectWithSMS.
8. Optionally, the application gets asynchronous response for answer/reject/rejectWithSMS using CommandResponseCallback.
9. The CallManager sends call info change i.e CALL\_ACTIVE or CALL\_ENDED to the application by using onCallInfoChange callback function.

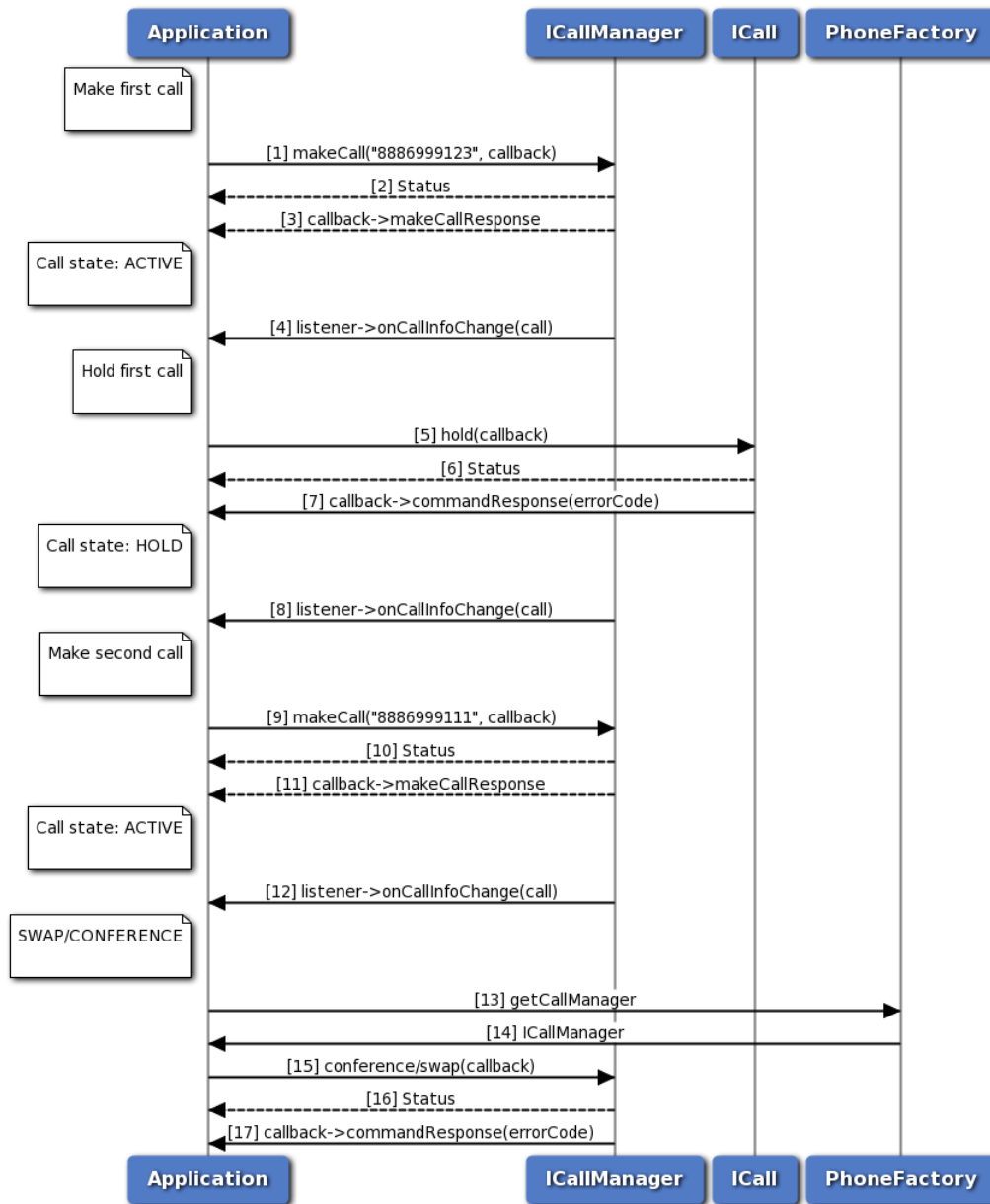
### 3.2.5 Hold call flow



**Figure 3-6 Hold call flow**

1. The application gets the PhoneManager object using PhoneFactory.
2. The application receives the PhoneManager object in order to get Phone.
3. The application gets the Phone object for given phone identifier using PhoneManager.
4. PhoneManager returns Phone object to application, returns default phone in case phone identifier is not specified.
5. The application registers a listener with CallManager to listen to the call info change notifications like DIALING, ALERTING, ACTIVE etc.
6. The application receives the status like SUCCESS or INVALIDPARAM based on registration of listener to CallManager.
7. The application dials a number by using makeCall API, optionally specifying callback to get asynchronous response.
8. The application receives the status like SUCCESS, INVALIDPARAM and FAILED etc based on the execution of makeCall API.
9. Optionally, the application gets asynchronous response for makeCall using makeCallResponseCallback.
10. The CallManager sends call info change i.e CALL\_ACTIVE to application by using onCallInfoChange API.
11. The application sends hold command to hold the call, optionally specifying callback to get asynchronous response.
12. The application receives the status like SUCCESS, FAILED etc based on the execution of hold API.
13. Optionally, the application gets asynchronous response for hold using CommandResponseCallback.
14. The application receives call info change i.e CALL\_ON\_HOLD from CallManager.

### 3.2.6 Hold, Conference, Swap call flow



**Figure 3-7 Hold, Conference, Swap call flow**

1. The application makes first call using ICall.
2. The application receives the status like SUCCESS, FAILED etc based on execution of makeCall operation.
3. Optionally, the application gets asynchronous response for makeCall using makeCallResponseCallback.
4. The CallManager sends call info change i.e CALL\_ACTIVE to application by using onCallInfoChange API.
5. The application sends hold command to hold the call, optionally specifying callback to get

- asynchronous response.
6. The application receives the status like SUCCESS, FAILED etc based on the execution of hold API.
  7. Optionally, the application gets asynchronous response for hold using CommandResponseCallback.
  8. The application receives call info change i.e CALL\_ON\_HOLD from CallManager.
  9. The application makes second call using ICall.
  10. The application receives the status like SUCCESS, FAILED etc based on execution of makeCall operation.
  11. Optionally, the application gets asynchronous response for makeCall using makeCallResponseCallback.
  12. The CallManager sends call info change i.e CALL\_ACTIVE to application by using onCallInfoChange API.
  13. The application requests the PhoneFactory to get ICallManager object.
  14. The application receives the ICallManager object using PhoneFactory.
  15. The application performs conference/swap operation using ICallManager by passing first call and second call. optionally application can pass callback to receive hold response asynchronously.
  16. The application receives the status like SUCCESS, FAILED etc based on the execution of conference/swap API.
  17. Optionally, the application gets asynchronous response for conference/swap using CommandResponseCallback.



### 3.2.7 SMS call flow

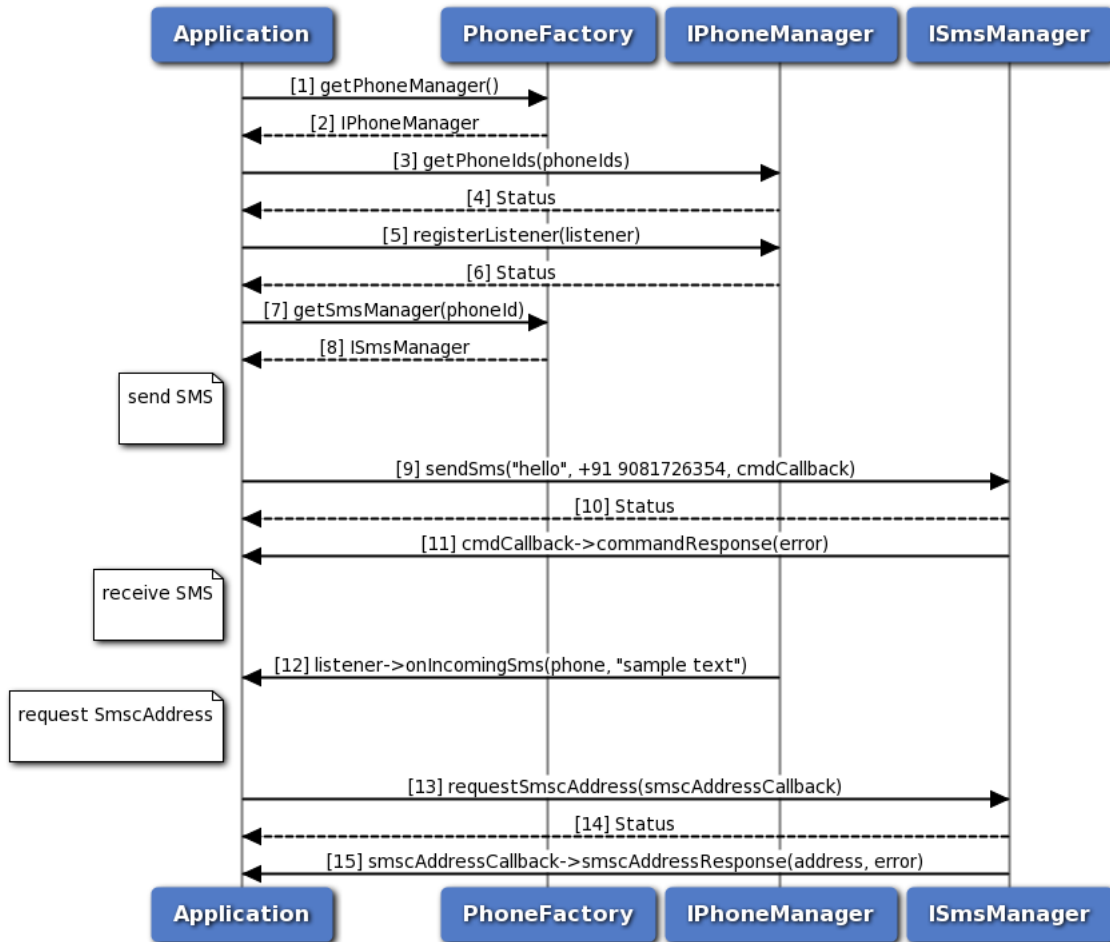


Figure 3-8 SMS call flow

1. Application gets PhoneManager object using PhoneFactory.
2. PhoneFactory returns the PhoneManager object to application in order to get list of phone identifiers.
3. The application retrieves the list of phone identifier on the device using PhoneManager object.
4. Application receives the status i.e. either SUCCESS or FAILED based on the execution of getPhoneIds API in PhoneManager.
5. Application registers the listener for incoming SMS with IPhoneManager.
6. The application receives the status i.e. either SUCCESS or INVALIDPARAM based on successful registration of the listener.
7. The application gets SmsManager object corresponding to specific phone identifier.
8. PhoneFactory returns the SmsManager object to application in order to perform operations like send SMS and get SMSC address.
9. The application sends SMS to the receiver address and optionally gets asynchronous response using CommandResponseCallback.
10. Application receives the status i.e. either SUCCESS or FAILED based on execution of sendSms API

in SmsManager.

11. Optionally, the response for send SMS is received by the application.
12. Application gets notified for incoming SMS.
13. The application requests for SmscAddress and optionally gets asynchronous response using ISmscAddressCallback.
14. Application receives the status i.e. either SUCCESS or FAILED based on successful execution of requestSmscAddress API in SmsManager.
15. Optionally, the application receives the SMSC address on success or gets error on failure in the command response callback.

### 3.2.8 Radio and Service state call flow

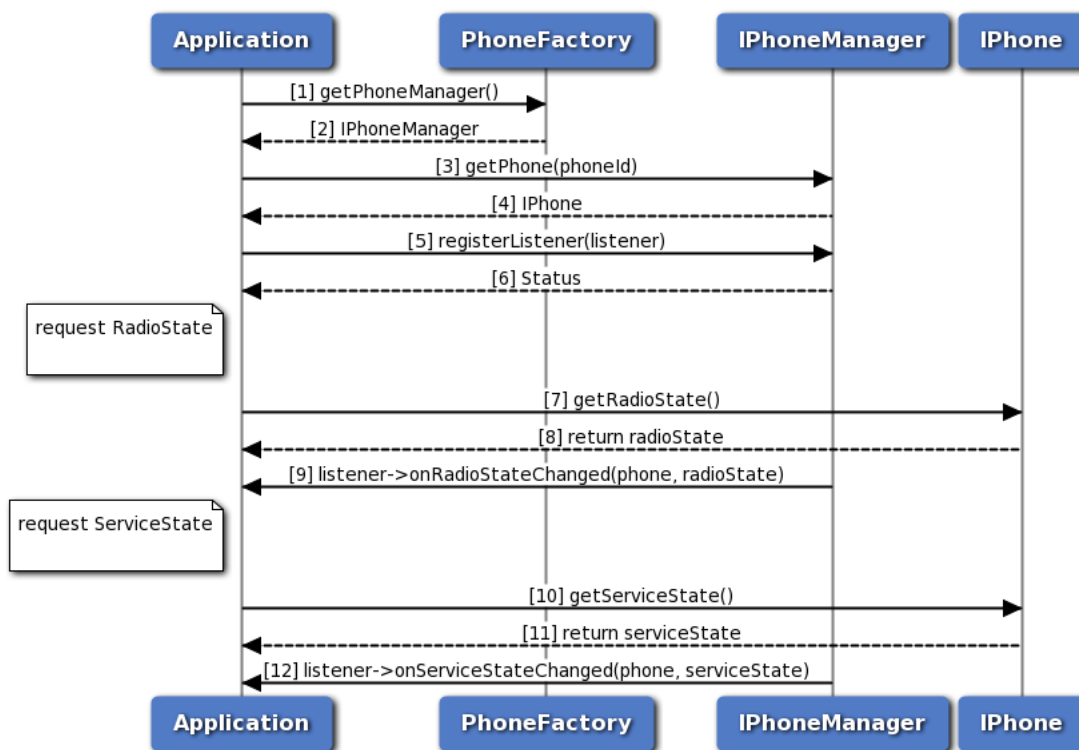


Figure 3-9 Radio and Service state call flow

1. The application gets PhoneManager object using PhoneFactory.
2. The application receives the PhoneManager object in order to get Phone instance.
3. The application gets phone instance for a given phone identifier using PhoneManager object.
4. PhoneManager returns IPhone object to the application.
5. Application registers the listener to get notifications for radio and service state change.
6. The application receives the status i.e. either SUCCESS or FAILED based on the registration of the listener.
7. The application request the Phone to get radio state.

8. The application receives the radio state like RADIO\_STATE\_ON, OFF or UNAVAILABLE.
9. Application receives a notification when there is a change in radio state.
10. The application request the Phone to get service state.
11. The application receives the service state like IN\_SERVICE, OUT\_OF\_SERVICE, EMERGENCY\_ONLY or RADIO\_OFF.
12. Application receives a notification when there is a change in service state.

### 3.2.9 Network Selection Manager call flow

Network selection manager provides APIs to get and set network selection mode, get and set preferred networks and perform network scan for availbale networks. Registered listener will get notified for the change in network selection mode.

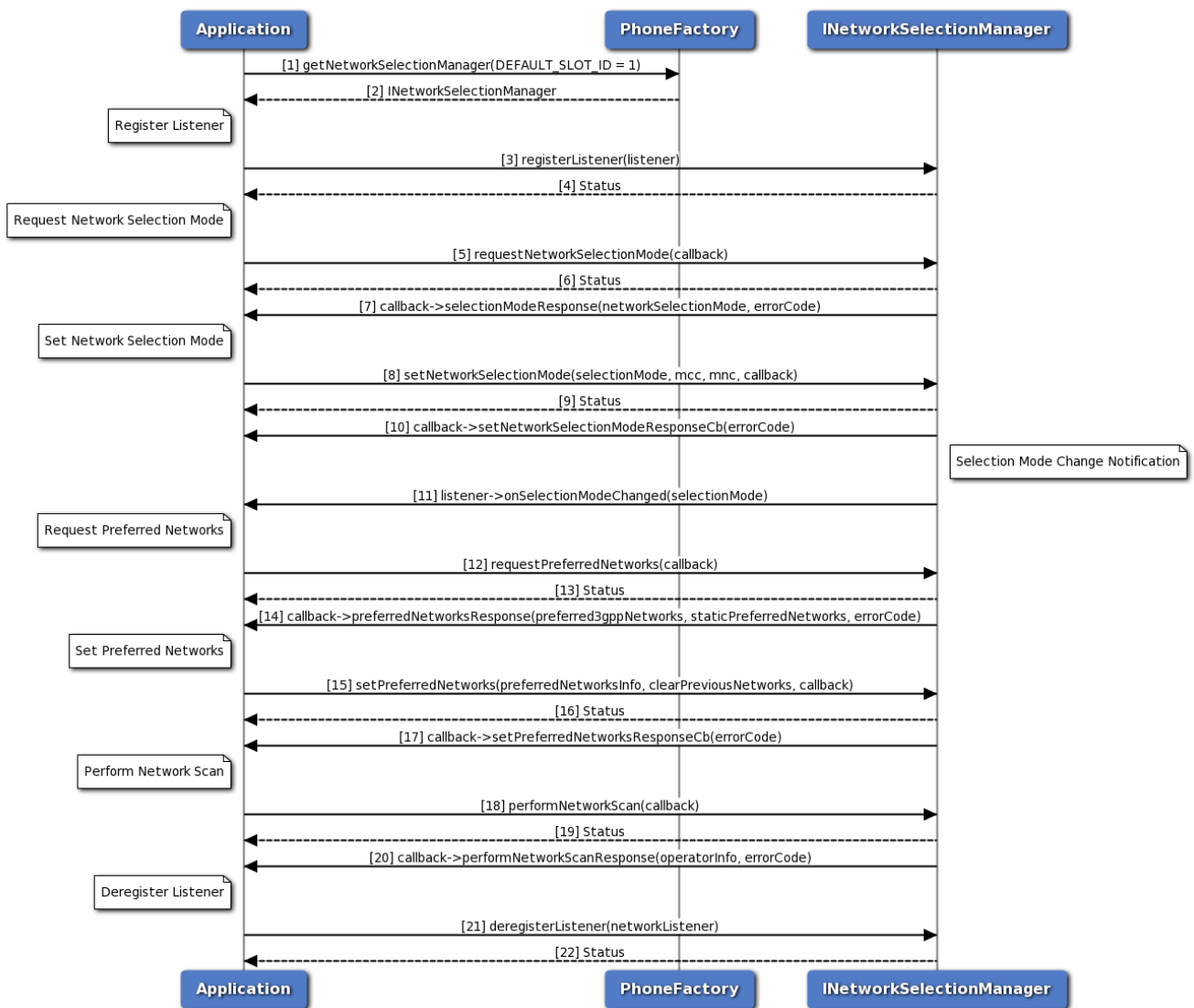


Figure 3-10 Network selection manager call flow

1. Application requests phone factory for network selection manager.
2. Phone factory returns INetworkSelectionManager object using which application will register or deregister a listener.
3. Application can register a listener to get notifications for network selection mode change.
4. Status of register listener i.e. either SUCCESS or other status will be returned to the application.
5. Application requests for network selection mode using INetworkSelectionManager object and gets asynchronous response using SelectionModeResponseCallback.
6. The application receives the status i.e. either SUCCESS or other status based on the execution of requestNetworkSelectionMode API.
7. The response for get network selection mode request is received by the application.
8. The application can also set network selection mode and optionally gets asynchronous response using ResponseCallback. MCC and MNC are optional for AUTOMATIC network selection mode.
9. Application receives the status i.e. either SUCCESS or other status based on the execution of setNetworkSelectionMode API.
10. Optionally the response for set network selection mode request is received by the application.
11. Registered listener will get notified for the network selection mode change.
12. Similarly, the application requests for preferred networks using INetworkSelectionManager object and gets asynchronous response using PreferredNetworksCallback.
13. The application receives the status i.e. either SUCCESS or other status based on the execution of requestPreferredNetworks API.
14. The response for get preferred networks request i.e. 3GPP preferred network list and static 3GPP preferred network list is received by the application asynchronously. Higher priority networks appear first in the list. The networks that appear in the 3GPP Preferred Networks list get higher priority than the networks in the static 3GPP preferred networks list.
15. The application can set 3GPP preferred network list and optionally gets asynchronous response using ResponseCallback. If clear previous networks flag is false then new 3GPP preferred network list is appended to existing preferred network list. If flag is true then old list is flushed and new 3GPP preferred network list is added.
16. Application receives the status i.e. either SUCCESS or other status based on the execution of setPreferredNetworks API.
17. Optionally the response for set preferred networks request is received by the application.
18. The application can perform network scan for available networks using INetworkSelectionManager object and gets asynchronous response using NetworkScanCallback.
19. Application receives the status i.e. either SUCCESS or other status based on the execution of performNetworkScan API.
20. Network name, MCC, MNC and status of the operator will be received by the application.
21. Application can deregister a listener there by it would not get notifications.
22. Status of deregister listener i.e. either SUCCESS or other status will be returned to the application.

### 3.2.10 Serving System Manager Call Flow

Serving system manager provides APIs to get and set RAT mode preference and get and set service domain preference. Registered listener will get notified for the change in RAT mode and service domain preference change.

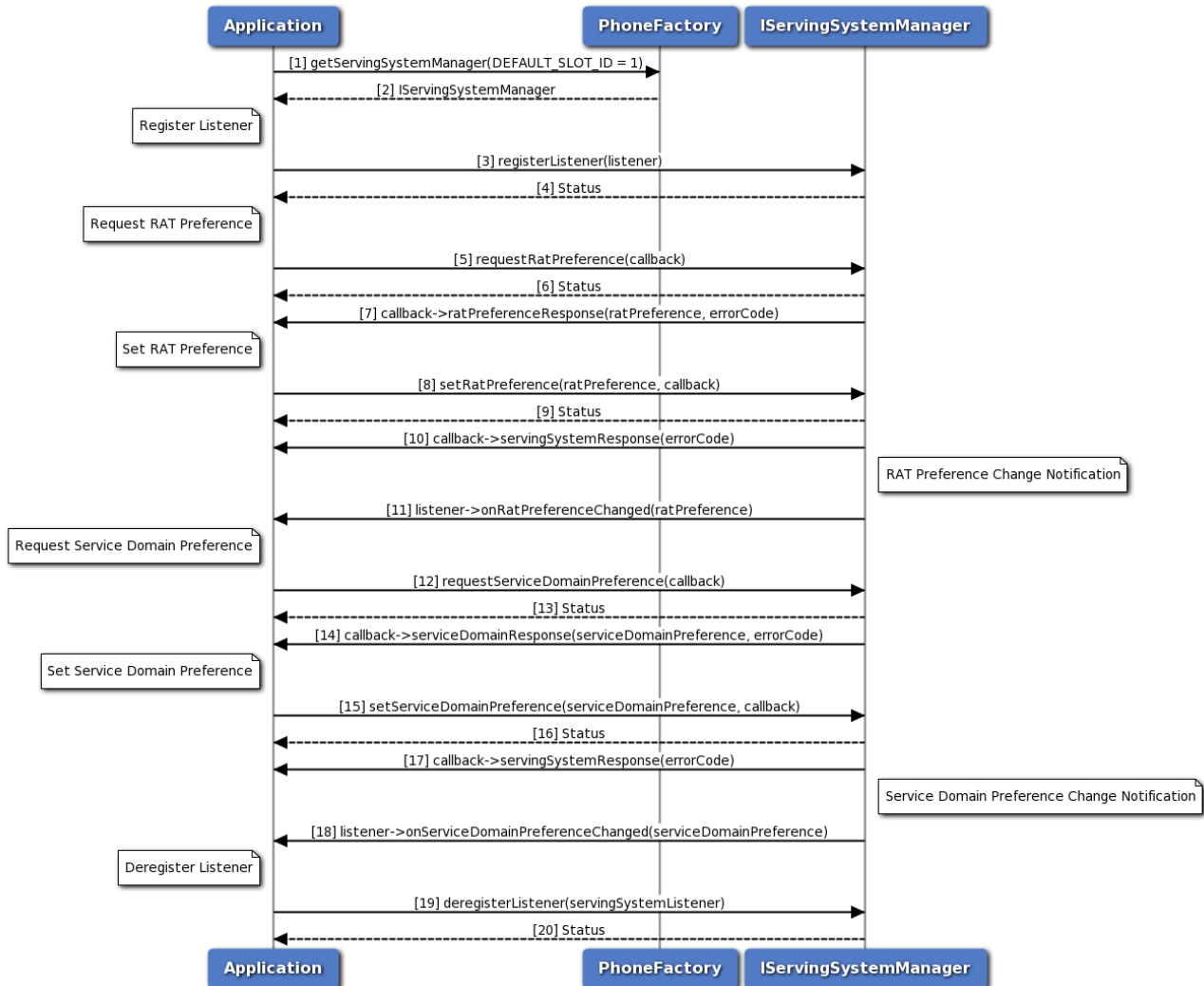


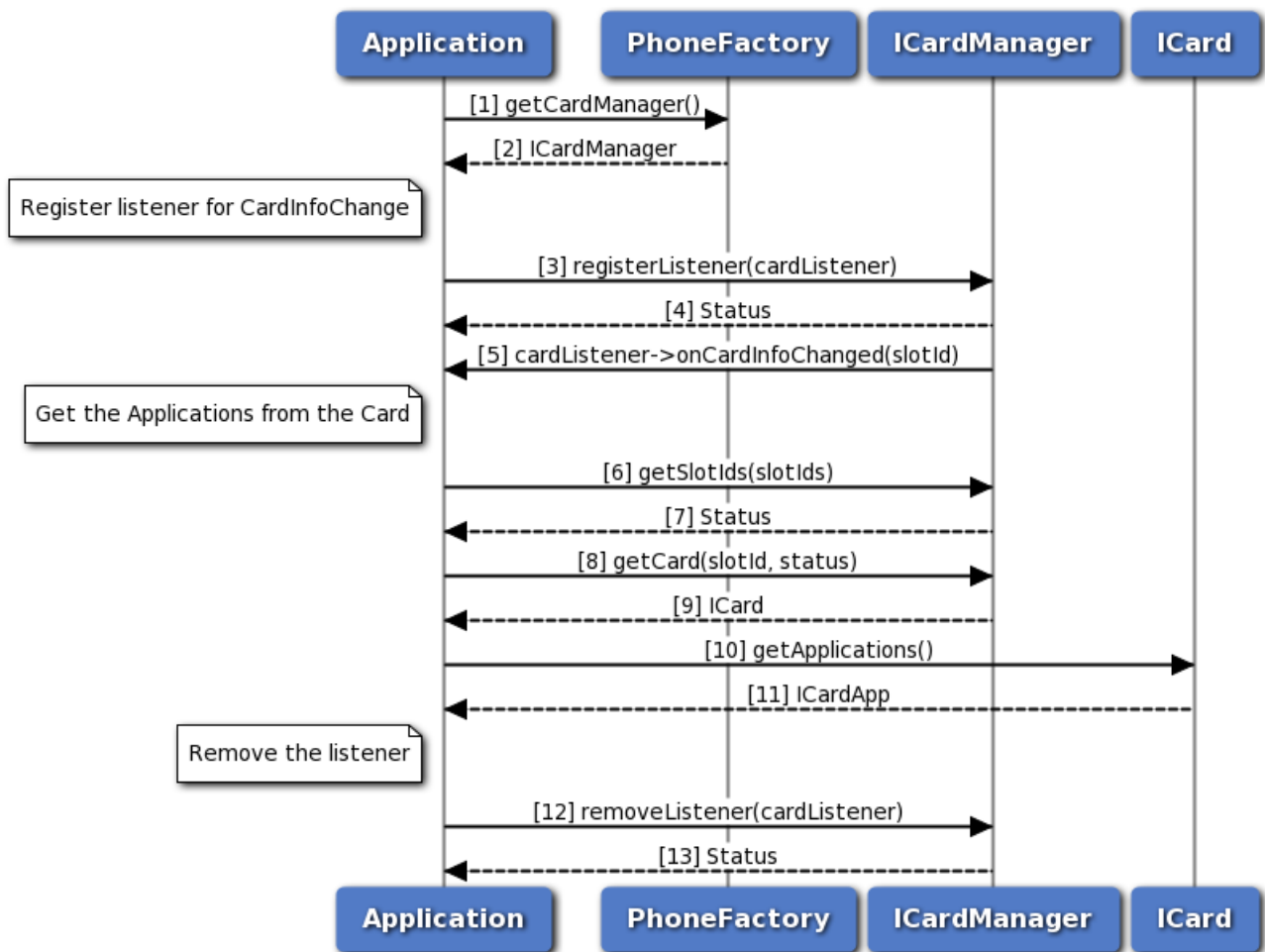
Figure 3-11 Serving System Manager Call Flow

1. Application requests phone factory for serving system manager.
2. Phone factory returns `IServingSystemManager` object using which application will register or deregister a listener.
3. Application can register a listener to get notifications for RAT mode and service domain preference changes.
4. Status of register listener i.e. either SUCCESS or other status will be returned to the application.
5. Application requests for RAT mode preference using `IServingSystemManager` object and gets asynchronous response using `RatPreferenceCallback`.

6. The application receives the status i.e. either SUCCESS or other status based on the execution of requestRatPreference API.
7. The response for get RAT preference request is received by the application.
8. The application can also set RAT mode preference and optionally gets asynchronous response using ResponseCallback.
9. Application receives the status i.e. either SUCCESS or other status based on the execution of setRatPreference API.
10. Optionally the response for set RAT preference request is received by the application.
11. Registered listener will get notified for the RAT mode preference change.
12. Application requests for service domain preference using IServingSystemManager object and gets asynchronous response using ServiceDomainPreferenceCallback.
13. The application receives the status i.e. either SUCCESS or other status based on the execution of requestServiceDomainPreference API.
14. The response for get service domain preference request is received by the application.
15. The application can also set service domain preference and optionally gets asynchronous response using ResponseCallback.
16. Application receives the status i.e. either SUCCESS or other status based on the execution of setServiceDomainPreference API.
17. Optionally the response for set service domain preference request is received by the application.
18. Registered listener will get notified for the service domain preference change.
19. Application can deregister a listener there by it would not get notifications.
20. Status of deregister listener i.e. either SUCCESS or other status will be returned to the application.

### 3.3 Card Services

### 3.3.1 Get applications call flow



**Figure 3-12 Get applications call flow**

1. Application gets the CardManager object from PhoneFactory.
2. Application receives CardManager object in order to perform operations like getSlotIds and getCard.
3. The application registers a listener for Card info change event with CardManager.
4. Application receives the status i.e. either SUCCESS or INVALIDPARAM based on the registration of listener.
5. The response from onCardInfoChanged is received by the application whenever there is card info change.
6. The application gets the slotIds from the sub-system using CardManager.
7. Application receives the status i.e. either SUCCESS or NOTREADY along with the updated slotIds.
8. Then, the application sends request to CardManager to get Card object for a specific slotId.
9. Application receives Card object from CardManager in order to perform card operation like getApplications.

10. The application gets the CardApps from Card object.
11. The application receives CardApps which contain information such as AppId, AppType and AppState.
12. Now the application removes the listener associated with CardManager.
13. Application receives the status i.e. either SUCCESS or NOSUCH for the removal of listener.

### 3.3.2 Transmit APDU call flow

#### 3.3.2.1 On logical channel

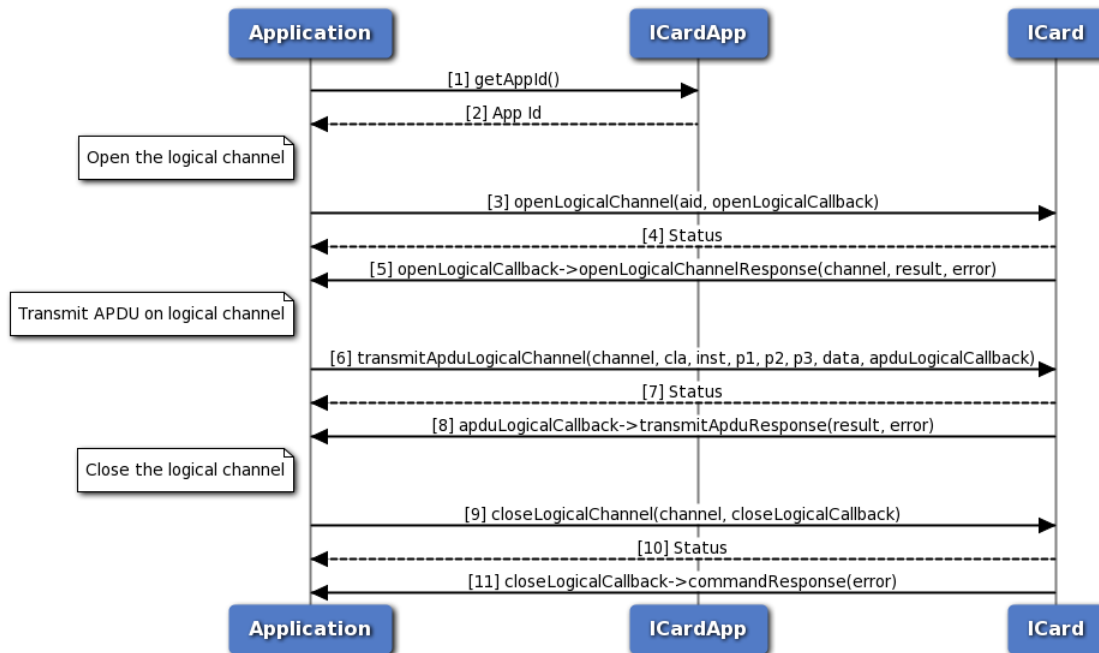


Figure 3-13 On logical channel

1. The Application requests CardApp for the SIM application identifier.
2. The application receives the application identifier to perform open logical channel.
3. Application sends request to open the logical channel with the application identifier and optionally, gets asynchronous response in OpenLogicalChannelCallback.
4. The application receives the status i.e. either SUCCESS or FAILED based on execution of openLogicalChannel API.
5. Optionally, the application receives the response which contains either channel number on success or error in case of failure.
6. Then, the application transmits the APDU data on logical channel using the channel obtained earlier. Optionally, gets asynchronous response in TransmitAduResponseCallback.
7. The application receives the status i.e. either SUCCESS or FAILED based on execution of transmitAduLogicalChannel API.
8. Optionally, the application receives the response which contains either result on success or error in



- case of failure.
9. Finally, the application closes the logical channel that is opened to transmit APDU and optionally, gets asynchronous response in `CommandResponseCallback`.
  10. The application receives the status i.e. either `SUCCESS` or `FAILED` based on execution of `closeLogicalChannel` API.
  11. Optionally, the application receives the response which contains error in case of failure.

### 3.3.2.2 On basic channel

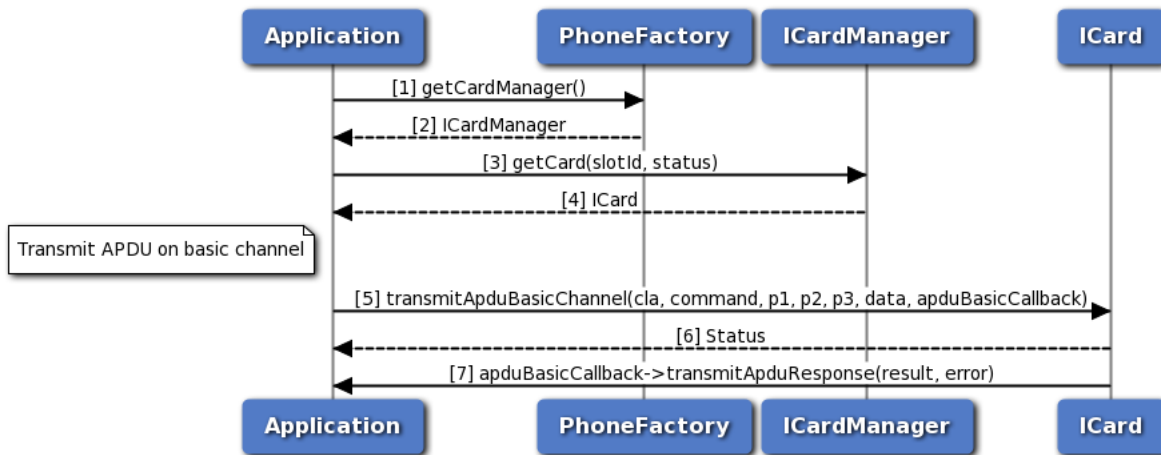


Figure 3-14 On basic channel

1. Application gets the `ICardManager` object from `PhoneFactory`.
2. Application receives `ICardManager` object in order to perform operation like `getCard`.
3. The application gets `ICard` object for a specific `slotId` from `ICardManager`.
4. Application receives `ICard` object in order to perform card operation like `transmitApduBasicChannel`.
5. The application transmits APDU data on basic channel and optionally, gets asynchronous response in `TransmitApduResponseCallback`.
6. The application receives the status i.e. either `SUCCESS` or `FAILED` based on execution of `transmitApduBasicChannel` API.
7. Optionally, the application receives the response which contains either result on success or error in case of failure.

### 3.3.3 SAP card manager call flow

### 3.3.3.1 Request card reader status, Request ATR, Transmit APDU call flow

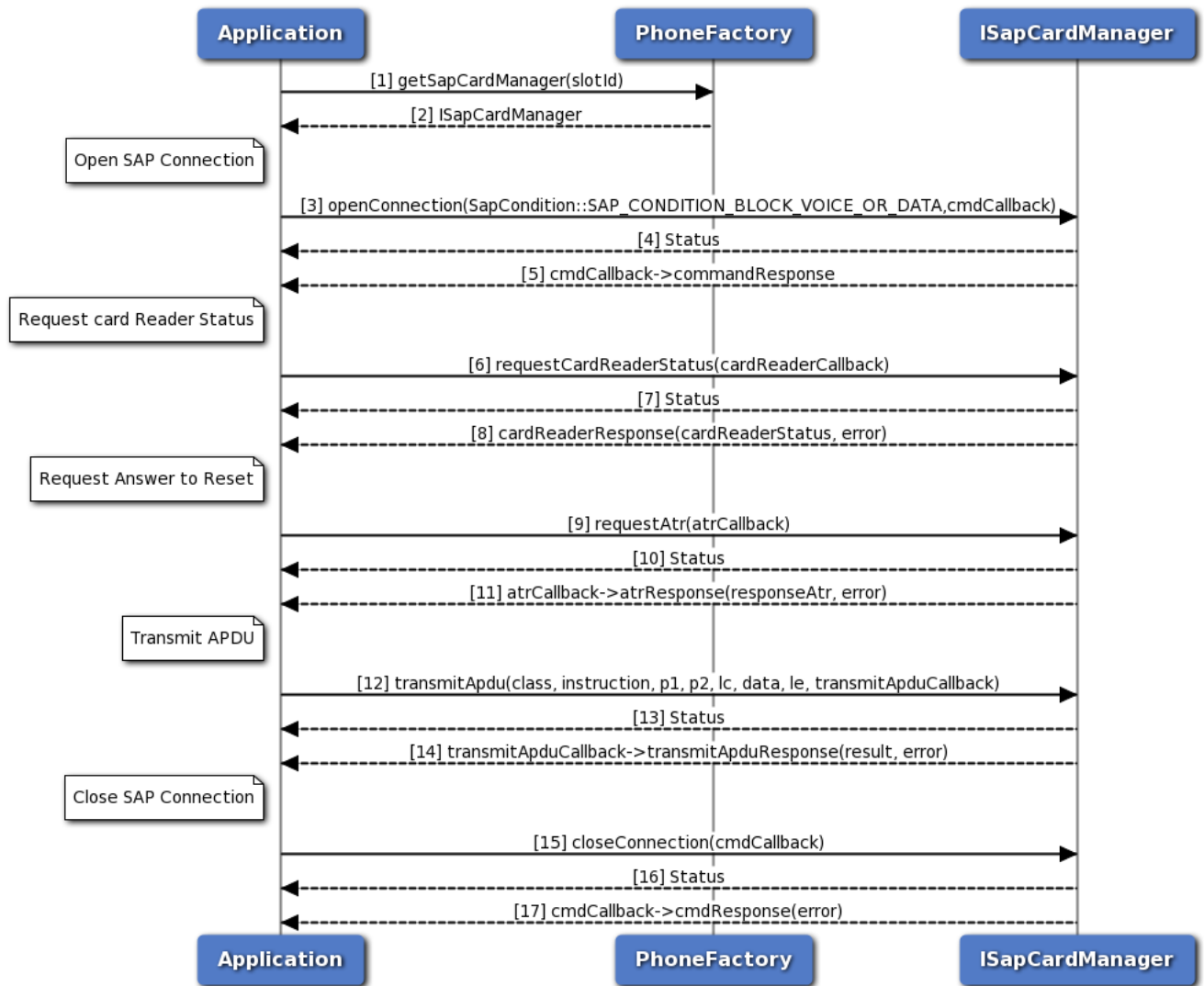


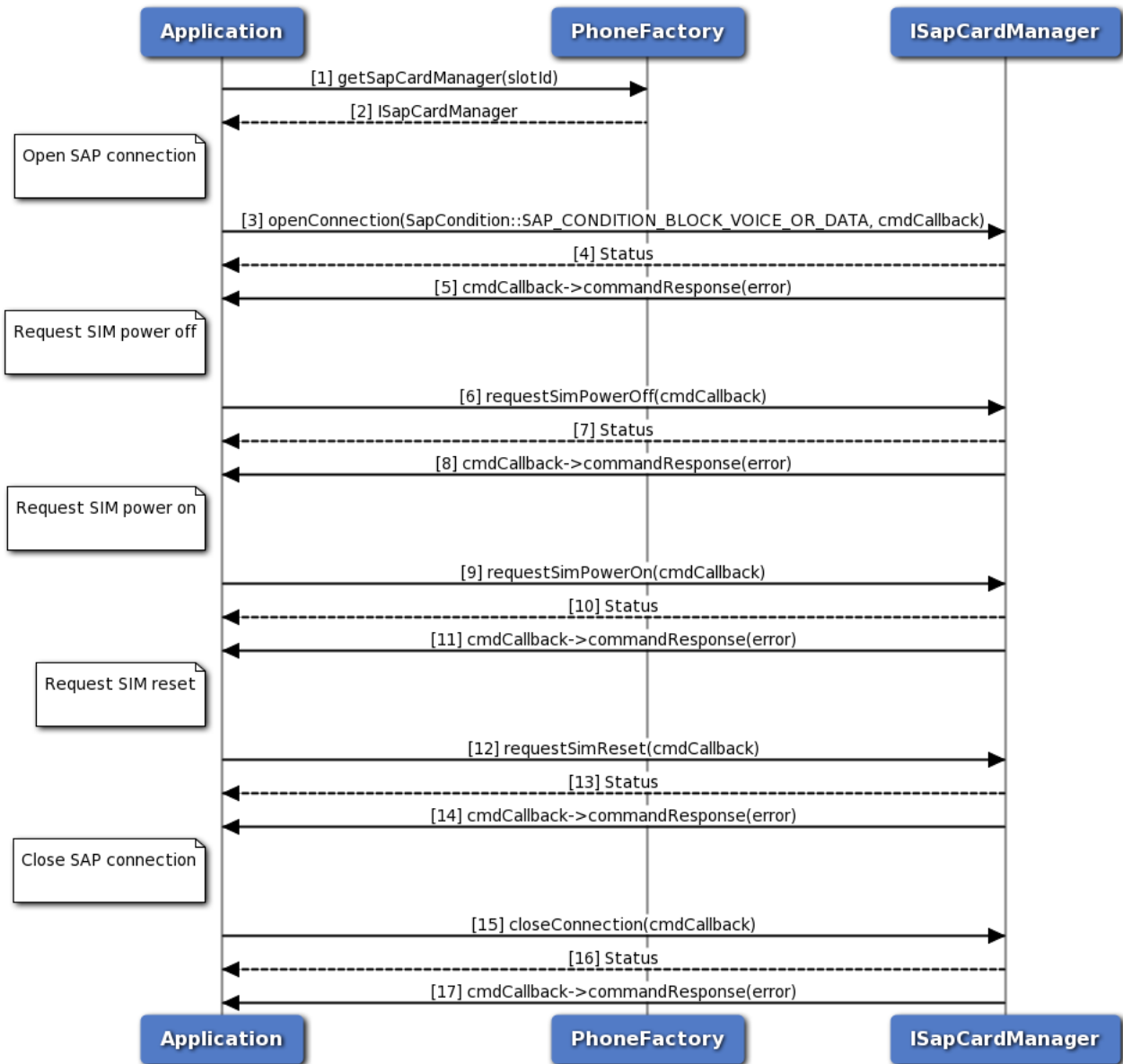
Figure 3-15 Request card reader status, Request ATR, Transmit APDU call flow

{sap\_card\_operations.png,Request card reader status, Request ATR, Transmit APDU call flow,80,80,Request card reader status, Request ATR, Transmit APDU call flow }

1. The application gets SapCardManager object corresponding to slotId using PhoneFactory.
2. The application receives the SapCardManager object in order to perform SAP operations like request ATR, Card Reader Status and transmit APDU.
3. The application opens SIM Access Profile(SAP) connection with SIM card using default SAP condition (i.e. SAP\_CONDITION\_BLOCK\_VOICE\_OR\_DATA) and optionally, gets asynchronous response using CommandResponseCallback.
4. The application receives the status i.e. either SUCCESS or FAILED based on execution of openConnection API in SapCardManager.
5. Optionally, the response for openConnection is received by the application.

6. The application sends request card reader status command and optionally, gets asynchronous response using ICardReaderCallback.
7. The application receives the status i.e. either SUCCESS or FAILED based on execution of requestCardReaderStatus API in SapCardManager.
8. Optionally, the response for card reader status is received by the application.
9. Similarly, the application can send SAP Answer To Reset command and optionally, gets asynchronous response using IAttrResponseCallback.
10. The application receives the status i.e. either SUCCESS or FAILED based on execution of requestAttr API in SapCardManager.
11. Optionally, the response for SAP Answer To Reset is received by the application.
12. Similarly, the application sends the APDU on SAP mode and optionally, gets asynchronous response using ISapTransmitApduResponseCallback.
13. The application receives the status i.e. either SUCCESS or FAILED based on execution of transmitApdu API in SapCardManager.
14. Optionally, the response for transmit APDU is received by the application.
15. Now the application closes the SAP connection with SIM and optionally, gets asynchronous response using CommandResponseCallback.
16. The application receives the status i.e. either SUCCESS or FAILED based on execution of closeConnection API in SapCardManager.
17. Optionally, the response for SAP close connection is received by the application.

### 3.3.3.2 SIM Turn off, Turn on and Reset call flow



**Figure 3-16 SIM Turn off, Turn on and Reset call flow**

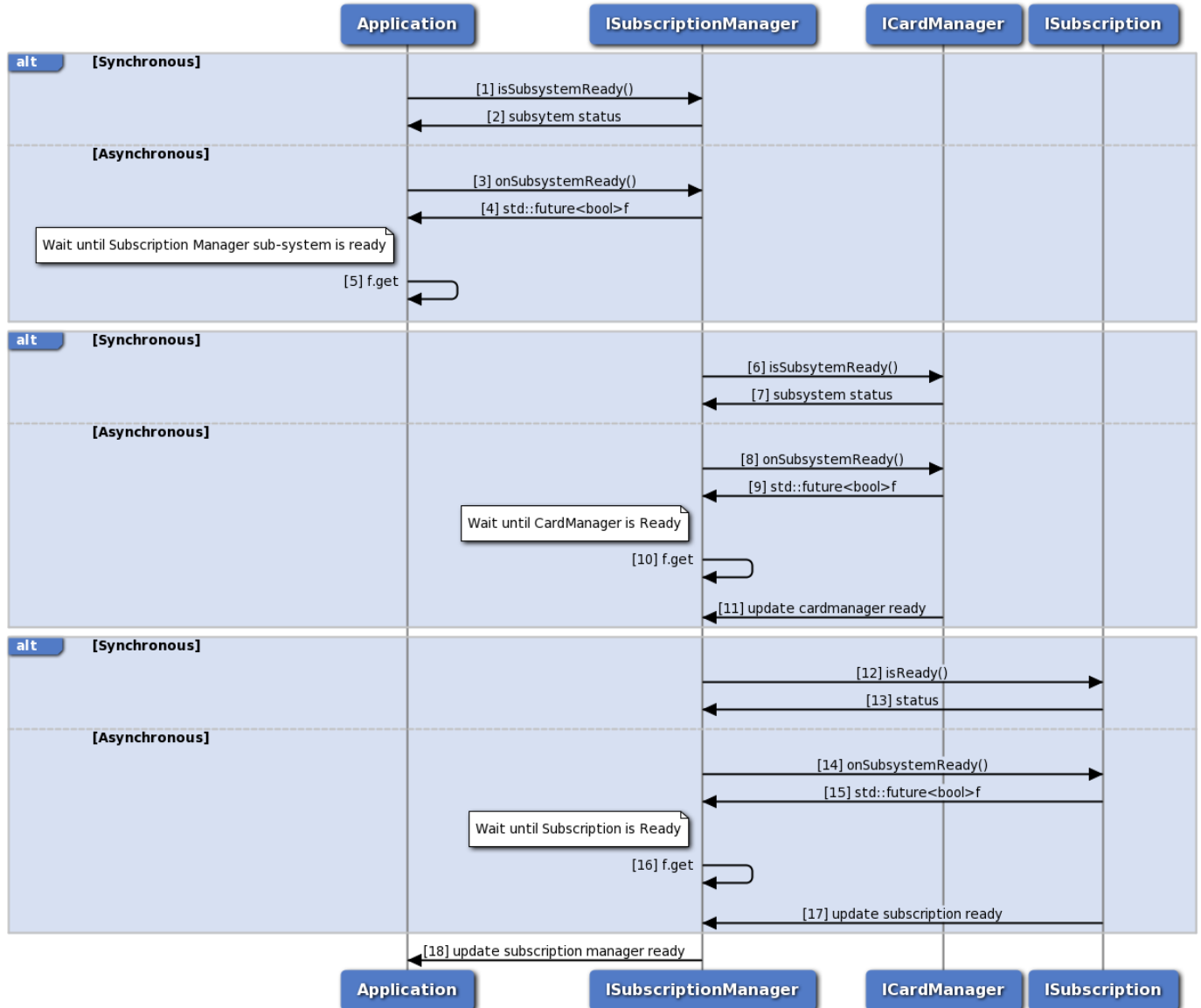
{sap\_mgr\_sim\_call\_flow.png, SIM Turn off, Turn on and Reset call flow, 80, 80, SIM Turn off, Turn on and Reset call flow}

1. Application gets SapCardManager object corresponding to slotID using PhoneFactory.
2. PhoneFactory returns the SapCardManager object to application in order to perform SAP operations like SIM power off, on or reset.
3. The application opens SIM Access Profile(SAP) connection with SIM card using default SAP condition (i.e. SAP\_CONDITION\_BLOCK\_VOICE\_OR\_DATA) and optionally, gets asynchronous response using CommandResponseCallback.

4. Application receives the status i.e. either SUCCESS or FAILED based on execution of openConnection API in SapCardManager.
5. Optionally, the response for openConnection is received by the application.
6. The application sends SIM Power Off command to turn off the SIM and optionally, gets asynchronous response using CommandResponseCallback.
7. The application receives the status i.e. either SUCCESS or FAILED based on execution of requestSimPowerOff API in SapCardManager.
8. Optionally, the response for SIM Power Off is received by the application.
9. Similarly, the application can send SIM Power On command to turn on the SIM and optionally, gets asynchronous response using CommandResponseCallback.
10. The application receives the status i.e. either SUCCESS or FAILED based on execution of requestSimPowerOn API in SapCardManager.
11. Optionally, the response for SIM Power On is received by the application.
12. Similarly, the application sends SIM Reset command to perform SIM Reset and optionally, gets asynchronous response.
13. The application receives the status i.e. either SUCCESS or FAILED based on execution of requestSimReset API in SapCardManager.
14. Optionally, the response for SIM Reset is received by the application.
15. Now the application closes the SAP connection with SIM and optionally, gets asynchronous response using CommandResponseCallback.
16. The application receives the status i.e. either SUCCESS or FAILED based on execution of closeConnection API in SapCardManager.
17. Optionally, the response for SAP close connection is received by the application.

### 3.3.4 Subscription Call flow

### 3.3.4.1 Subscription initialization

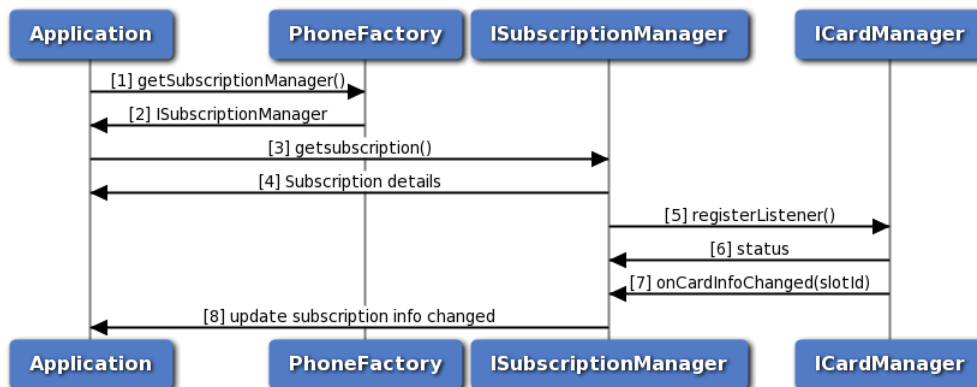


**Figure 3-17 Subscription initialization call flow**

1. Application can use ISubscriptionManager::isSubsystemReady to determine if the SubscriptionManager is ready.
2. The application receives the status i.e either true or false whether sub-system is ready or not.
3. If it is not ready, then the application could use onSubsystemReady which returns std::future.
4. SubscriptionManager notifies the application when the subsystem is ready through the std::future object.
5. The application waits until the asynchronous operation i.e onSubsystemReady completes.
6. SubscriptionManager uses ICardManager::isSubsystemReady to determine if the CardManager is ready.

7. The SubscriptionManager receives the status i.e either true or false whether sub-system is ready or not.
8. If it is not ready, then the SubscriptionManager could use onSubsystemReady which returns std::future.
9. CardManager notifies the SubscriptionManager when the subsystem is ready through the std::future object.
10. The SubscriptionManager waits until the asynchronous operation i.e onSubsystemReady completes.
11. CardManager updates the SubscriptionManager once the card manager sub-system is ready
12. SubscriptionManager uses ISubscription::isReady to determine if the Subscription is ready.
13. The SubscriptionManager receives the status i.e either true or false whether sub-system is ready or not.
14. If it is not ready, then the SubscriptionManager could use onSubsystemReady which returns std::future.
15. Subscription notifies the SubscriptionManager when the subsystem is ready through the std::future object.
16. The SubscriptionManager waits until the asynchronous operation i.e onSubsystemReady completes.
17. Subscription updates the SubscriptionManager when the subscription is ready.
18. SubscriptionManager updates the application once subscription manager initialization completes.

### 3.3.4.2 Subscription call flow



**Figure 3-18 Subscription call flow**

1. The application gets the PhoneManager object using PhoneFactory.
2. The application receives the PhoneManager object in order to get Subscription.
3. The application gets the Subscription object for given slot identifier using SubscriptionManager.
4. SubscriptionManager returns Subscription object to application. Subscription can be used to get subscription details like countryISO, operator details etc.
5. The Subscription manager registers a listener with CardManager to listen to the card info change notifications like card state PRESENT, ABSENT, UNKNOWN, ERROR and RESTRICTED.

6. The SubscriptionManager receives the status like SUCCESS or INVALIDPARAM based on registration of listener to CardManager.
7. The SubscriptionManager receives callback card info change i.e subscription info changed or removed.
8. The SubscriptionManager updates the application once the subscription info is updated.

### 3.4 Call flow for location services

Application will get the location manager object from location factory. The caller needs to register a listener. Application would then need to start the reports using one of 2 APIs depending on if the detailed or basic reports are needed. When reports are no longer required, the app needs to stop the report and de-register the listener.

#### 3.4.1 Call flow to register/remove listener for generating basic reports

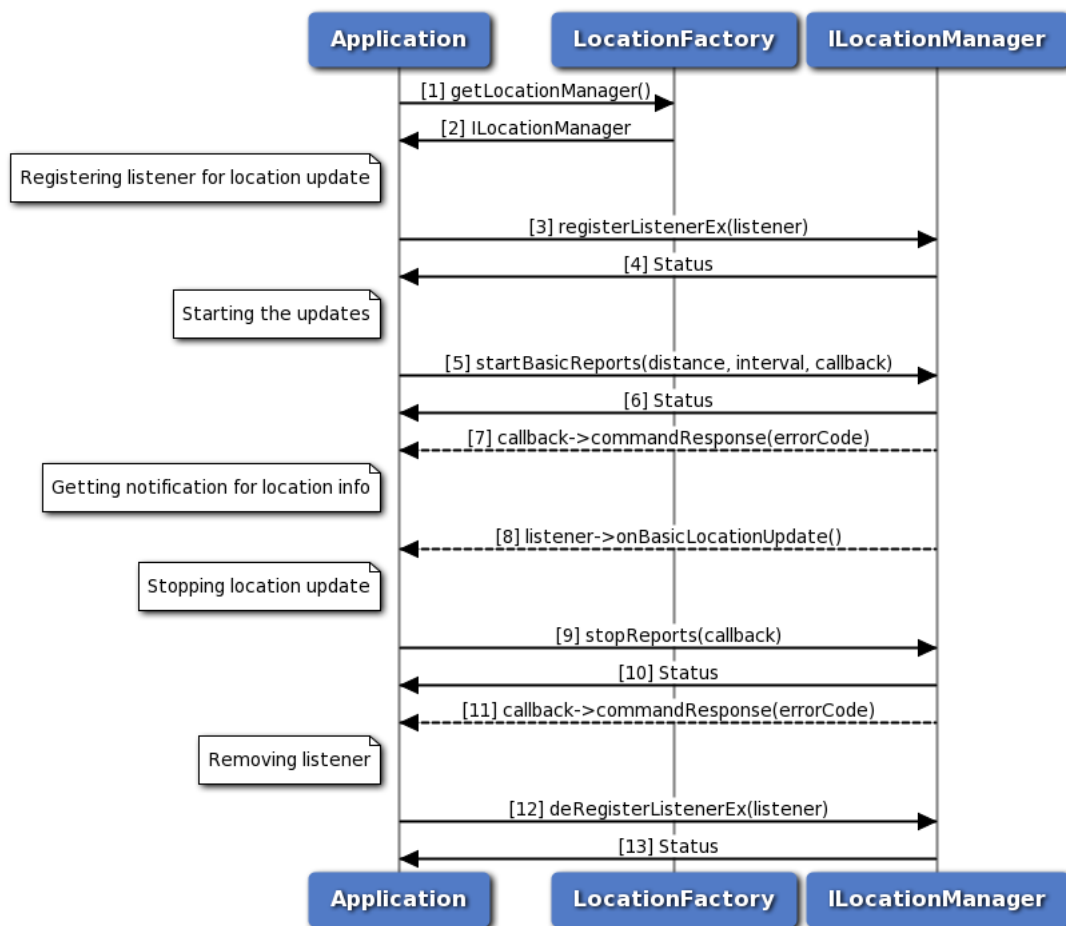


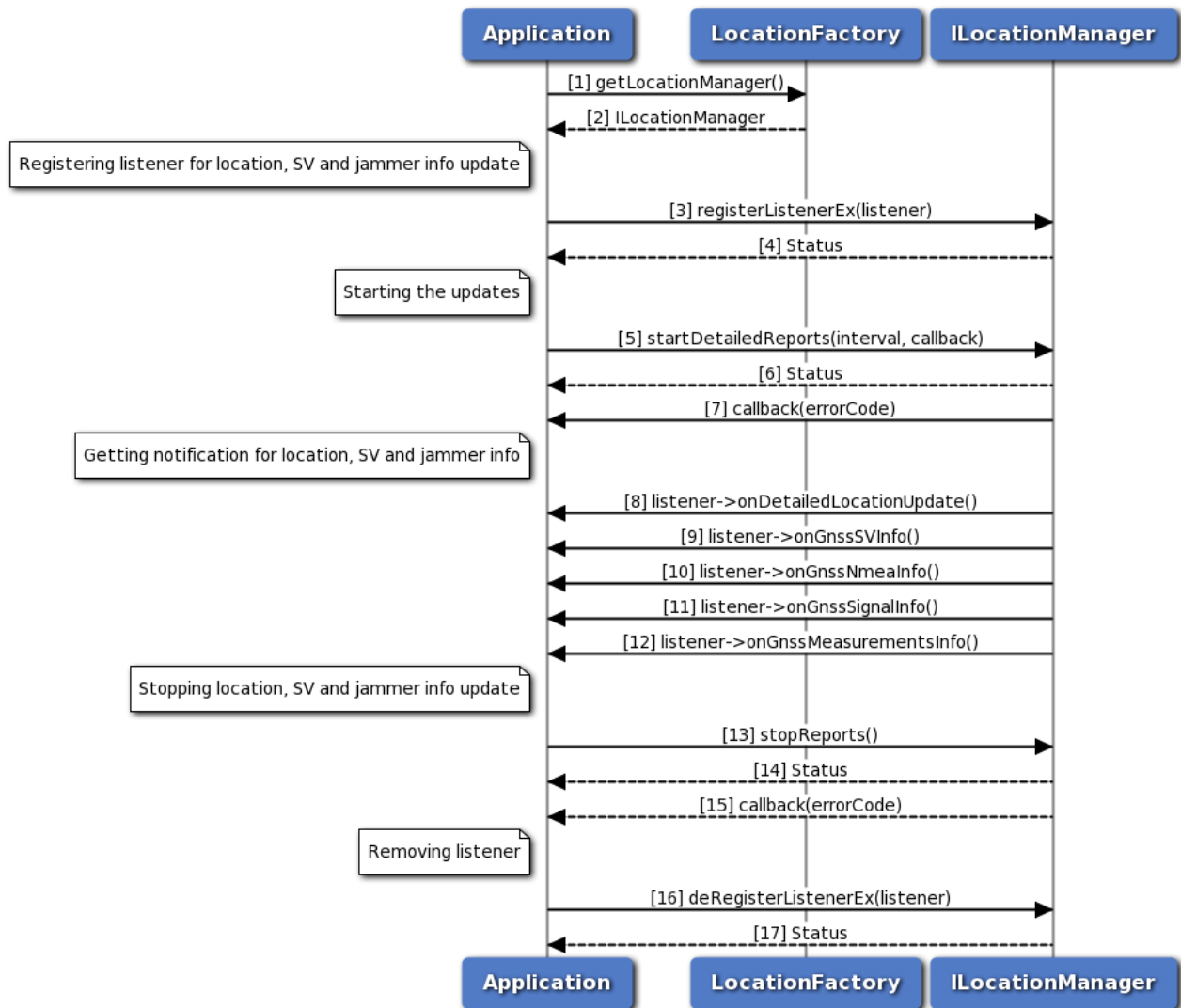
Figure 3-19 Call flow to register/remove listener for generating basic reports

1. Application requests location factory for location manager object.
2. Location factory returns ILocationManager object using which application will register or remove a listener.



3. Application can register a listener for getting notifications for location updates.
4. Status of register listener i.e. either SUCCESS or FAILED will be returned to the application.
5. Application starts the basic reports using startBasicReports API for getting location updates.
6. Status of startBasicReports i.e. either SUCCESS or FAILED will be returned to the application.
7. The response for startBasicReports is received by the application.
8. Application will get location updates like latitude, longitude and altitude etc.
9. Application stops receiving the report through stopReports API.
10. Status of stopReports i.e. either SUCCESS or FAILED will be returned to the application.
11. The response for stopReports is received by the application.
12. Application can remove listener and when the number of listeners are zero then location service will get stopped automatically.
13. Status of remove listener i.e. either SUCCESS or FAILED will be returned to the application.

### 3.4.2 Call flow to register/remove listener for generating detailed reports

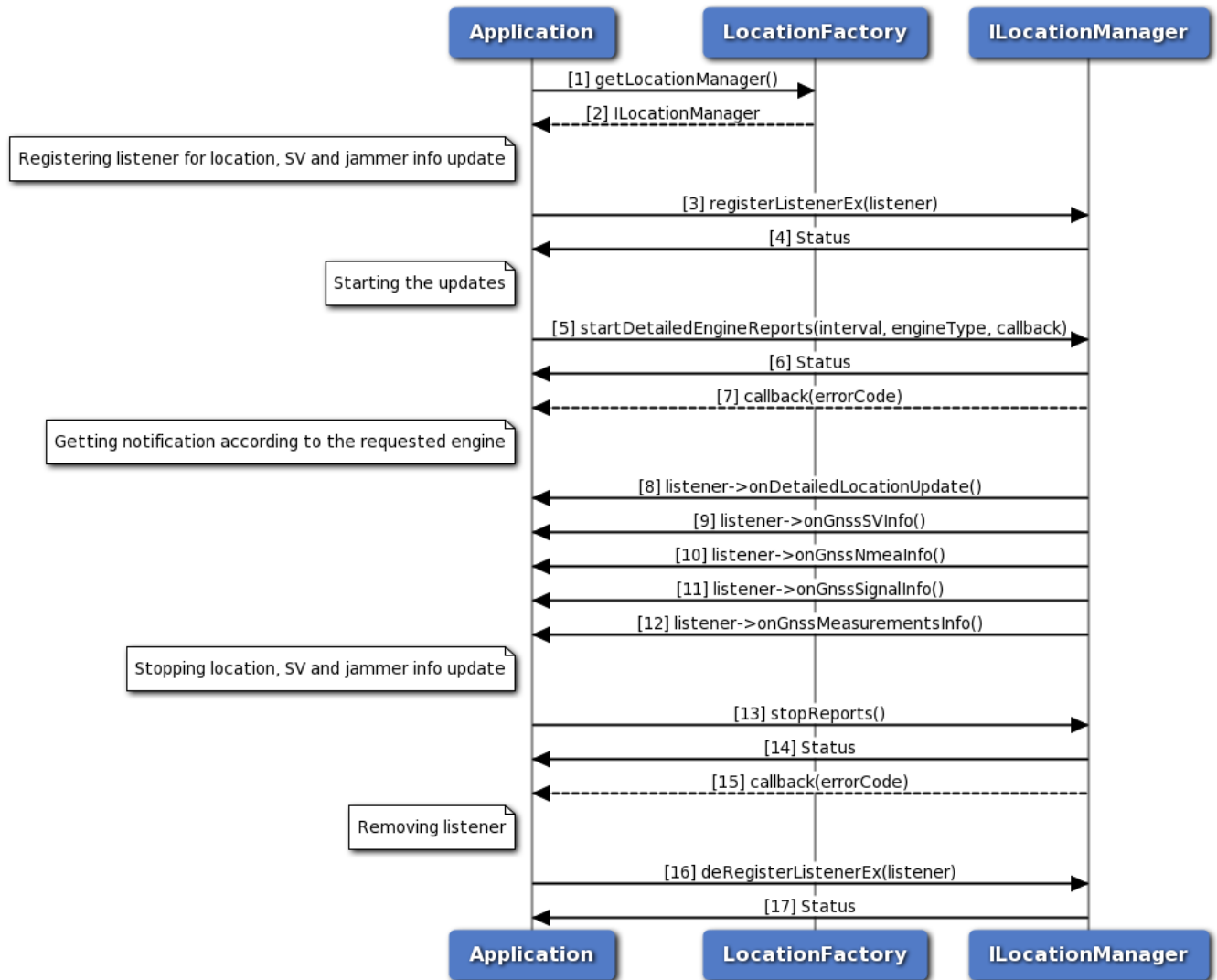


**Figure 3-20 Call flow to register/remove listener for generating detailed reports**

1. Application requests location factory for location manager object.
2. Location factory returns ILocationManager object using which application will register or remove a listener.
3. Application can register a listener for getting notifications for location, satellite vehicle, jammer signal, nmea and measurements updates.
4. Status of register listener i.e. either SUCCESS or FAILED will be returned to the application.
5. Application starts the detailed reports using startDetailedReports API for getting location, satellite vehicle, jammer signal, nmea and measurements updates.
6. Status of startDetailedReports i.e. either SUCCESS or FAILED will be returned to the application.
7. The response for startDetailedReports is received by the application.

8. Application will get location updates like latitude, longitude and altitude etc.
9. Application will receive satellite vehicle information like SV status and constellation etc.
10. Application will receive nmea information.
11. Application will receive jammer information etc.
12. Application will receive measurement information.
13. Application stops receiving all the reports through stopReports API.
14. Status of stopReports i.e. either SUCCESS or FAILED will be returned to the application.
15. The response for stopReports is received by the application.
16. Application can remove listener and when the number of listeners are zero then location service will get stopped automatically.
17. Status of remove listener i.e. either SUCCESS or FAILED will be returned to the application.

### 3.4.3 Call flow to register/remove listener for generating detailed engine reports



**Figure 3-21 Call flow to register/remove listener for generating detailed engine reports**

1. Application requests location factory for location manager object.
2. Location factory returns ILocationManager object using which application will register or remove a listener.
3. Application can register a listener for getting notifications for location, satellite vehicle and jammer signal, nmea and measurements updates.
4. Status of register listener i.e. either SUCCESS or FAILED will be returned to the application.
5. Application starts the detailed engine reports using startDetailedEngineReports API for getting location, satellite vehicle, jammer signal, nmea and measurements updates.
6. Status of startDetailedReports i.e. either SUCCESS or FAILED will be returned to the application.

7. The response for startDetailedReports is received by the application.
8. Application will get location updates like latitude, longitude and altitude etc from the requested engine type(SPE/PPE/Fused).
9. Application will receive satellite vehicle information like SV status and constellation etc depending on the requested SPE/PPE/Fused engine type.
10. Application will receive nmea information depending on the requested SPE/PPE/Fused engine type
11. Application will receive jammer information etc depending on the requested SPE/PPE/Fused engine type.
12. Application will receive measurement information etc depending on the requested SPE/PPE/Fused engine type.
13. Application stops receiving all the reports through stopReports API.
14. Status of stopReports i.e. either SUCCESS or FAILED will be returned to the application.
15. The response for stopReports is received by the application.
16. Application can remove listener and when the number of listeners are zero then location service will be stopped automatically.
17. Status of remove listener i.e. either SUCCESS or FAILED will be returned to the application.

### 3.4.4 Call flow to register/remove listener for system info updates

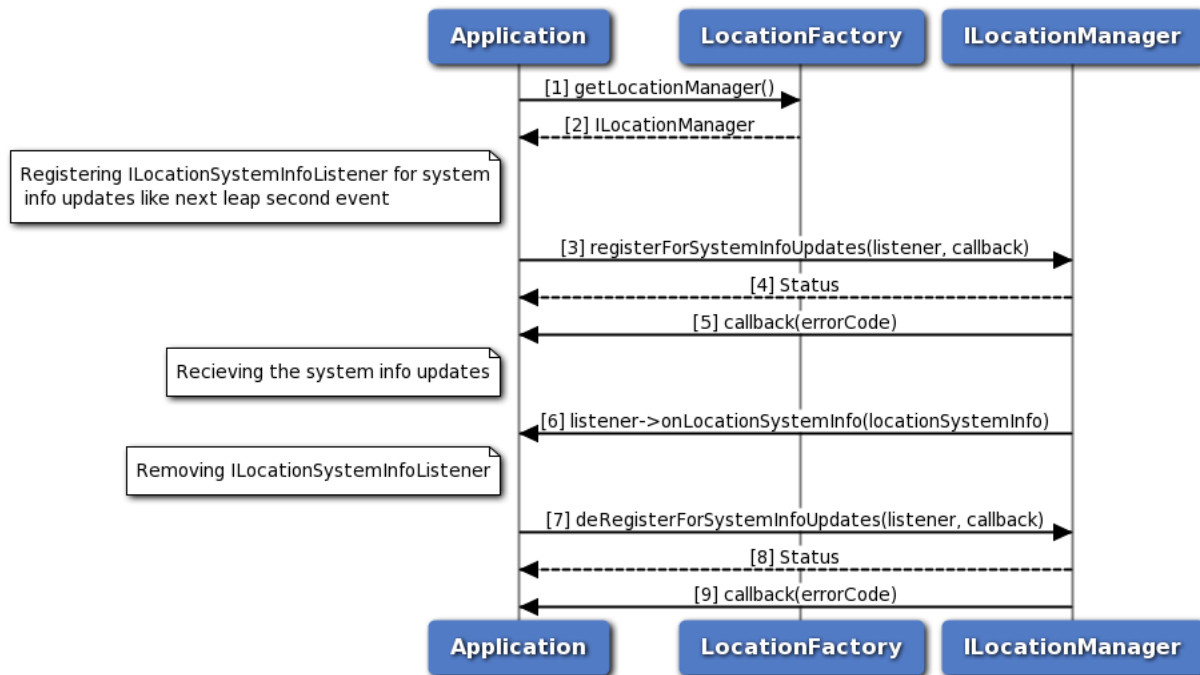
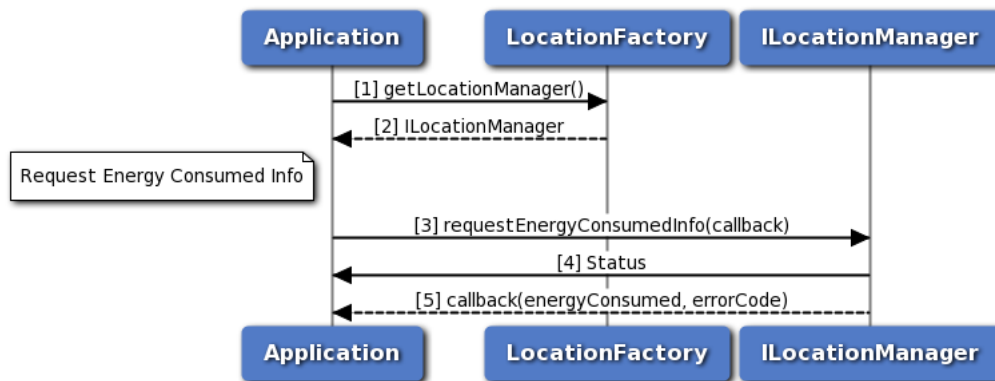


Figure 3-22 Call flow to register/remove listener for system info updates

1. Application requests location factory for location manager object.
2. Location factory returns ILocationManager object using which application will register or remove a listener.

3. Application can register a listener for system information updates with `registerForSystemInfoUpdates`.
4. Status of `registerForSystemInfoUpdates` i.e. either SUCCESS or FAILED will be returned to the application.
5. The response for `registerForSystemInfoUpdates` is received by the application.
6. Application will get system information update.
7. Application can remove listener with `deRegisterForSystemInfoUpdates`.
8. Status of `deRegisterForSystemInfoUpdates` i.e. either SUCCESS or FAILED will be returned to the application.
9. The response for `deRegisterForSystemInfoUpdates` is received by the application.

### 3.4.5 Call flow to request energy consumed information



**Figure 3-23 Call flow to request energy consumed information**

1. Application requests location factory for location manager object.
2. Location factory returns `ILocationManager` object.
3. Application can request for energy consumed information with `requestEnergyConsumedInfo`.
4. Status of `requestEnergyConsumedInfo` i.e. either SUCCESS or FAILED will be returned to the application.
5. The response for `requestEnergyConsumedInfo` is received by the application.

### 3.4.6 Call flow to get terrestrial positioning information

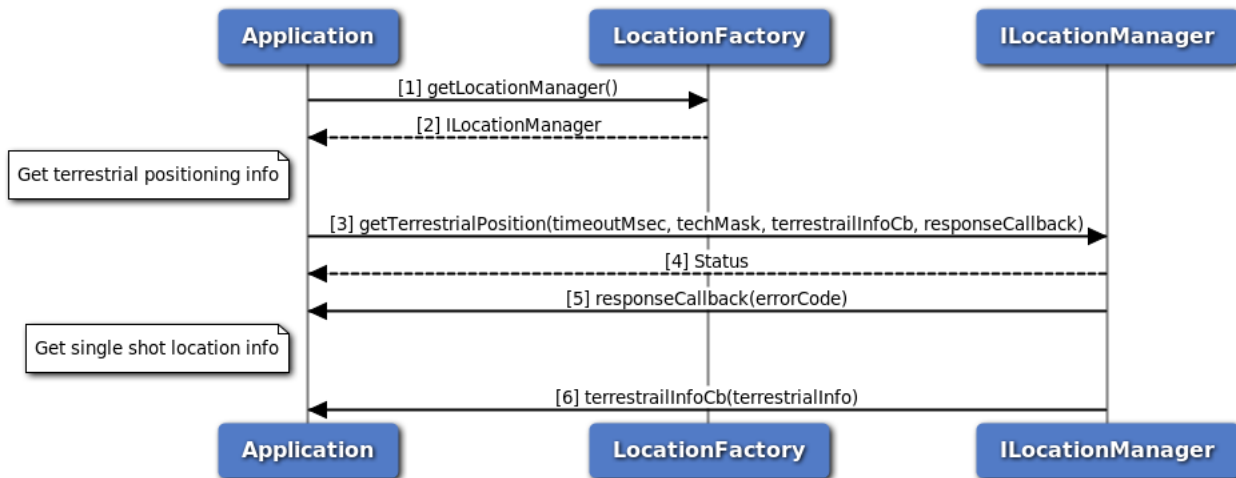


Figure 3-24 Call flow to get terrestrial positioning information

1. Application requests location factory for location manager object.
2. Location factory returns ILocationManager object.
3. Application can request for terrestrial positioning information with getTerrestrialPosition API.
4. Status of getTerrestrialPosition i.e. either SUCCESS or FAILED will be returned to the application.
5. The response for getTerrestrialPosition is received by the application.
6. Single shot terrestrial position information is received by the application.

### 3.4.7 Call flow to cancel terrestrial positioning information

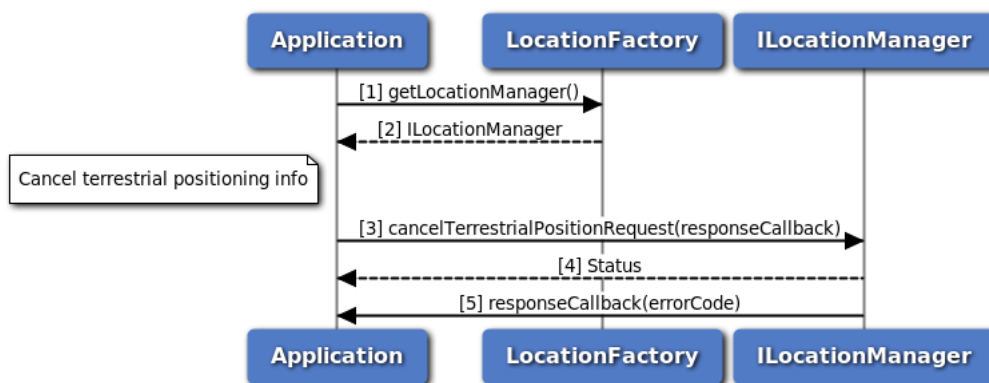


Figure 3-25 Call flow to cancel terrestrial positioning information

1. Application requests location factory for location manager object.
2. Location factory returns ILocationManager object.

3. Application can cancel request for terrestrial positioning information with `cancelTerrestrialPositionRequest` API.
4. Status of `cancelTerrestrialPositionRequest` i.e. either SUCCESS or FAILED will be returned to the application.
5. The response for `cancelTerrestrialPositionRequest` is received by the application.

### 3.4.8 Call flow to enable/disable constraint time uncertainty

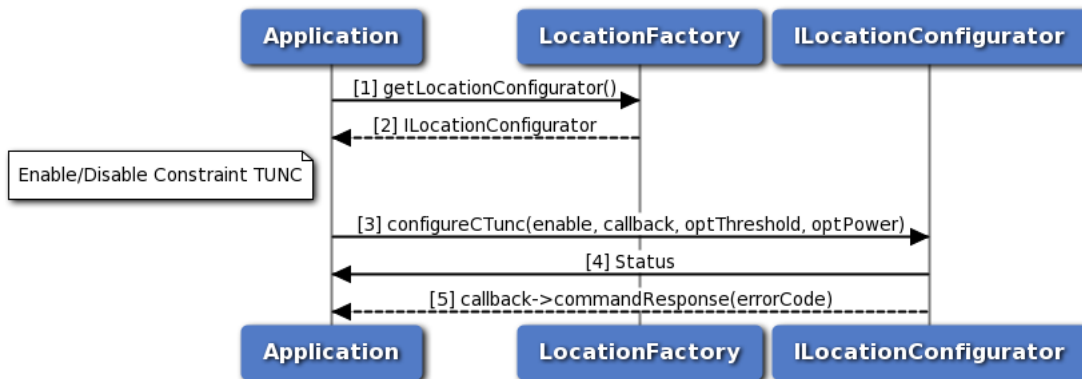


Figure 3-26 Call flow to enable/disable constraint time uncertainty

1. Application requests location factory for location configurator object.
2. Location factory returns `ILocationConfigurator` object.
3. Application enables/disables constraint tunc using `configureCTunc` API.
4. Status of `configureCTunc` i.e. either SUCCESS or FAILED will be returned to the application.
5. The response for `configureCTunc` is received by the application.

### 3.4.9 Call flow to enable/disable PACE

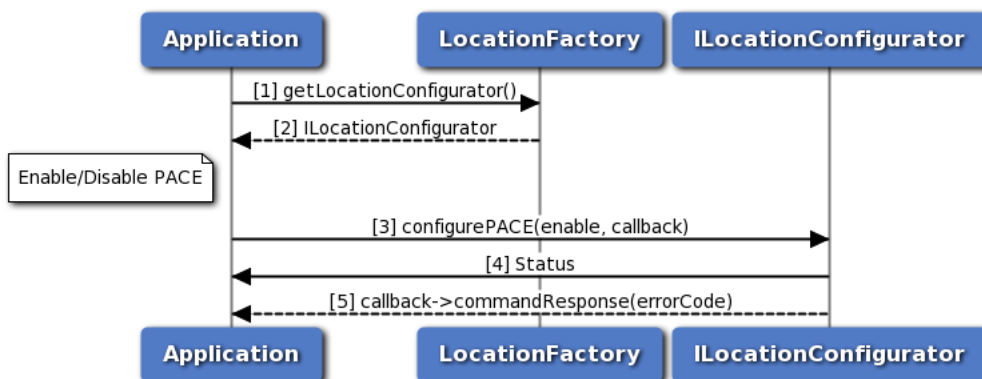


Figure 3-27 Call flow to enable/disable PACE

1. Application requests location factory for location configurator object.



2. Location factory returns ILocationConfigurator object.
3. Application enables/disables PACE using configurePACE API.
4. Status of configurePACE i.e. either SUCCESS or FAILED will be returned to the application.
5. The response for configurePACE is received by the application.

### 3.4.10 Call flow to delete all aiding data

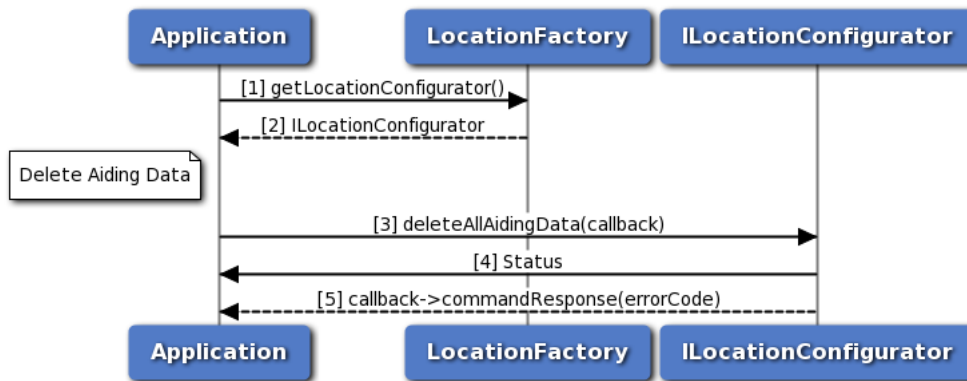


Figure 3-28 Call flow to delete all aiding data

1. Application requests location factory for location configurator object.
2. Location factory returns ILocationConfigurator object.
3. Application deletes all aiding data using deleteAllAidingData API.
4. Status of deleteAllAidingData i.e. either SUCCESS or FAILED will be returned to the application.
5. The response for deleteAllAidingData is received by the application.

### 3.4.11 Call flow to configure lever arm parameters

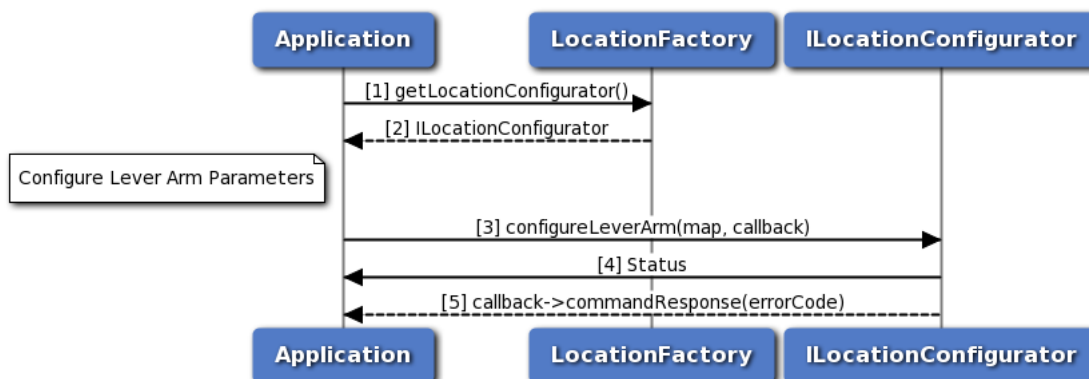


Figure 3-29 Call flow to configure lever arm parameters

1. Application requests location factory for location configurator object.
2. Location factory returns ILocationConfigurator object.

3. Application configures lever arm parameters using configureLeverArm API.
4. Status of configureLeverArm i.e. either SUCCESS or FAILED will be returned to the application.
5. The response for configureLeverArm is received by the application.

### 3.4.12 Call flow to configure blacklisted constellations

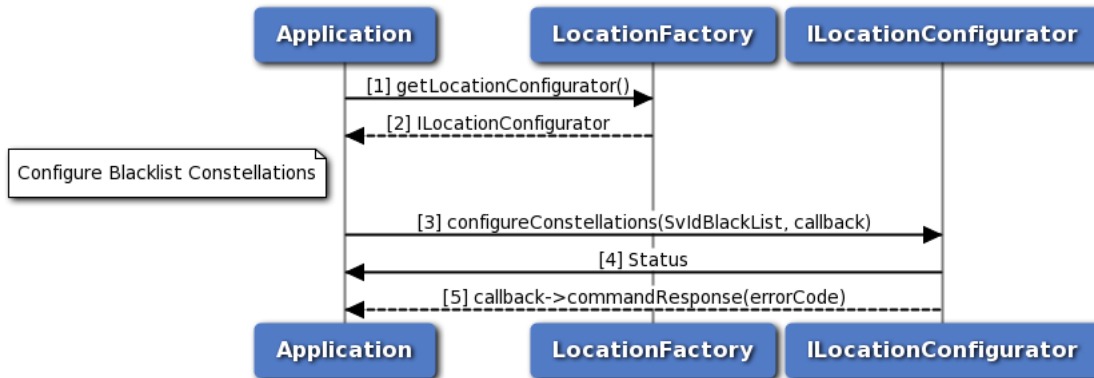


Figure 3-30 Call flow to configure blacklisted constellations

1. Application requests location factory for location configurator object.
2. Location factory returns ILocationConfigurator object.
3. Application configures blacklisted constellations using configureConstellations API.
4. Status of configureConstellations i.e. either SUCCESS or FAILED will be returned to the application.
5. The response for configureConstellations is received by the application.

### 3.4.13 Call flow to configure robust location

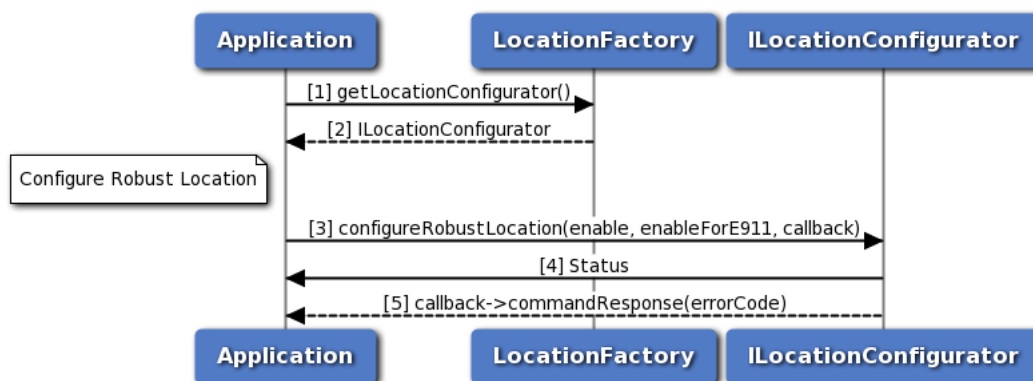


Figure 3-31 Call flow to configure robust location

1. Application requests location factory for location configurator object.
2. Location factory returns ILocationConfigurator object.
3. Application configures robust location using configureRobustLocation API.

4. Status of configureRobustLocation i.e. either SUCCESS or FAILED will be returned to the application.
5. The response for configureRobustLocation is received by the application.

### 3.4.14 Call flow to configure min gps week

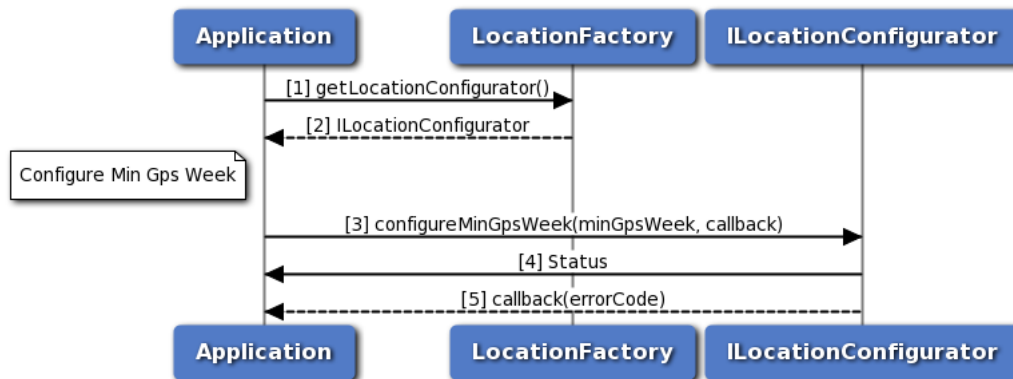


Figure 3-32 Call flow to configure min gps week

1. Application requests location factory for location configurator object.
2. Location factory returns ILocationConfigurator object.
3. Application configures min gps week using configureMinGpsWeek API.
4. Status of configureMinGpsWeek i.e. either SUCCESS or FAILED will be returned to the application.
5. The response for configureMinGpsWeek is received by the application.

### 3.4.15 Call flow to request min gps week

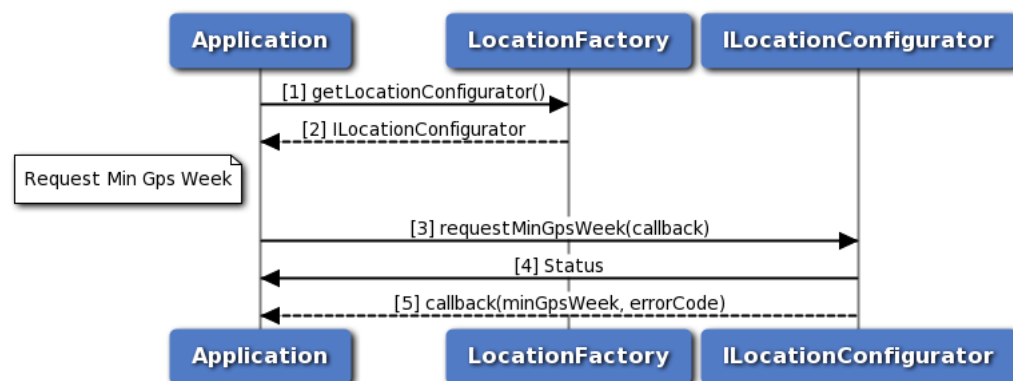


Figure 3-33 Call flow to request min gps week

1. Application requests location factory for location configurator object.
2. Location factory returns ILocationConfigurator object.
3. Application requests min gps week using requestMinGpsWeek API.

4. Status of requestMinGpsWeek i.e. either SUCCESS or FAILED will be returned to the application.
5. The response for requestMinGpsWeek is received by the application.

### 3.4.16 Call flow to delete specified data

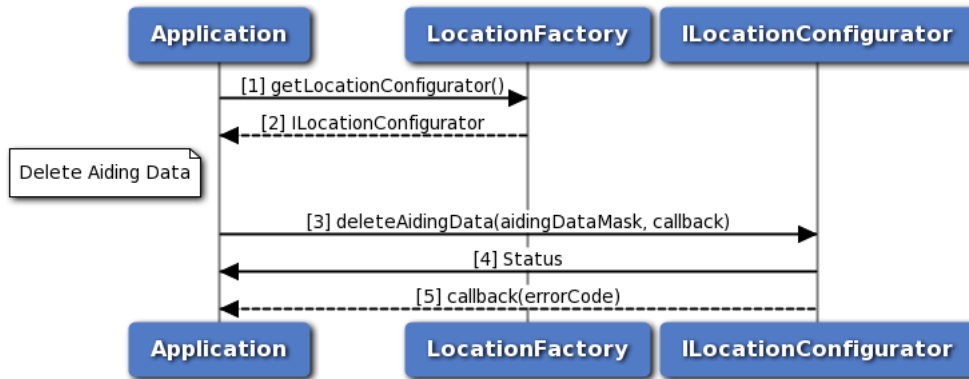


Figure 3-34 Call flow to delete specified data

1. Application requests location factory for location configurator object.
2. Location factory returns ILocationConfigurator object.
3. Application requests delete specified data using deleteAidingData API.
4. Status of deleteAidingData i.e. either SUCCESS or FAILED will be returned to the application.
5. The response for deleteAidingData is received by the application.

### 3.4.17 Call flow to configure min sv elevation

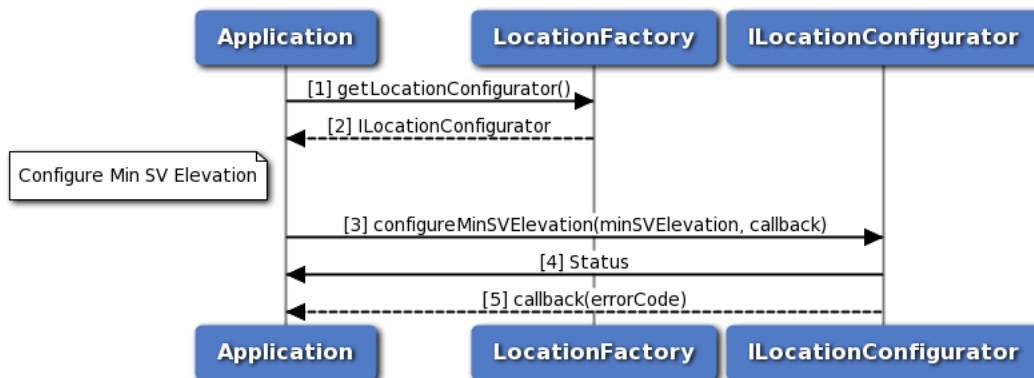


Figure 3-35 Call flow to configure min sv elevation

1. Application requests location factory for location configurator object.
2. Location factory returns ILocationConfigurator object.
3. Application configures min SV elevation using configureMinSVElevation API.

4. Status of configureMinSVElevation i.e. either SUCCESS or FAILED will be returned to the application.
5. The response for configureMinSVElevation is received by the application.

### 3.4.18 Call flow to request min sv elevation

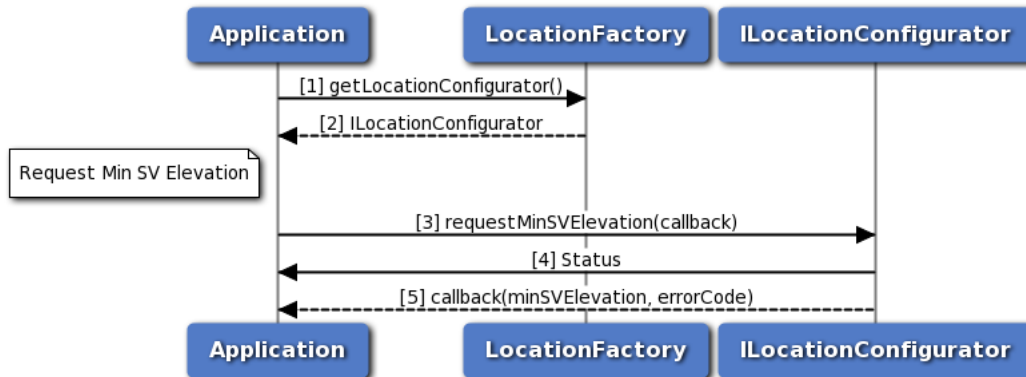


Figure 3-36 Call flow to request min sv elevation

1. Application requests location factory for location configurator object.
2. Location factory returns ILocationConfigurator object.
3. Application requests min SV elevation using requestMinSVElevation API.
4. Status of requestMinSVElevation i.e. either SUCCESS or FAILED will be returned to the application.
5. The response for requestMinSVElevation is received by the application.

### 3.4.19 Call flow to request robust location

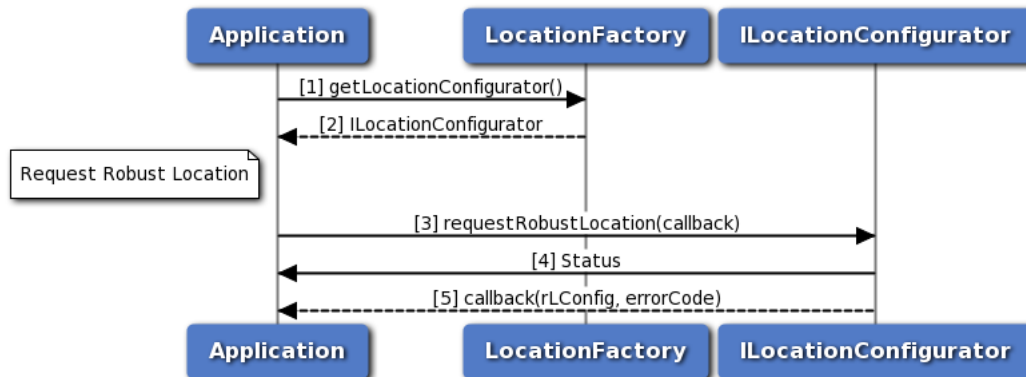


Figure 3-37 Call flow to request robust location

1. Application requests location factory for location configurator object.
2. Location factory returns ILocationConfigurator object.

3. Application requests robust location using requestRobustLocation API.
4. Status of requestRobustLocation i.e. either SUCCESS or FAILED will be returned to the application.
5. The response for requestRobustLocation is received by the application.

### 3.4.20 Call flow to configure dead reckoning engine

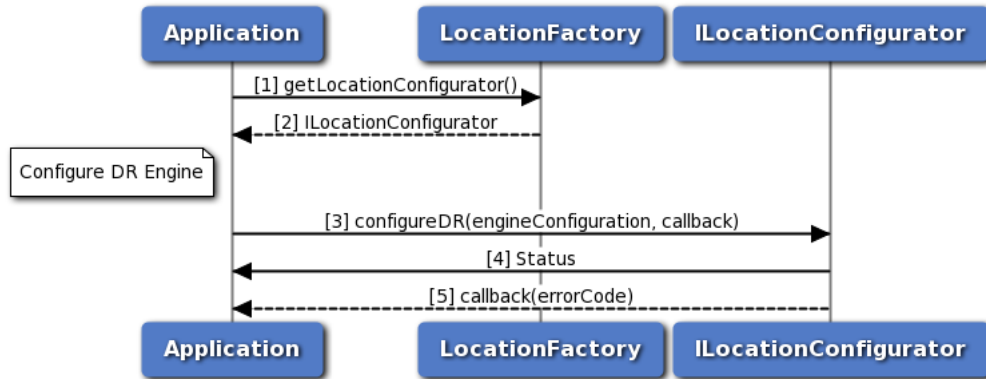


Figure 3-38 Call flow to configure dead reckoning engine

1. Application requests location factory for location configurator object.
2. Location factory returns ILocationConfigurator object.
3. Application configures dead reckoning engine using configureDR API.
4. Status of configureDR i.e. either SUCCESS or FAILED will be returned to the application.
5. The response for configureDR is received by the application.

### 3.4.21 Call flow to provide user consent for terrestrial positioning

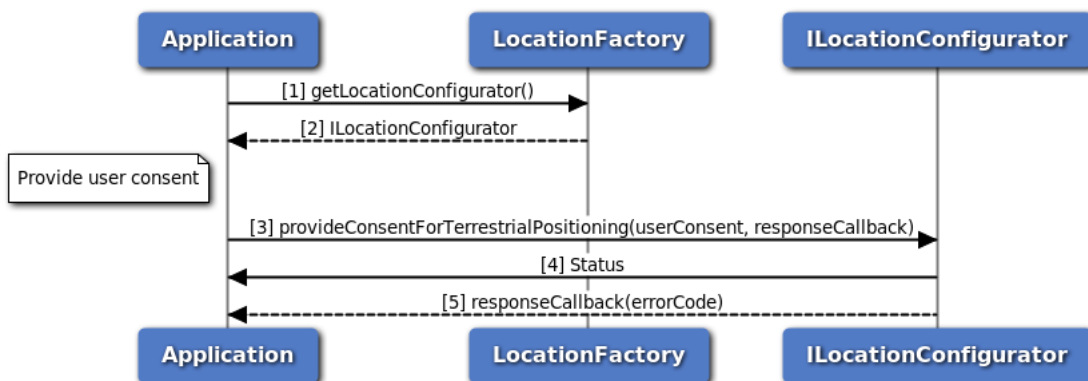


Figure 3-39 Call flow to provide user consent for terrestrial positioning

1. Application requests location factory for location configurator object.
2. Location factory returns ILocationConfigurator object.

3. Application provides user consent for terrestrial positioning using provideConsentForTerrestrialPositioning API.
4. Status of provideConsentForTerrestrialPositioning i.e. either SUCCESS or FAILED will be returned to the application.
5. The response for provideConsentForTerrestrialPositioning is received by the application.

### 3.4.22 Call flow to represent Xtra Feature

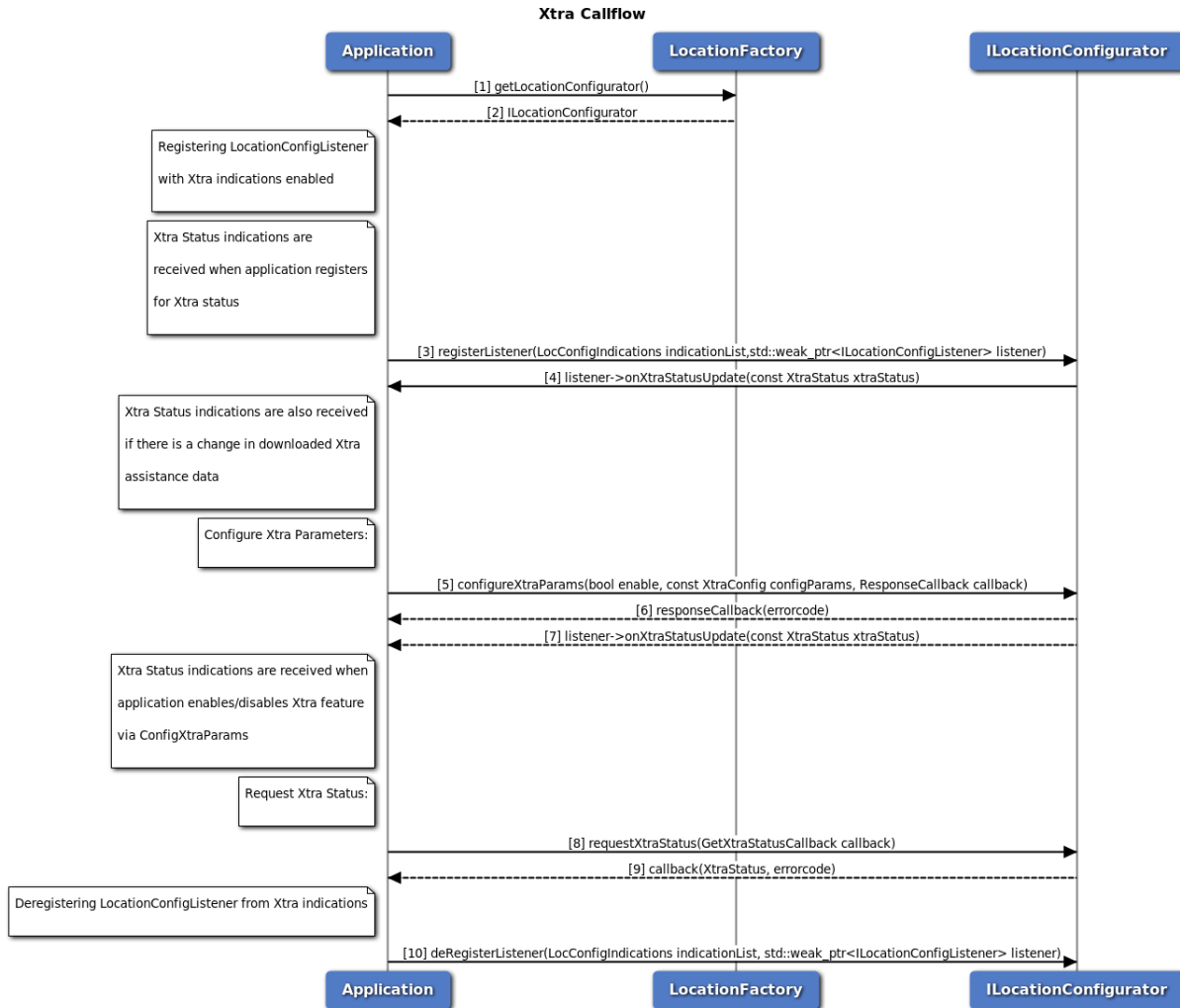


Figure 3-40 Call flow to represent Xtra Feature

1. Application requests location factory for location configurator object.
2. Location factory returns ILocationConfigurator object.
3. Application registers config listener for Xtra Status indications.
4. Listener API onXtraStatusUpdate is invoked while registering for Xtra indications.
5. Application configures Xtra Params using configureXtraParams.
6. Location Configurator invokes response callback with errorcode.

7. Listener API on XtraStatusUpdate is invoked when Xtra feature is enabled/disabled.
8. Application requests Xtra Status using requestXtraStatus.
9. Location Configurator invokes GetXtraStatus callback with XtraStatus and errorcode.
10. Application deregisters config listener from Xtra Status indications.

### 3.5 Data

#### 3.5.1 Start/Stop for data connection manager call flow

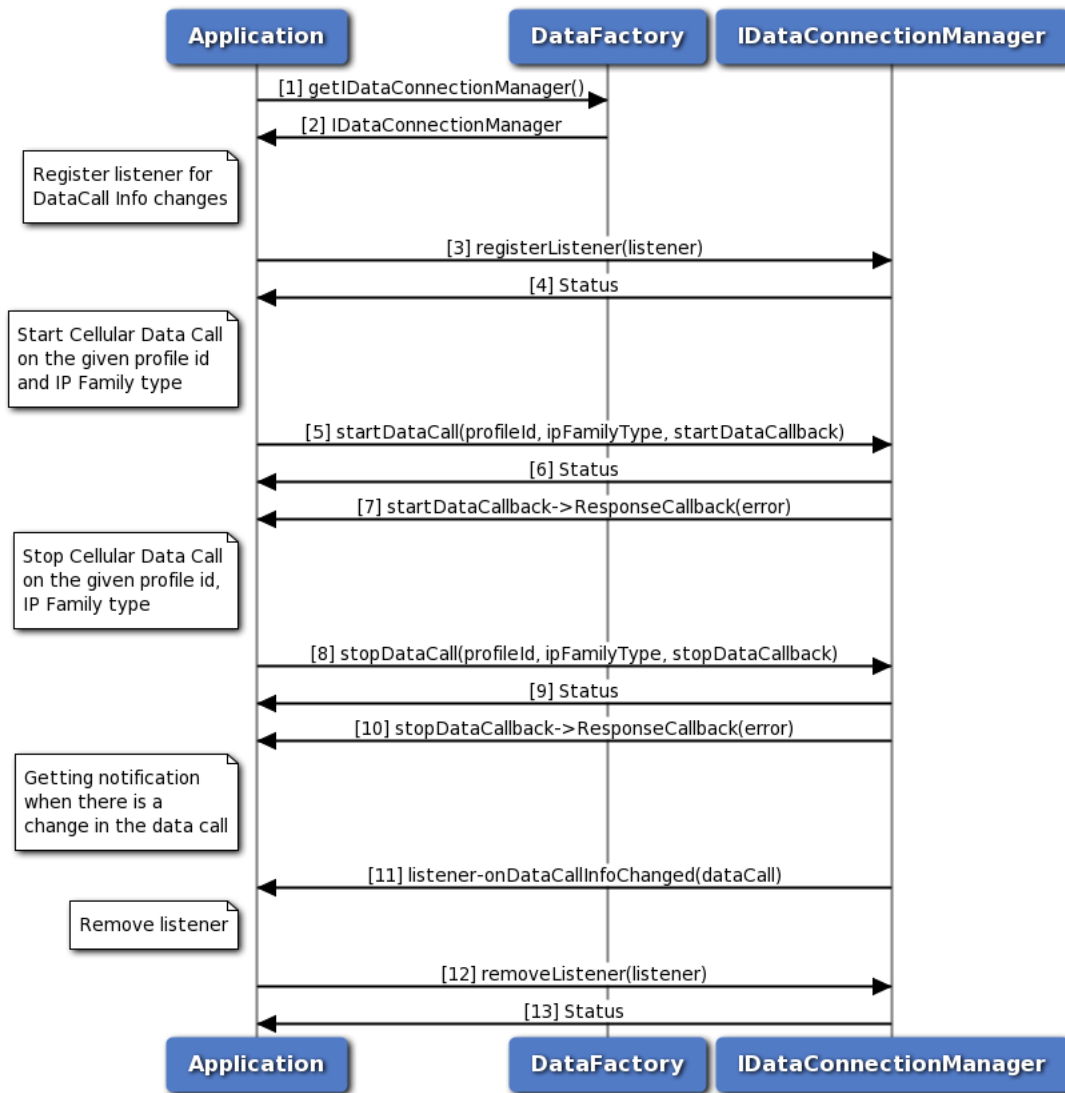


Figure 3-41 Start/Stop for data connection manager call flow

1. Application requests data factory for data connection manager object.
2. Data factory returns IDataConnectionManager object to application.
3. Application registers the listener to get notifications for data call change.



4. The application receives the status i.e. either SUCCESS or FAILED based on the registration of the listener.
5. Application requests for start data call and optionally gets asynchronous response using startDataCallback.
6. Application receives the status i.e. either SUCCESS or FAILED based on the execution of startDataCall.
7. Optionally, the application gets asynchronous response for startDataCall using startDataCallback.
8. Application requests for stop data call and optionally gets asynchronous response using stopDataCallback.
9. Application receives the status i.e. either SUCCESS or FAILED based on the execution of stopDataCall.
10. Optionally, the application gets asynchronous response for stopDataCall using stopDataCallback.
11. Application receives a notification when there is a change in data call.
12. Application removes the listener.
13. Application receives the status i.e. SUCCESS or FAILED for the removal of listener.

### 3.5.2 Request/Create/Delete/Modify for data profile manager call flow

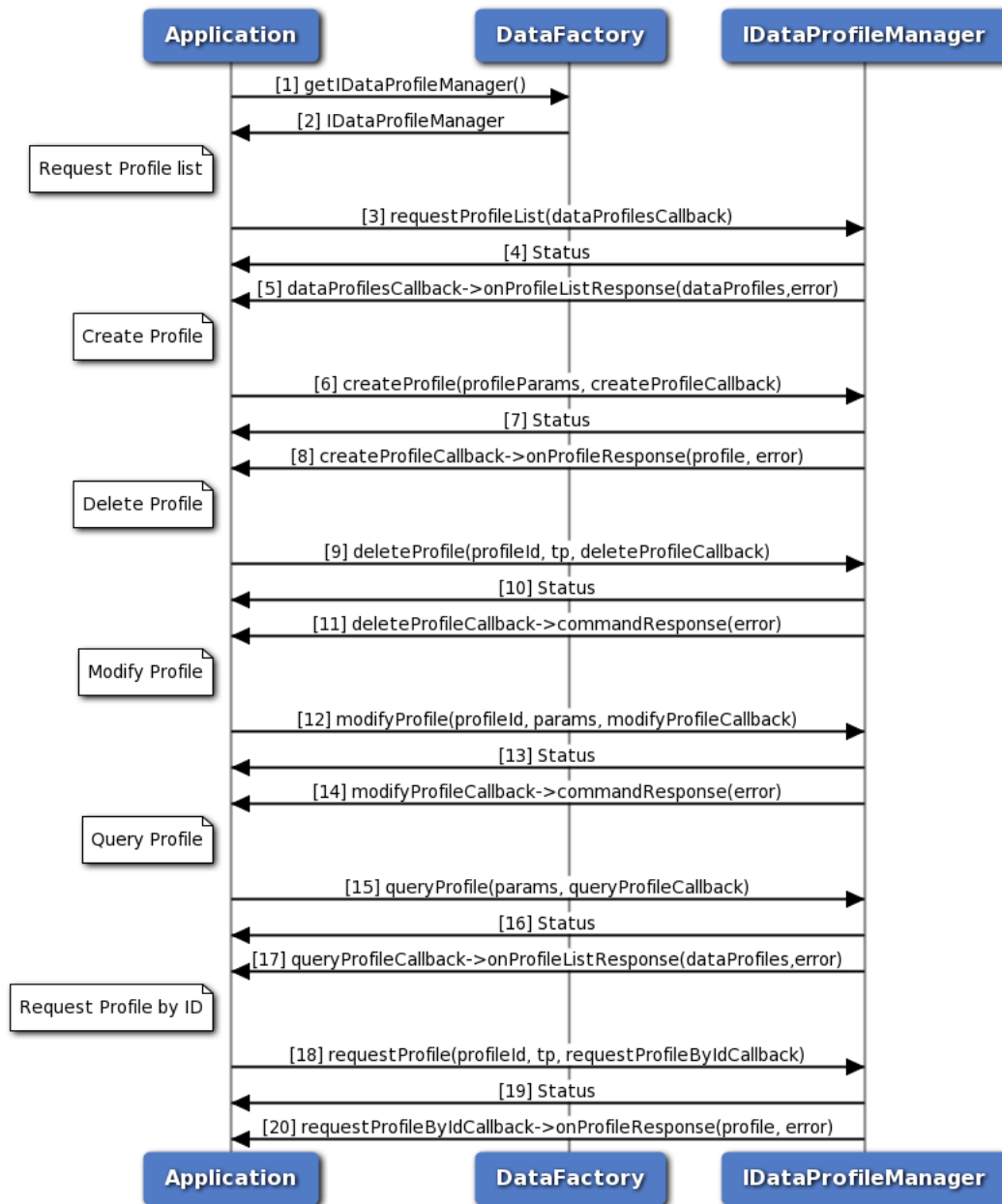


Figure 3-42 Request/Create/Delete/Modify for data profile manager call flow

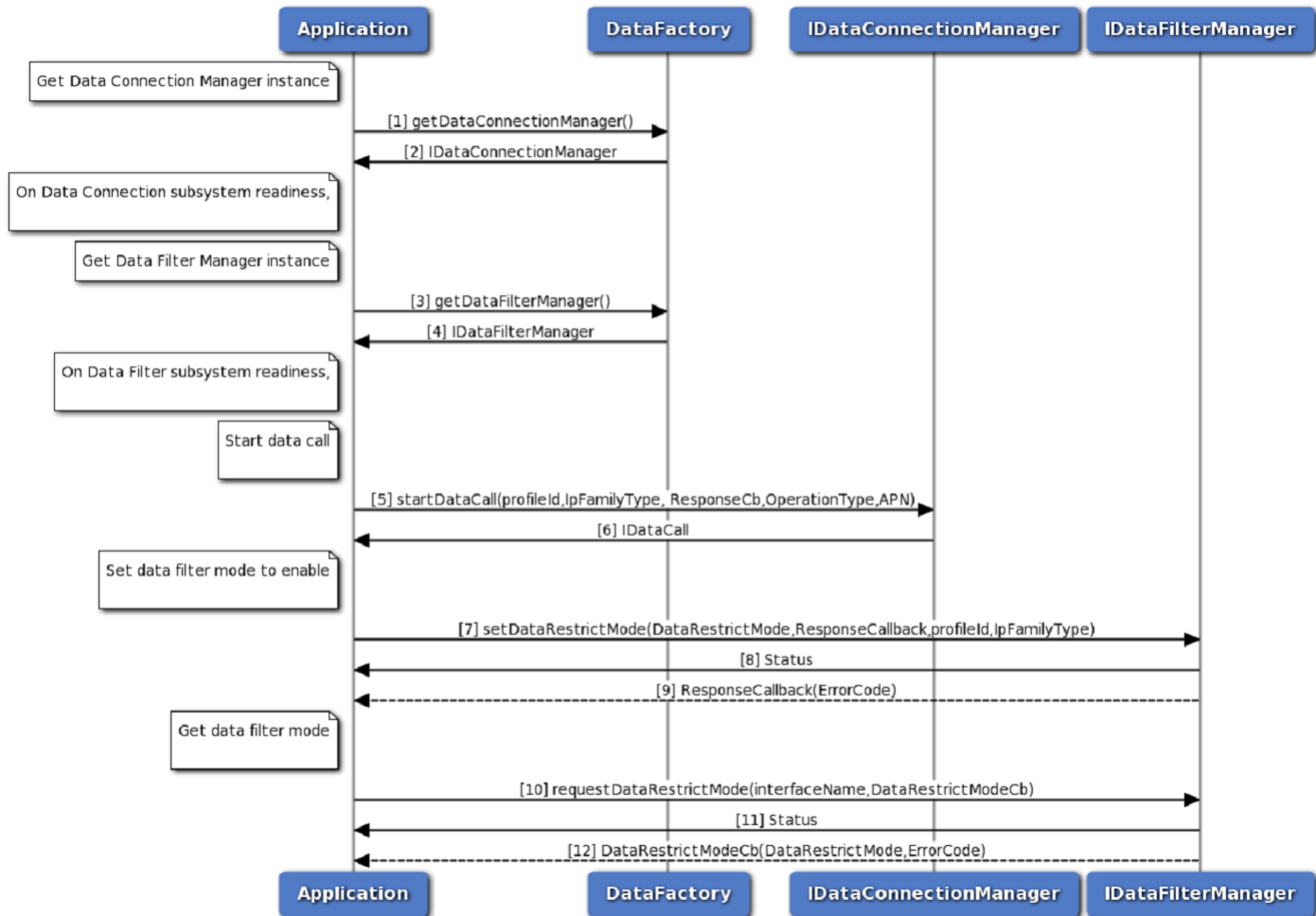
1. Application requests data factory for data profile manager object.
2. Data factory returns IDataProfileManager object to application.
3. Application requests for profile list and optionally gets asynchronous response using dataProfilesCallback.
4. Application receives the status i.e. either SUCCESS or FAILED based on the execution of requestProfileList.
5. Optionally, the application gets asynchronous response for requestProfileList using

- dataProfilesCallback.
6. Application requests for create profile and optionally gets asynchronous response using createProfilesCallback.
  7. Application receives the status i.e. either SUCCESS or FAILED based on the execution of createProfile.
  8. Optionally, the application gets asynchronous response for createProfile using createProfilesCallback.
  9. Application requests for delete profile and optionally gets asynchronous response using deleteProfilesCallback.
  10. Application receives the status i.e. either SUCCESS or FAILED based on the execution of deleteProfile.
  11. Optionally, the application gets asynchronous response for deleteProfile using deleteProfilesCallback.
  12. Application requests for modify profile and optionally gets asynchronous response using modifyProfilesCallback.
  13. Application receives the status i.e. either SUCCESS or FAILED based on the execution of modifyProfile.
  14. Optionally, the application gets asynchronous response for modifyProfile using modifyProfilesCallback.
  15. Application requests for query profile and optionally gets asynchronous response using queryProfilesCallback.
  16. Application receives the status i.e. either SUCCESS or FAILED based on the execution of queryProfile.
  17. Optionally, the application gets asynchronous response for queryProfile using queryProfilesCallback.
  18. Application requests for request profile and optionally gets asynchronous response using requestProfileByIdCallback.
  19. Application receives the status i.e. either SUCCESS or FAILED based on the execution of requestProfile.
  20. Optionally, the application gets asynchronous response for requestProfile using requestProfileByIdCallback.

### 3.5.3 Data Filter Manager Call Flow

Data Filter manager provides APIs to get/set data filter mode, add/remove data restrict filters. Its API can be used per data call or globally to apply the same changes to all the underlying currently up data call. It also has listener interface for notifications for data filter status update. Application will get the Data Filter manager object from data factory. The application can register a listener for data filter mode change updates.

### 3.5.3.1 Call flow to Set/Get data filter mode



**Figure 3-43 Get/Set data filter mode call flow**

1. Application requests data factory for data connection manager object.
2. Data factory returns IDataConnectionManager object to application.
3. Application requests data factory for data filter manager object.
4. Data factory returns IDataFilterManager object to application.
5. Application requests for start data call and optionally gets asynchronous response using startDataCallback.
6. Application receives the status i.e. either SUCCESS or FAILED based on the execution of startDataCall.
7. Optionally, the application gets asynchronous response for startDataCall using startDataCallback.
8. Application requests for set data filter mode to enable and optionally gets asynchronous response using ResponseCallback.

9. Application receives the status i.e. either SUCCESS or FAILED based on the execution of setDataRestrictMode.
10. Optionally, the application gets asynchronous response for setDataRestrictMode using ResponseCallback.
11. Application requests for get data filter mode and optionally gets asynchronous response using DataRestrictModeCb.
12. Application receives the status i.e. either SUCCESS or FAILED based on the execution of requestDataRestrictMode.
13. Optionally, the application gets asynchronous response for requestDataRestrictMode using DataRestrictModeCb.

### 3.5.3.2 Call flow to Add data restrict filter

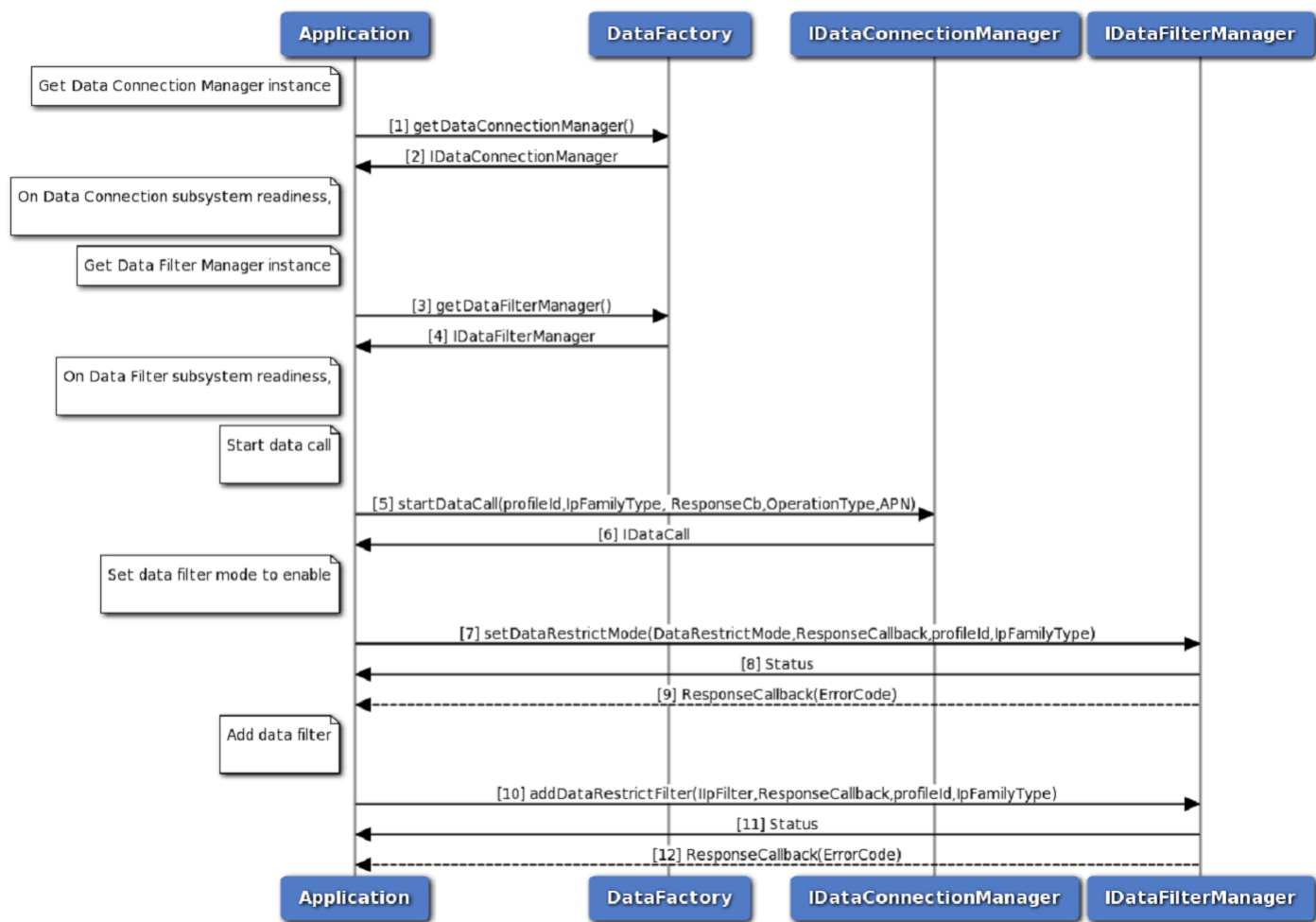
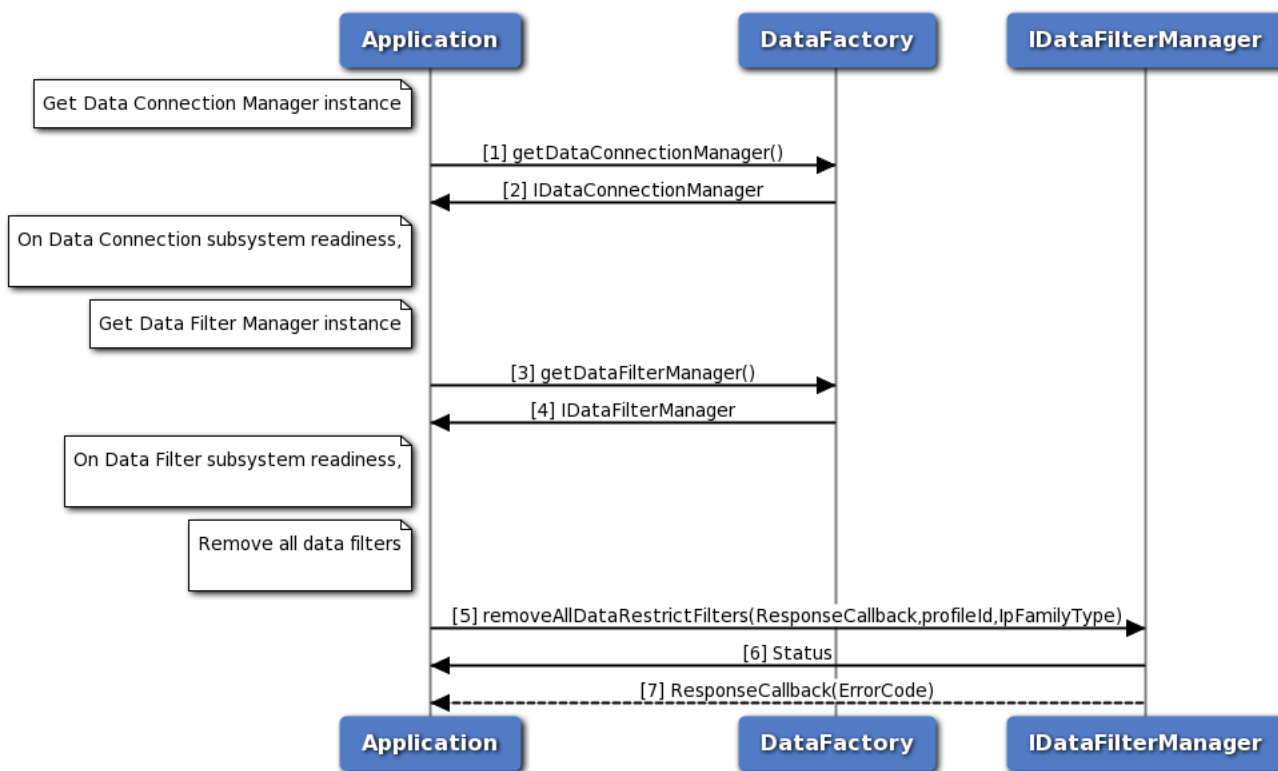


Figure 3-44 Add data restrict filter call flow

1. Application requests data factory for data connection manager object.
2. Data factory returns IDataConnectionManager object to application.

3. Application requests data factory for data filter manager object.
4. Data factory returns IDataFilterManager object to application.
5. Application requests for start data call and optionally gets asynchronous response using startDataCallback.
6. Application receives the status i.e. either SUCCESS or FAILED based on the execution of startDataCall.
7. Optionally, the application gets asynchronous response for startDataCall using startDataCallback.
8. Application requests for add data filter and optionally gets asynchronous response using ResponseCallback.
9. Application receives the status i.e. either SUCCESS or FAILED based on the execution of addDataRestrictFilter.
10. Optionally, the application gets asynchronous response for addDataRestrictFilter using ResponseCallback.

**3.5.3.3 Call flow to Remove data restrict filter**



**Figure 3-45 Remove data restrict filter call flow**

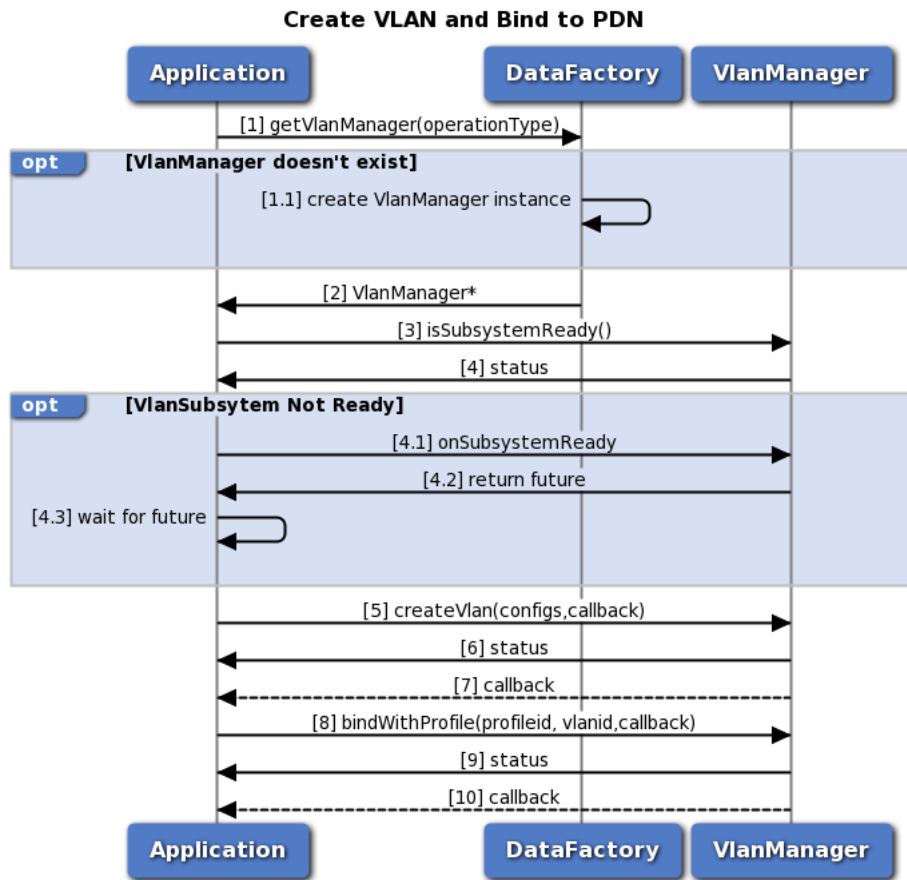
1. Application requests data factory for data connection manager object.
2. Data factory returns IDataConnectionManager object to application.
3. Application requests data factory for data filter manager object.

4. Data factory returns IDataFilterManager object to application.
5. Application requests for start data call and optionally gets asynchronous response using startDataCallback.
6. Application receives the status i.e. either SUCCESS or FAILED based on the execution of startDataCall.
7. Optionally, the application gets asynchronous response for startDataCall using startDataCallback.
8. Application requests for add data filter and optionally gets asynchronous response using ResponseCallback.
9. Application receives the status i.e. either SUCCESS or FAILED based on the execution of removeAllDataRestrictFilters.
10. Optionally, the application gets asynchronous response for removeAllDataRestrictFilters using ResponseCallback.

### 3.5.4 Data Networking Call Flow

Application will get the following manager objects from data factory to configure networking. IVlanManager is used to access all VLAN APIs. INatManager is used to access all Static NAT APIs. IFirewallManager is used to access all Firewall APIs.

### 3.5.4.1 Create VLAN and Bind it to PDN in data vlan manager call flow



**Figure 3-46 Create VLAN and bind it to PDN for data VLAN manager call flow**

1. Application requests data factory for data IVlanManager object. 1.1. If IVlanManager object does not exist, data factory will create new object.
2. Data factory returns IVlanManager object to application.
3. Application can use IVlanManager::isSubsystemReady to determine if the system is ready.
4. The application receives the Status i.e. either true or false to indicate whether sub-system is ready or not. 4.1. If it is not ready, then the application could use onSubsystemReady which returns std::future. 4.2. Application gets future for IVlanManager readiness. 4.3. Application wait for indefinitely for IVlanManager readiness.
5. On Readiness, application calls IVlanManager::createVlan with assigned id, interface, and acceleration type.
6. Application receives synchronous Status which indicates if the IVlanManager::createVlan request was sent successfully.
7. Application is notified of the Status of the IVlanManager::createVlan request (either SUCCESS or FAILED) via the application-supplied callback.
8. Application calls IVlanManager::bindWithProfile with Vlan id and profile id.
9. Application receives synchronous Status which indicates if the IVlanManager::bindWithProfile



request was sent successfully.

1. Application is notified of the Status of the IVlanManager::bindWithProfile request (either SUCCESS or FAILED) via the application-supplied callback.

### 3.5.4.2 LAN-LAN VLAN Configuration from EAP usecase call flow

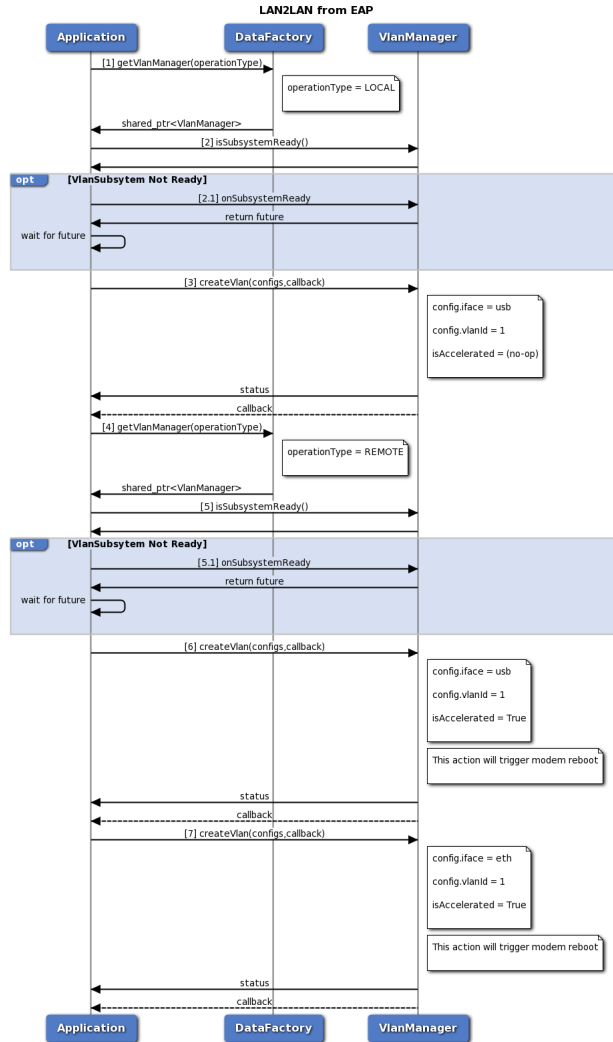


Figure 3-47 LAN-LAN VLAN Configuration Usecase from EAP call flow

1. Application requests data factory for data IVlanManager for local operation object.
2. Application use IVlanManager::isSubsystemReady to determine if the system is ready. 2.1. If it is not ready, then the application could use onSubsystemReady which returns std::future.
3. On readiness, application calls IVlanManager::createVlan with USB interface, Vlan id 1 and no acceleration and wait for callback to indicate createVlan result
4. Application requests data factory for data IVlanManager for remote operation object.
5. Application use IVlanManager::isSubsystemReady to determine if the system is ready. 5.1. If it is not ready, then the application could use onSubsystemReady which returns std::future.

6. On readiness, application calls IVlanManager::createVlan with USB interface, Vlan id 1 and acceleration and wait for callback to indicate createVlan result
7. Application calls IVlanManager::createVlan with ETH interface, Vlan id 1 and acceleration and wait for callback to indicate createVlan result

### 3.5.4.3 LAN-LAN VLAN Configuration from A7 usecase call flow

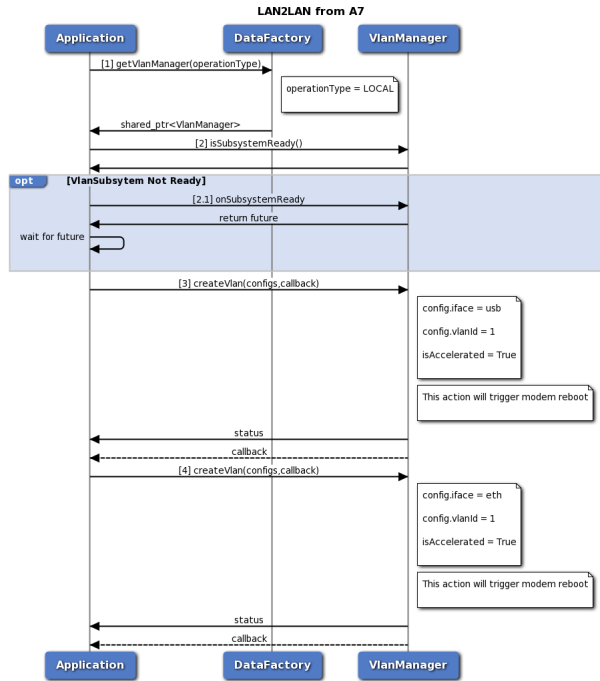


Figure 3-48 LAN-LAN VLAN Configuration Usecase from A7 call flow

1. Application requests data factory for data IVlanManager for local operation object.
2. Application use IVlanManager::isSubsystemReady to determine if the system is ready. 2.1. If it is not ready, then the application could use onSubsystemReady which returns std::future.
3. On readiness, application calls IVlanManager::createVlan with USB interface, Vlan id 1 and acceleration and wait for callback to indicate createVlan result
1. Application calls IVlanManager::createVlan with ETH interface, Vlan id 1 and acceleration and wait for callback to indicate createVlan result

### 3.5.4.4 LAN-WAN VLAN Configuration from EAP usecase call flow

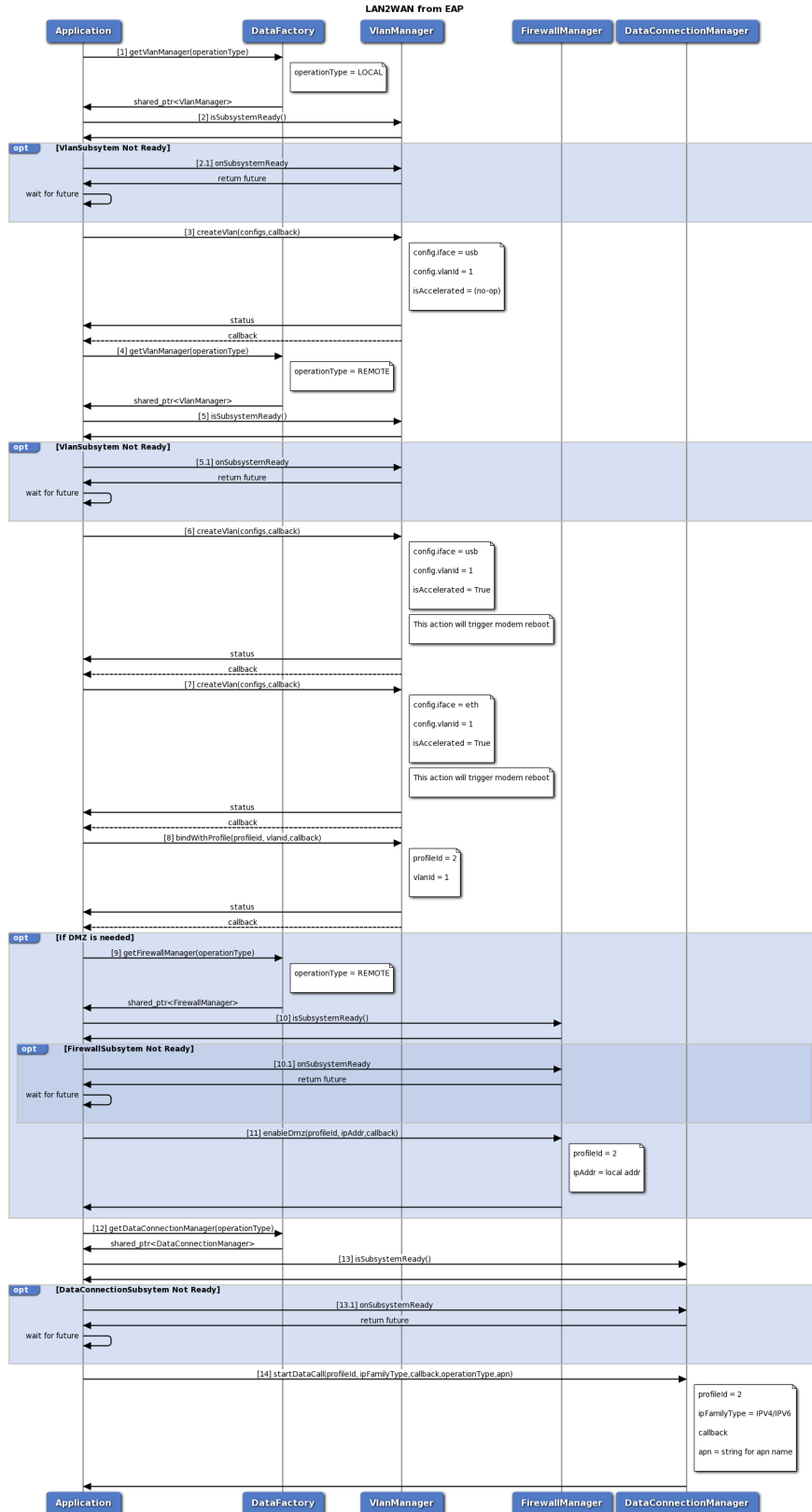


Figure 3-49 LAN-WAN VLAN Configuration Usecase from EAP call flow

1. Application requests data factory for data IVlanManager for local operation object.
2. Application use IVlanManager::isSubsystemReady to determine if the system is ready. 2.1. If it is not ready, then the application could use onSubsystemReady which returns std::future.
3. On readiness, application calls IVlanManager::createVlan with USB interface, Vlan id 1 and no acceleration and wait for callback to indicate createVlan result
4. Application requests data factory for data IVlanManager for remote operation object.
5. Application use IVlanManager::isSubsystemReady to determine if the system is ready. 5.1. If it is not ready, then the application could use onSubsystemReady which returns std::future.
6. On readiness, application calls IVlanManager::createVlan with USB interface, Vlan id 1 and acceleration and wait for callback to indicate createVlan result
7. Application calls IVlanManager::createVlan with ETH interface, Vlan id 1 and acceleration and wait for callback to indicate createVlan result
8. Application calls IVlanManager::bindWithProfile with profile id to bind with and wait for callback to indicate bindWithProfile result If DMZ is needed:
  - (a) Application requests data factory for data IFirewallManager object
  - (b) Application can use IFirewallManager::isSubsystemReady to determine if the system is ready. 10.1. If FirewallManager is not ready, then the application could use onSubsystemReady which returns std::future and wait for future.
  - (c) Application can use IFirewallManager::enableDmz with profile id and local address to be enable Dmz on
9. Application calls getDataConnectionManager to get object of IDataConnectionManager
10. Application can use IDataConnectionManager::isSubsystemReady to determine if the system is ready. 13.1. If DataConnectionManager is not ready, then the application could use onSubsystemReady which returns std::future and wait for future.
11. Application can use IDataConnectionManager::startDataCall with profile id to start data call on, Ip Family type, operation Type and APN Name

### 3.5.4.5 LAN-WAN VLAN Configuration from A7 usecase call flow

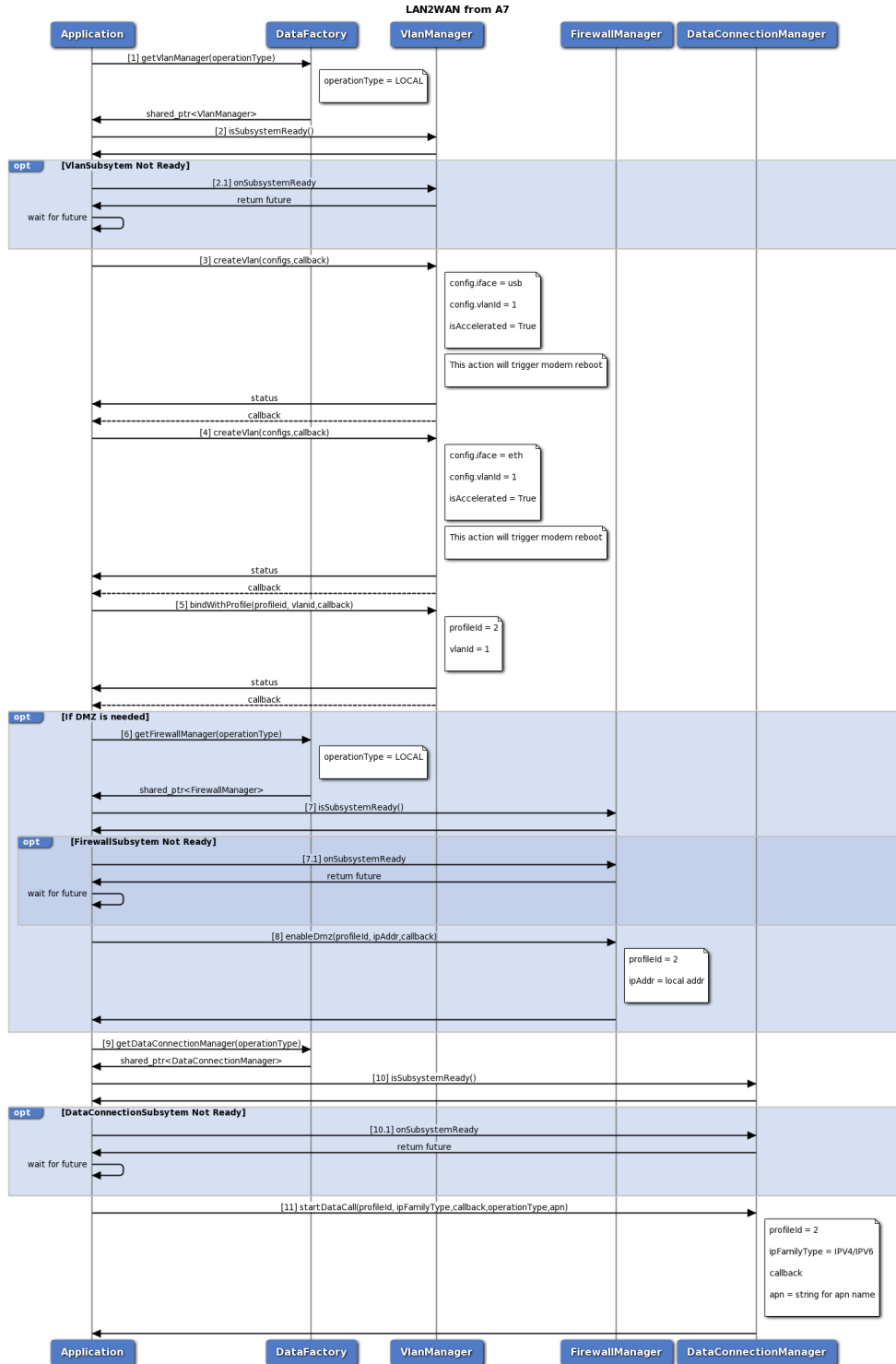
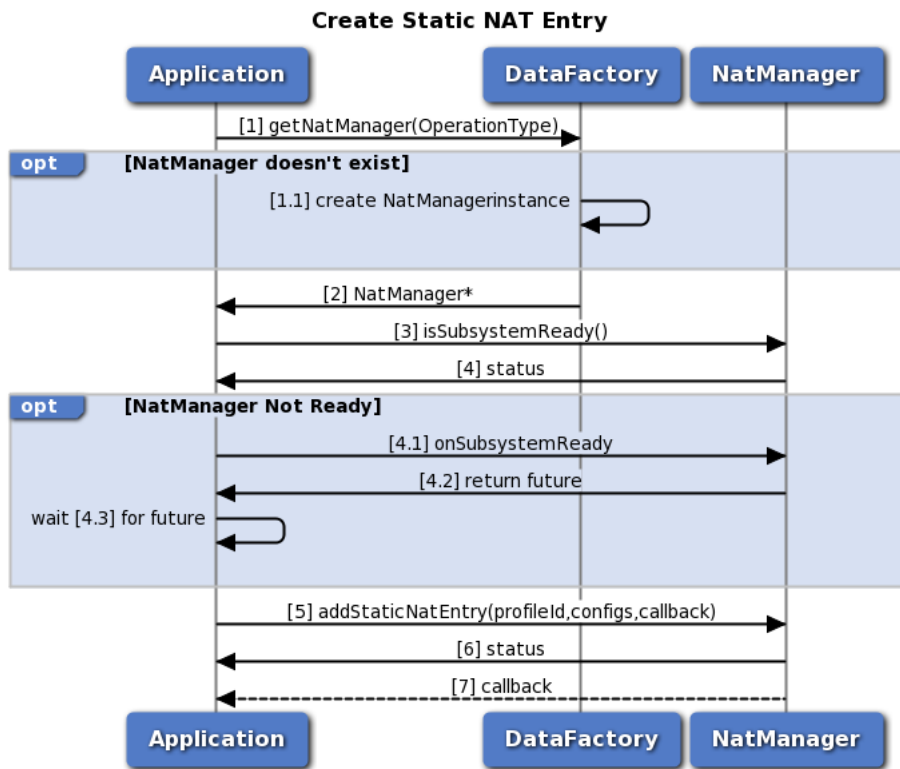


Figure 3-50 LAN-WAN VLAN Configuration Usecase from A7 call flow

1. Application requests data factory for data IVlanManager for remote operation object.
2. Application use IVlanManager::isSubsystemReady to determine if the system is ready. 2.1. If it is not ready, then the application could use onSubsystemReady which returns std::future.

3. On readiness, application calls IVlanManager::createVlan with USB interface, Vlan id 1 and acceleration and wait for callback to indicate createVlan result
4. Application calls IVlanManager::createVlan with ETH interface, Vlan id 1 and acceleration and wait or callback to indicate createVlan result
5. Application calls IVlanManager::bindWithProfile with profile id to bind with and wait for callback to indicate bindWithProfile result If DMZ is needed:
  - (a) Application requests data factory for data IFirewallManager object
  - (b) Application can use IFirewallManager::isSubsystemReady to determine if the system is ready. 7.1. If FirewallManager is not ready, then the application could use onSubsystemReady which returns std::future and wait for future.
  - (c) Application can use IFirewallManager::enableDmz with profile id and local address to be enable Dmz on
6. Application calls getDataConnectionManager to get object of IDataConnectionManager
7. Application can use IDataConnectionManager::isSubsystemReady to determine if the system is ready. 10.1. If DataConnectionManager is not ready, then the application could use onSubsystemReady which returns std::future and wait for future.
8. Application can use IDataConnectionManager::startDataCall with profile id to start data call on, Ip Family type, operation Type and APN Name

**3.5.4.6 Create Static NAT entry in data Static NAT manager call flow**



**Figure 3-51 Create Static NAT entry for data NAT manager call flow**

1. Application requests data factory for data INatManager object. 1.1. If INatManager object does not exist, data factory will create new object.
2. Data factory returns NatManager object to application.
3. Application can use INatManager::isSubsystemReady to determine if the system is ready.
4. The application receives the Status i.e. either true or false to indicate whether sub-system is ready or not. 4.1. If it is not ready, then the application could use onSubsystemReady which returns std::future. 4.2. Application gets future for INatManager readiness. 4.3. Application wait for indefinitely for INatManager readiness.
5. On Readiness, application calls INatManager::addStaticNatEntry with profileId, private IP address port, private port, global port and IP Protocol.
6. Application receives synchronous Status which indicates if the INatManager::addStaticNatEntry request was sent successfully.
7. Application is notified of the Status of the INatManager::addStaticNatEntry request (either SUCCESS or FAILED) via the application-supplied callback.

### 3.5.4.7 Firewall Enablement in data Firewall manager call flow

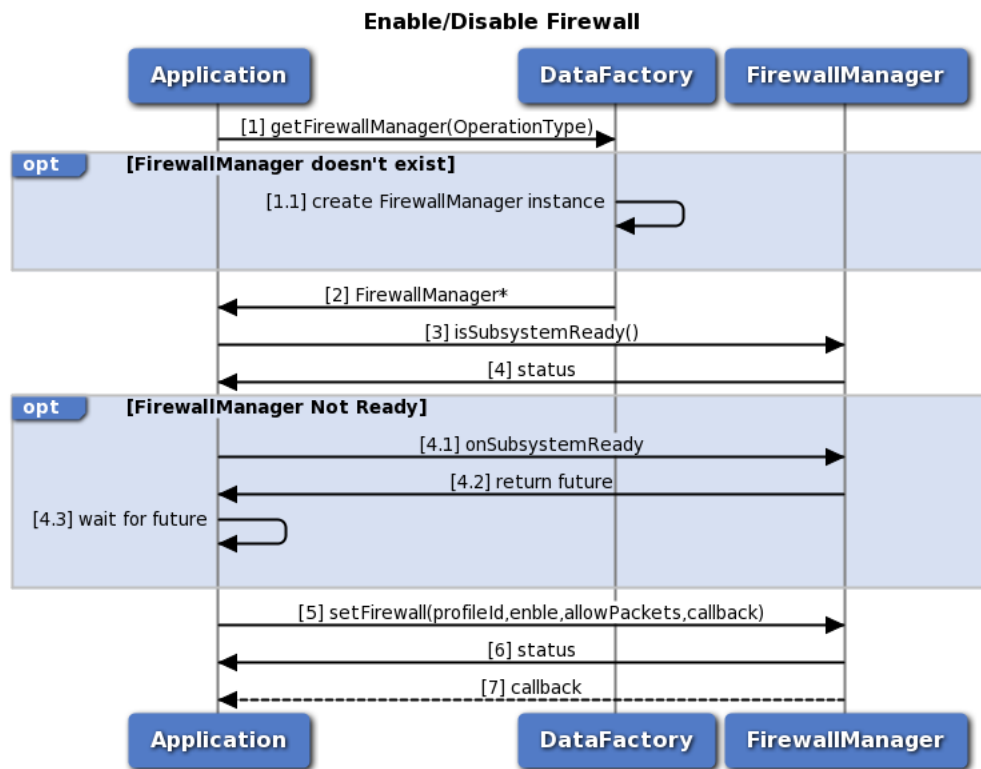


Figure 3-52 Firewall enablement in data Firewall manager call flow

1. Application requests data factory for data IFirewallManager object. 1.1. If IFirewallManager object does not exist, data factory will create new object.
2. Data factory returns FirewallManager object to application.
3. Application can use IFirewallManager::isSubsystemReady to determine if the system is ready.

4. The application receives the Status i.e. either true or false to indicate whether sub-system is ready or not. 4.1. If it is not ready, then the application could use onSubsystemReady which returns std::future. 4.2. Application gets future for IFirewallManager readiness. 4.3. Application wait for indefinitely for IFirewallManager readiness.
5. On Readiness, application calls IFirewallManager::setFirewall with enable/disable and allow/drop packets.
6. Application receives synchronous Status which indicates if the IFirewallManager::setFirewall request was sent successfully.
7. Application is notified of the Status of the IFirewallManager::setFirewall request (either SUCCESS or FAILED) via the application-supplied callback.

### 3.5.4.8 Add Firewall Entry in data Firewall manager call flow

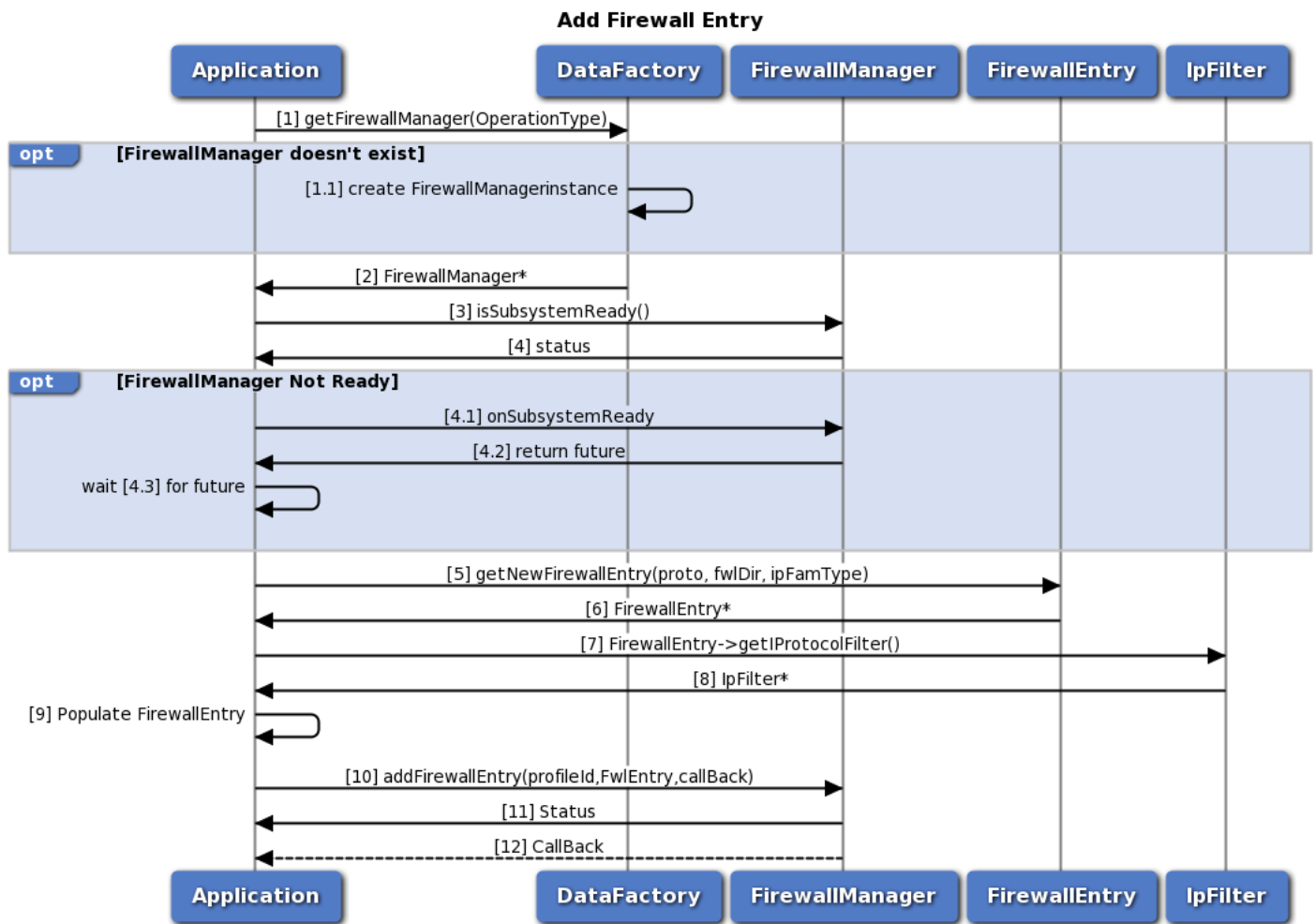


Figure 3-53 Add Firewall entry in data Firewall manager call flow

1. Application requests data factory for data IFirewallManager object. 1.1. If IFirewallManager object does not exist, data factory will create new object.
2. Data factory returns FirewallManager object to application.



3. Application can use `IFirewallManager::isSubsystemReady` to determine if the system is ready.
4. The application receives the Status i.e. either true or false to indicate whether sub-system is ready or not.
  - 4.1. If it is not ready, then the application could use `onSubsystemReady` which returns `std::future`.
  - 4.2. Application gets future for `IFirewallManager` readiness.
  - 4.3. Application wait for indefinitely for `IFirewallManager` readiness.
5. On Readiness, application calls `IFirewallManager::getNewFirewallEntry` to get `FirewallEntry` object.
6. Application receives Firewall Entry object.
7. Using Firewall Entry object, application calls `IFirewallEntry::getProtocolFilter` to get protocol filter object
8. Application receives `IpFilter` object.
9. Application populates `FirewallEntry` and `IpFilter` objects.
10. Application calls `IFirewallManager::addFirewallEntry` with `profileId` and `FirewallEntry` to add firewall entry
11. Application receives synchronous Status which indicates if `addFirewallEntry` was sent successfully
12. Application is notified of the Status of the `IFirewallManager::addFirewallEntry` request (either SUCCESS or FAILED) via the application-supplied callback.

### 3.5.4.9 Set Firewall DMZ in data Firewall manager call flow

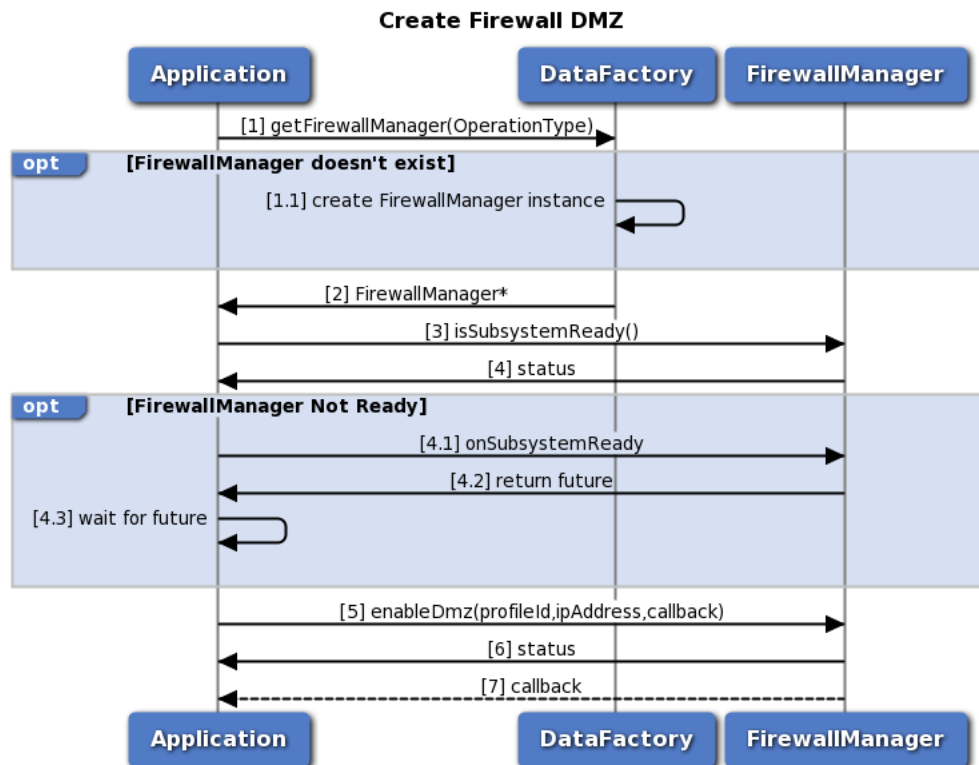
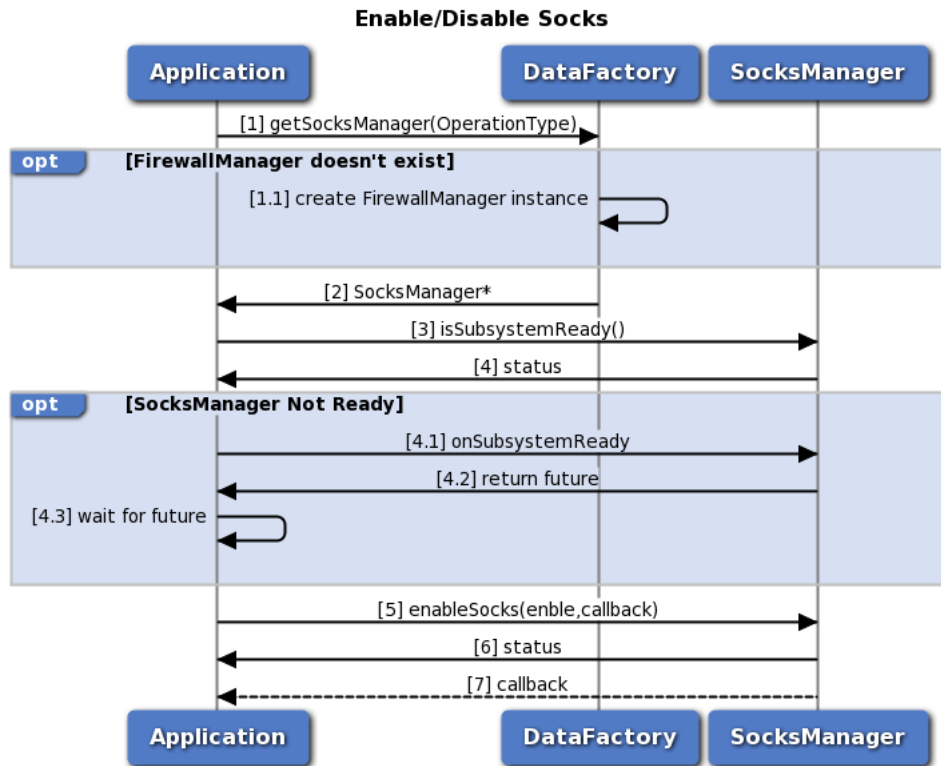


Figure 3-54 Set Firewall DMZ in data Firewall manager call flow

1. Application requests data factory for data IFirewallManager object. 1.1. If IFirewallManager object does not exist, data factory will create new object.
2. Data factory returns FirewallManager object to application.
3. Application can use IFirewallManager::isSubsystemReady to determine if the system is ready.
4. The application receives the Status i.e. either true or false to indicate whether sub-system is ready or not. 4.1. If it is not ready, then the application could use onSubsystemReady which returns std::future. 4.2. Application gets future for IFirewallManager readiness. 4.3. Application wait for indefinitely for IFirewallManager readiness.
5. On Readiness, application calls IFirewallManager::enableDmz with profileId and IP Address.
6. Application receives synchronous Status which indicates if the IFirewallManager::enableDmz request was sent successfully.
7. Application is notified of the Status of the IFirewallManager::enableDmz request (either SUCCESS or FAILED) via the application-supplied callback.

### 3.5.4.10 Socks Enablement in data Socks manager call flow

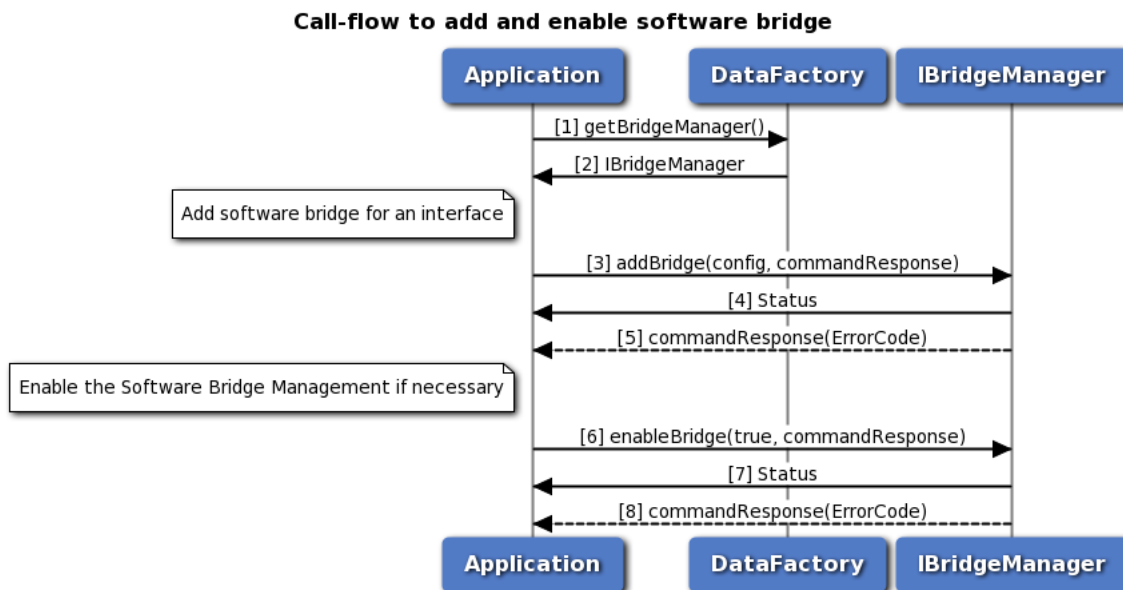


**Figure 3-55 Socks enablement in data Socks manager call flow**

1. Application requests data factory for data ISocksManager object. 1.1. If IFirewallManager object does not exist, data factory will create new object.
2. Data factory returns SocksManager object to application.
3. Application can use ISocksManager::isSubsystemReady to determine if the system is ready.

4. The application receives the Status i.e. either true or false to indicate whether sub-system is ready or not. 4.1. If it is not ready, then the application could use `onSubsystemReady` which returns `std::future`. 4.2. Application gets future for `ISocksManager` readiness. 4.3. Application wait for indefinitely for `ISocksManager` readiness.
5. On Readiness, application calls `ISocksManager::enableSocks` with enable/disable.
6. Application receives synchronous Status which indicates if the `ISocksManager::enableSocks` request was sent successfully.
7. Application is notified of the Status of the `ISocksManager::enableSocks` request (either SUCCESS or FAILED) via the application-supplied callback.

### 3.5.4.11 Call flow to add and enable software bridge

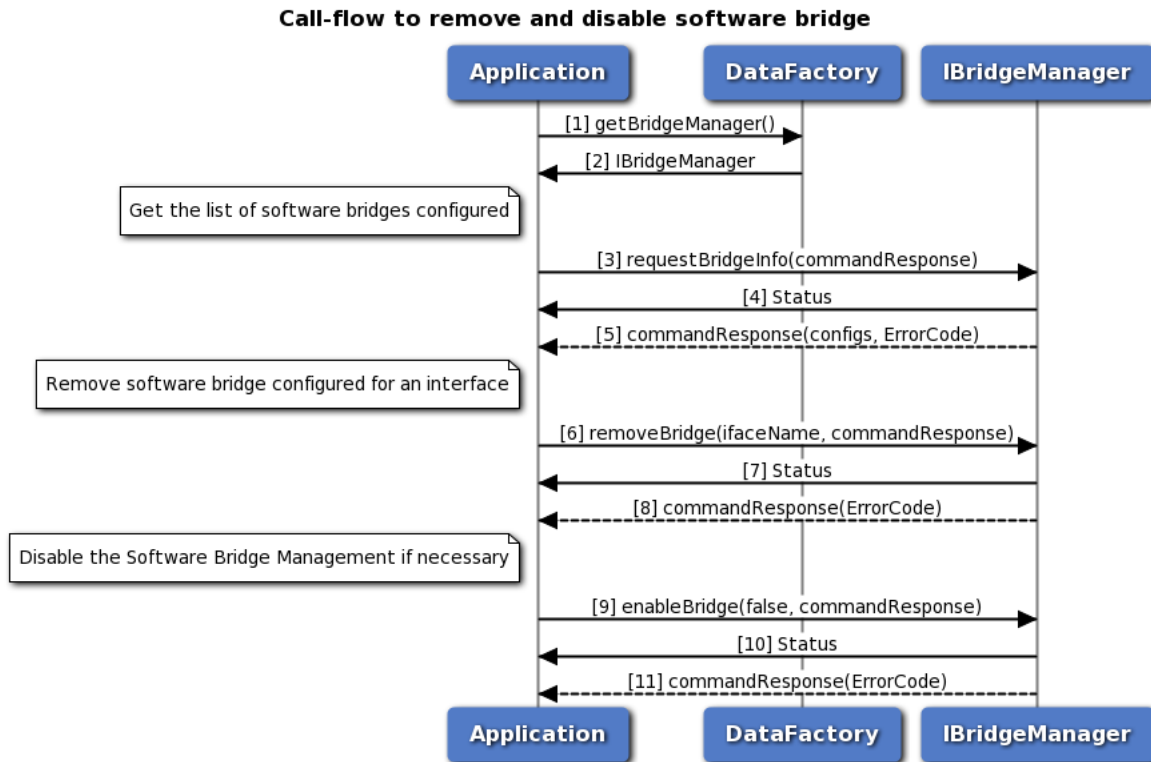


**Figure 3-56 Call flow to add and enable a software bridge**

1. Application requests data factory for Bridge manager object.
2. Data factory returns `IBridgeManager` object to application.
3. On readiness, application requests to add software bridge configuration for an interface, providing an optional asynchronous response callback using `addBridge` API.
4. Application receives the synchronous status i.e. either SUCCESS or FAILED which indicates if the request was sent successfully.
5. Optionally, the application gets asynchronous response for `addBridge` via the application-supplied callback.
6. If the software bridge management is not enabled already, application requests to enable it, providing an optional asynchronous response callback using `enableBridge` API. Please note that this step affects all the software bridges configured in the system.
7. Application receives the status i.e. either SUCCESS or FAILED which indicates if the request was sent successfully.

- Optionally, the application gets asynchronous response for enableBridge via the application-supplied callback.

### 3.5.4.12 Call flow to remove and disable software bridge



**Figure 3-57 Call flow to remove and disable a software bridge**

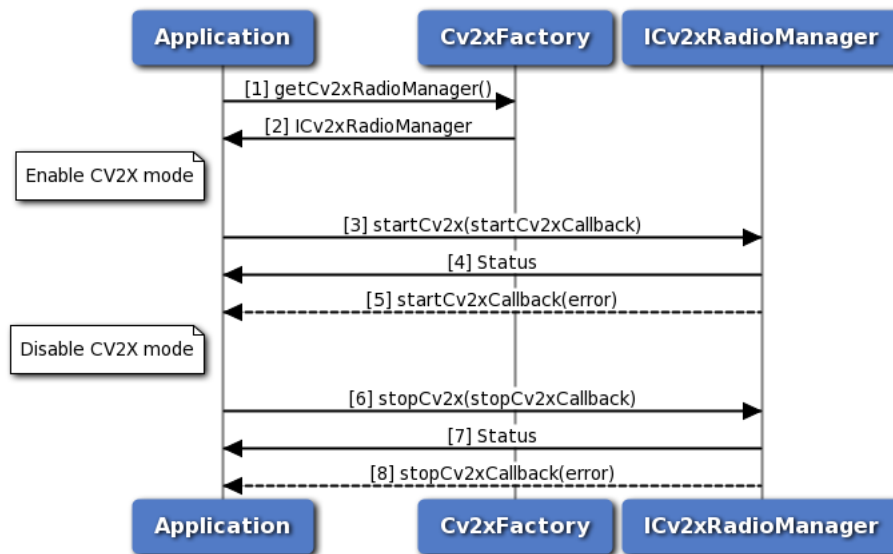
- Application requests data factory for Bridge manager object.
- Data factory returns IBridgeManager object to application.
- On readiness, application requests to get the list of software bridge configurations, providing an asynchronous response callback using requestBridgeInfo API.
- Application receives the synchronous status i.e. either SUCCESS or FAILED which indicates if the request was sent successfully.
- The application gets asynchronous response for requestBridgeInfo via the application-supplied callback.
- Application requests to remove software bridge configuration for an interface, providing an optional asynchronous response callback using removeBridge API.
- Application receives the synchronous status i.e. either SUCCESS or FAILED which indicates if the request was sent successfully.
- Optionally, the application gets asynchronous response for removeBridge via the application-supplied callback.
- If the software bridge management needs to be disabled, application requests to disable it, providing an optional asynchronous response callback using enableBridge API. Please note that this step affects

all the software bridges configured in the system.

10. Application receives the status i.e. either SUCCESS or FAILED which indicates if the request was sent successfully.
11. Optionally, the application gets asynchronous response for enableBridge via the application-supplied callback.

## 3.6 C-V2X

### 3.6.1 Start/Stop C-V2X Mode



**Figure 3-58 Start/Stop C-V2X Mode call flow**

Note: In normal operation, applications do not need to start or stop C-V2X mode. The system is configured by default to start C-V2X mode at boot. We include the call flow below for the sake of completeness.

1. Application requests C-V2X factory for a C-V2X Radio Manager.
2. C-V2X factory return ICv2xRadioManager object to application.
3. Application requests to put modem into C-V2X mode using startCv2x method.
4. Application receives synchronous status which indicates if the start request was sent successfully.
5. Application is notified of the status of the start request (either SUCCESS or FAILED) via the application-supplied callback.
6. Application requests to disable C-V2X mode using stopCv2x method.
7. Application receives synchronous status which indicates if the stop request was sent successfully.
8. Application is asynchronously notified of the status of the stop request (either SUCCESS or FAILED) via the application-supplied callback.

### 3.6.2 C-V2X Radio Manager API

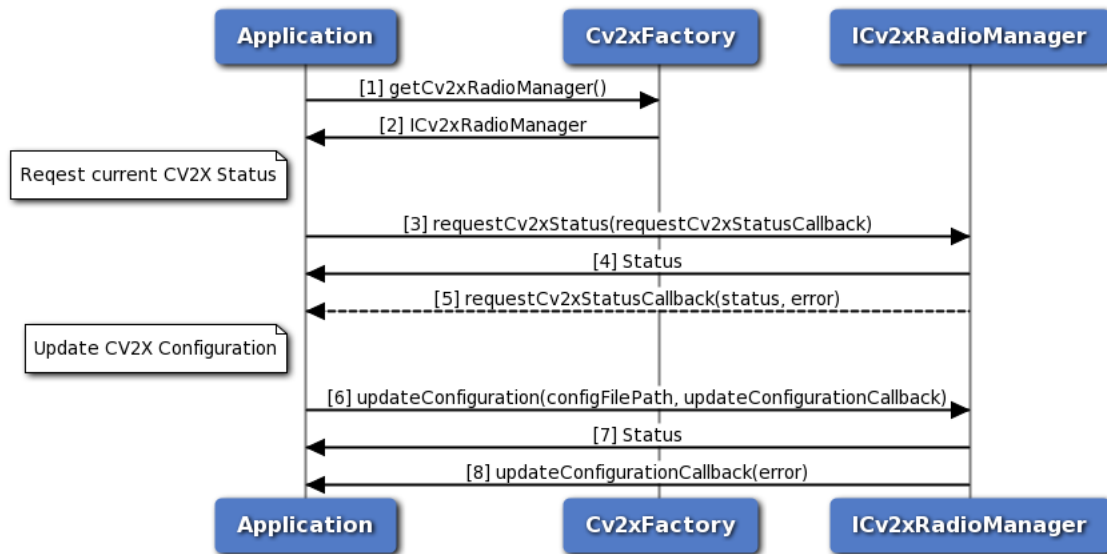


Figure 3-59 C-V2X Radio Manager call flow

#### API for C-V2X Radio Manager

1. Application requests C-V2X factory for a C-V2X Radio Manager.
2. C-V2X factory return `ICv2xRadioManager` object to application.
3. Application requests current C-V2X status using `requestCv2xStatus` method.
4. Application receives synchronous status (either `SUCCESS` or `FAILED`) which indicates if the request was sent successfully.
5. Application is asynchronously notified of the status of the request (either `SUCCESS` or `FAILED`) via the application-supplied callback. If `SUCCESS`, the requested C-V2X status is returned in the callback.
6. Application requests to update the C-V2X configuration by calling `updateConfiguration` and supplying it with a path to the new config XML file.
7. Application receives synchronous status (either `SUCCESS` or `FAILED`) which indicates if the request was sent successfully.
8. Application is asynchronously notified of the status of the request (either `SUCCESS` or `FAILED`) via the application-supplied callback.

### 3.6.3 C-V2X Radio Initialization

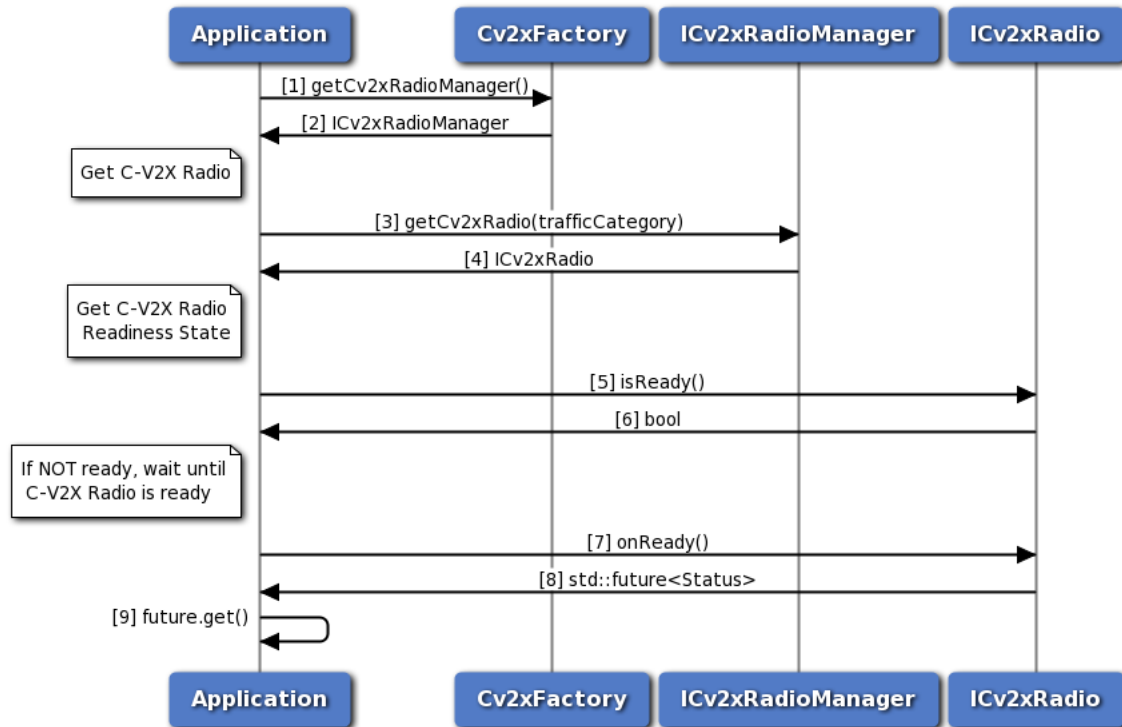


Figure 3-60 C-V2X Radio Initialization call flow

This call flow diagram describes the sequence of steps for initializing the ICv2xRadio object. Applications must perform this sequence before calling any other methods on the object. Note: for simplicity's sake, we omit this sequence in the remaining C-V2X Radio call flow diagrams but its inclusion is implied.

1. Application requests C-V2X factory for a C-V2X Radio Manager.
2. C-V2X factory return ICv2xRadioManager object to application.
3. Application requests C-V2X Radio from ICv2xRadioManager.
4. C-V2X Radio Manager returns ICv2xRadio object.
5. Application queries C-V2X Radio's readiness state isReady() method.
6. C-V2X Radio returns a bool indicating whether it is ready to be used.
7. If the C-V2X Radio is not ready, the application calls onReady() method.
8. The C-V2X Radio returns a future object.
9. Application calls future's get() method and blocks until the C-V2X Radio has completed its initialization steps. The return value of get() indicates the status of the initialization (either SUCCESS or FAILED).

### 3.6.4 C-V2X Radio RX subscription

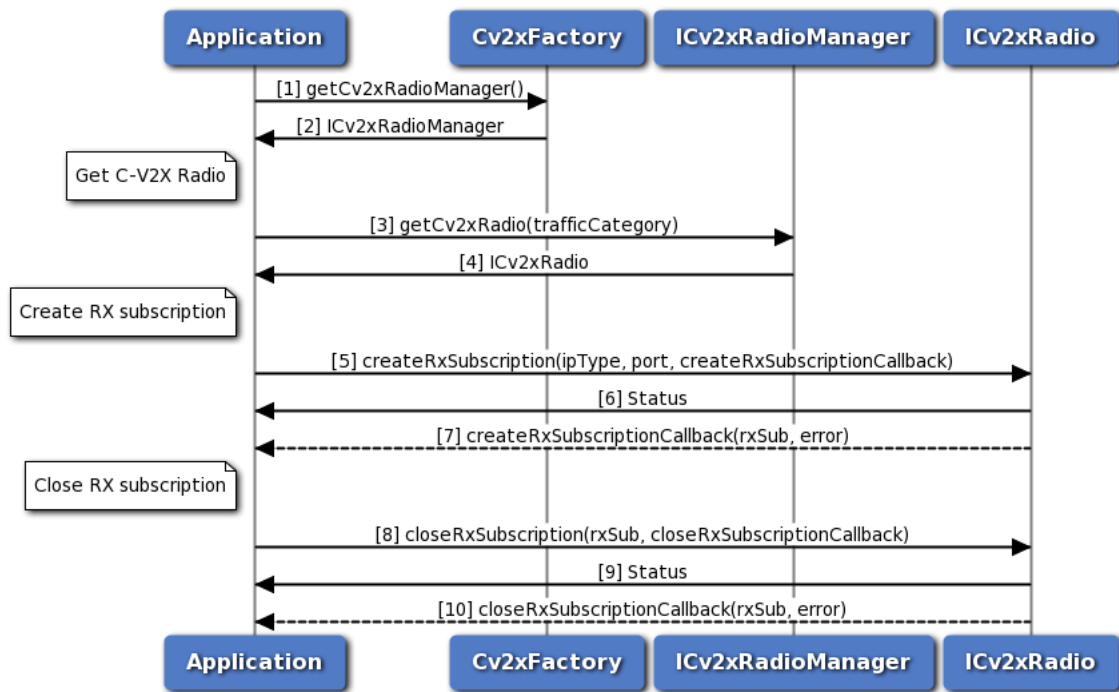


Figure 3-61 C-V2X Radio RX subscription call flow

1. Application requests C-V2X factory for a C-V2X Radio Manager.
2. C-V2X factory return ICv2xRadioManager object to application.
3. Application requests C-V2X Radio from ICv2xRadioManager.
4. C-V2X Radio Manager returns ICv2xRadio object.
5. Application requests a new RX subscription from the C-V2X Radio using createRxSubscription method.
6. Application receives synchronous status (either SUCCESS or FAILED) indicating whether the request was sent successfully.
7. C-V2X Radoi sends asynchronous notification via the callback function on the status of the request. If SUCCESS, the RX subscription is returned in the callback.
8. Application requests to close the RX subscription.
9. Application receives synchronous status (either SUCCESS or FAILED) indicating whether the request was sent successfully.
10. C-V2X Radio sends asynchronous notification via the callback (if a callback was specified) indicating the status of the request.



### 3.6.5 C-V2X Radio TX event-driven flow

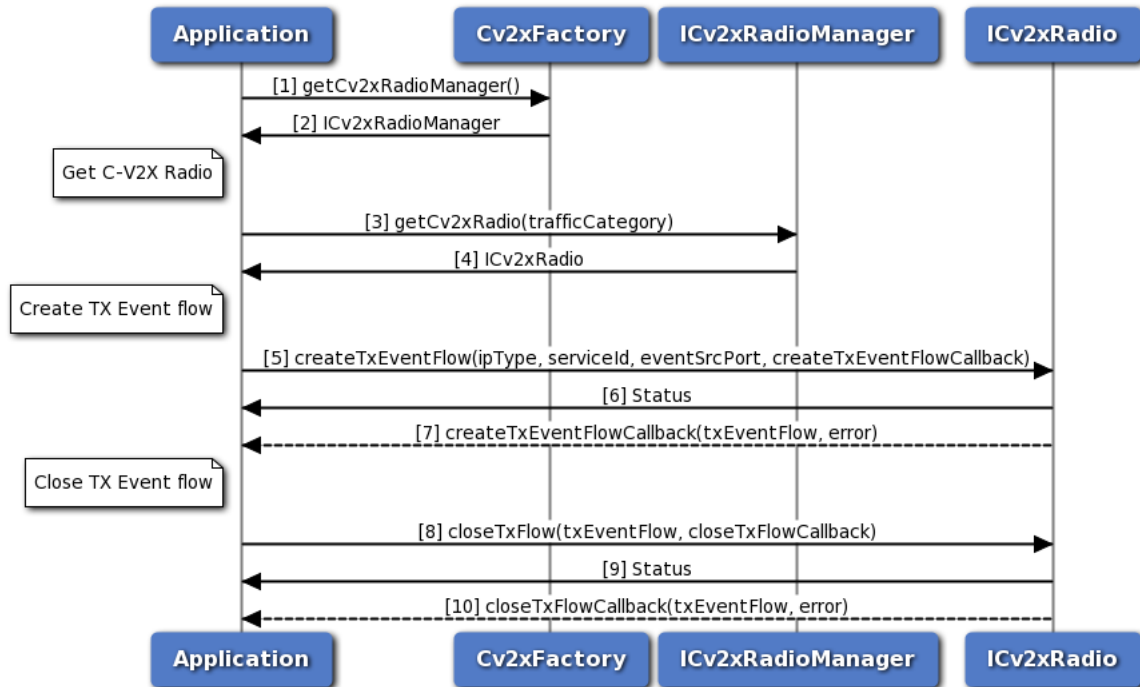


Figure 3-62 C-V2X Radio TX event-driven flow call flow

1. Application requests C-V2X factory for a C-V2X Radio Manager.
2. C-V2X factory return ICv2xRadioManager object to application.
3. Application requests C-V2X Radio from ICv2xRadioManager.
4. C-V2X Radio Manager returns ICv2xRadio object.
5. Application requests a new TX event-driven flow from the C-V2X Radio using createTxEventFlow method.
6. Application receives synchronous status (either SUCCESS or FAILED) indicating whether the request was sent successfully.
7. C-V2X Radio sends asynchronous notification via the callback function on the status of the request. If SUCCESS, the TX event-driven flow is returned in the callback.
8. Application requests to close the TX event-driven flow.
9. Application receives synchronous status (either SUCCESS or FAILED) indicating whether the request was sent successfully.
10. C-V2X Radio sends asynchronous notification via the callback (if a callback was specified) indicating the status of the request.

### 3.6.6 C-V2X Radio TX SPS flow

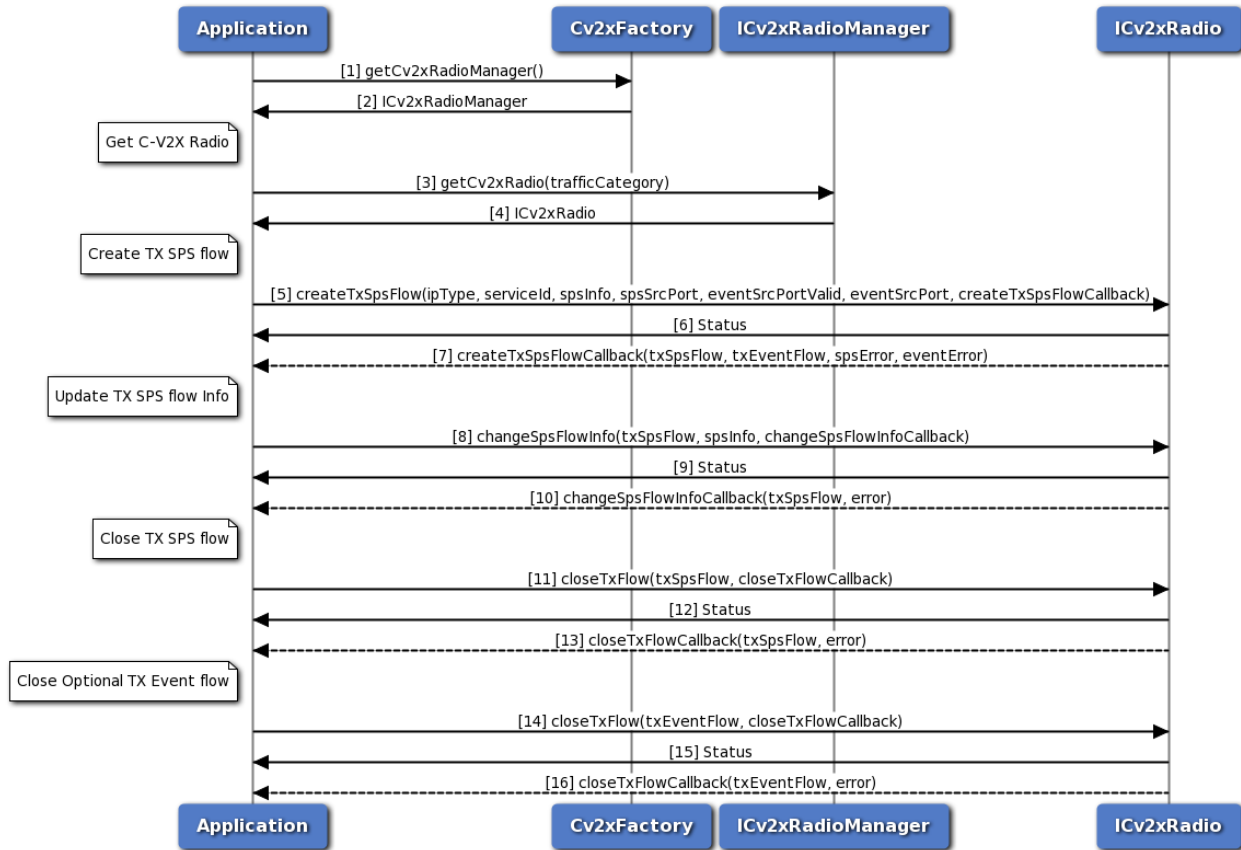


Figure 3-63 C-V2X Radio SPS flow call flow

1. Application requests C-V2X factory for a C-V2X Radio Manager.
2. C-V2X factory return ICv2xRadioManager object to application.
3. Application requests C-V2X Radio from ICv2xRadioManager.
4. C-V2X Radio Manager returns ICv2xRadio object.
5. Application requests a new TX SPS flow from the C-V2X Radio using createTxSPSFlow method. The application can also specify an optional event flow.
6. Application receives synchronous status (either SUCCESS or FAILED) indicating whether the request was sent successfully.
7. C-V2X Radio sends asynchronous notification via the callback function. The callback will return the SPS flow and its status as well as the optional event-driven flow and its status.
8. Application requests to change the SPS parameters using the changeSpsFlowInfo method.
9. Application received synchronous status (either SUCCESS or FAILED) indicating whether the request was sent successfully.
10. C-V2X Radio sends asynchronous notification via the callback (if callback was specified) indicating the status of the request.

11. Application requests to close the SPS flow.
12. Application receives synchronous status (either SUCCESS or FAILED) indicating whether the request was sent successfully.
13. C-V2X Radio sends asynchronous notification via the callback (if a callback was specified) indicating the status of the request.
14. Application requests to close optional event-driven flow (if one was created).
15. Application receives synchronous status (either SUCCESS or FAILED) indicating whether the request was sent successfully.
16. C-V2X Radio sends asynchronous notification via the callback (if a callback was specified) indicating the status of the request.

## 3.7 Audio

### 3.7.1 Audio Manager API call flow

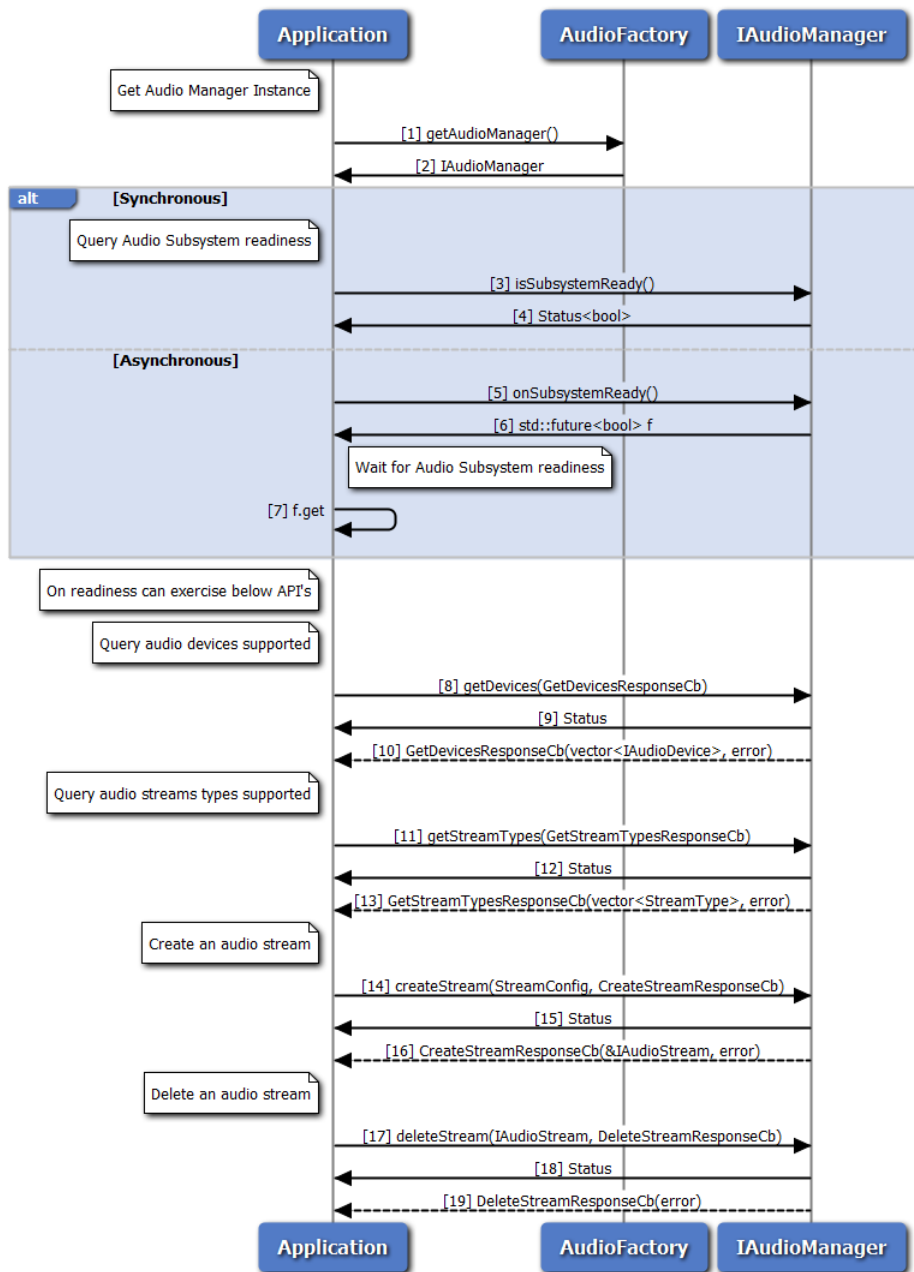


Figure 3-64 Audio Manager API call flow

1. Application requests Audio factory for an Audio Manager.
2. Audio factory return IAudioManager object to application.
3. Application can use IAudioManager::isSubsystemReady to determine if the system is ready.
4. The application receives the Status i.e. either true or false to indicate whether sub-system is ready or not.
5. If it is not ready, then the application could use onSubsystemReady which returns std::future.

6. AudioManager notifies the application when the subsystem is ready through the `std::future` object.
7. The application waits until the asynchronous operation i.e `onSubsystemReady` completes.
8. On Readiness, Application requests supported device types using `getDevices` method.
9. Application receives synchronous Status which indicates if the `getDevices` request was sent successfully.
10. Application is notified of the Status of the `getDevices` request (either SUCCESS or FAILED) via the application-supplied callback, with array of supported device types.
11. Application requests supported stream types using `getStreamTypes` method.
12. Application receives synchronous Status which indicates if the `getStreamTypes` request was sent successfully.
13. Application is notified of the Status of the `getStreamTypes` request (either SUCCESS or FAILED) via the application-supplied callback, with array of supported stream types.
14. Application requests create audio stream using `createStream` method.
15. Application receives synchronous Status which indicates if the `createStream` request was sent successfully.
16. Application is notified of the Status of the `createStream` request (either SUCCESS or FAILED) via the application-supplied callback, with pointer to stream interface.
17. Application requests delete audio stream using `deleteStream` method.
18. Application receives synchronous Status which indicates if the `deleteStream` request was sent successfully.
19. Application is notified of the Status of the `deleteStream` request (either SUCCESS or FAILED) via the application-supplied callback.

### 3.7.2 Audio Voice Call Start/Stop call flow

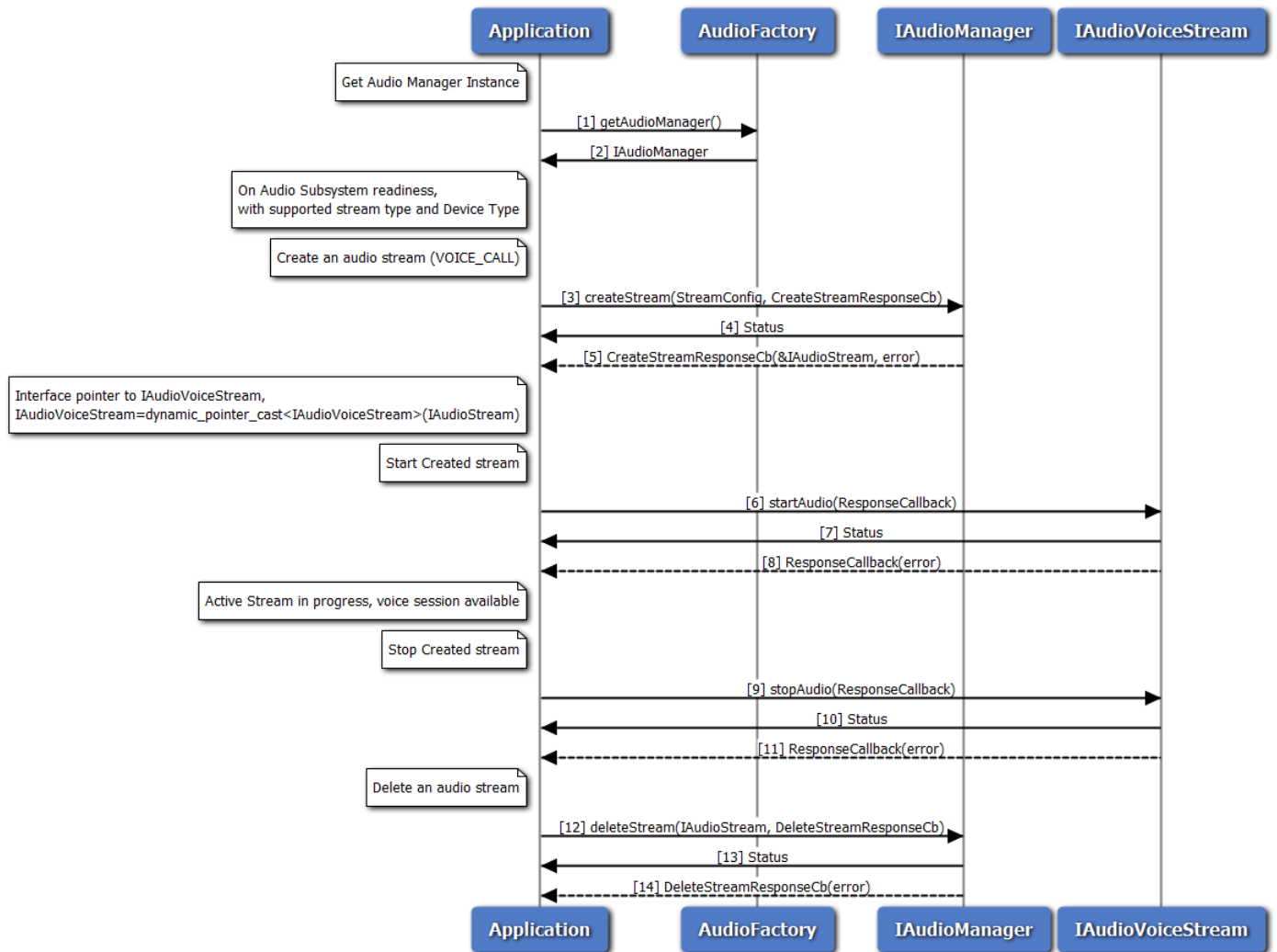


Figure 3-65 Audio Voice Call Start/Stop call flow

1. Application requests Audio factory for an Audio Manager.
2. Audio factory return IAudioManager object to application.
3. On Readiness, Application requests create audio voice stream using createStream method with streamType as VOICE\_CALL.
4. Application receives synchronous Status which indicates if the createStream request was sent successfully.
5. Application is notified of the Status of the createStream request (either SUCCESS or FAILED) via the application-supplied callback, with pointer to stream interface referring to IAudioVoiceStream.
6. Application requests start audio stream using startAudio method on IAudioVoiceStream.
7. Application receives synchronous Status which indicates if the startAudio request was sent successfully.

8. Application is notified of the Status of the startAudio request (either SUCCESS or FAILED) via the application-supplied callback.
9. Application requests stop audio stream using stopAudio method on IAudioVoiceStream.
10. Application receives synchronous Status which indicates if the stopAudio request was sent successfully.
11. Application is notified of the Status of the stopAudio request (either SUCCESS or FAILED) via the application-supplied callback.
12. Application requests delete audio stream using deleteStream method.
13. Application receives synchronous Status which indicates if the deleteStream request was sent successfully.
14. Application is notified of the Status of the deleteStream request (either SUCCESS or FAILED) via the application-supplied callback.

### 3.7.3 Audio Voice Call Device Switch call flow

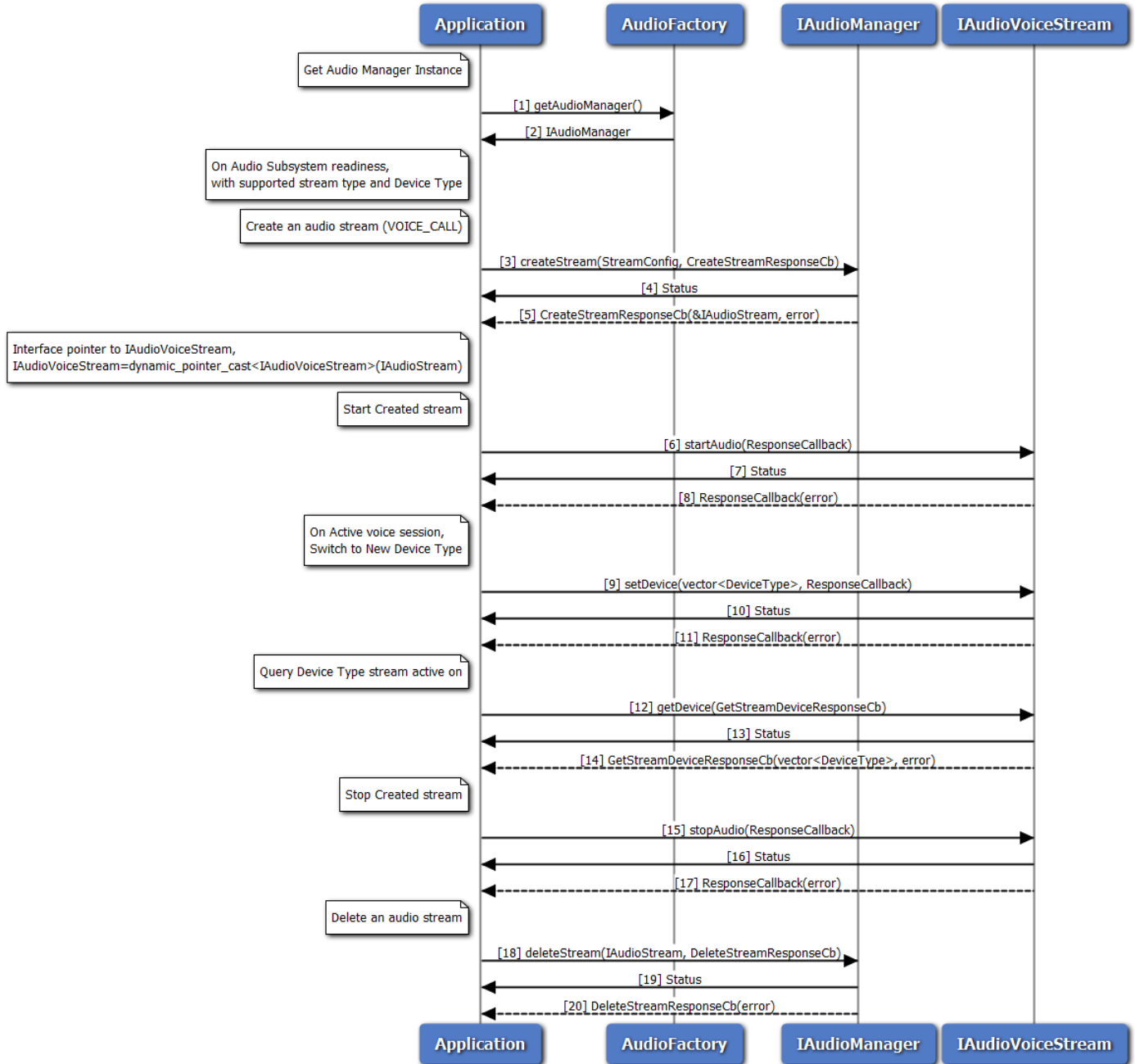


Figure 3-66 Audio Voice Call Device Switch call flow

1. Application requests Audio factory for an Audio Manager.
2. Audio factory return IAudioManager object to application.
3. On Readiness, Application requests create audio voice stream using createStream method with streamType as VOICE\_CALL.
4. Application receives synchronous Status which indicates if the createStream request was sent successfully.



5. Application is notified of the Status of the createStream request (either SUCCESS or FAILED) via the application-supplied callback, with pointer to stream interface referring to IAudioVoiceStream.
6. Application requests start audio stream using startAudio method on IAudioVoiceStream.
7. Application receives synchronous Status which indicates if the startAudio request was sent successfully.
8. Application is notified of the Status of the startAudio request (either SUCCESS or FAILED) via the application-supplied callback.
9. Application requests new device routing of stream using setDevice method on IAudioVoiceStream.
10. Application receives synchronous Status which indicates if the setDevice request was sent successfully.
11. Application is notified of the Status of the setDevice request (either SUCCESS or FAILED) via the application-supplied callback.
12. Application query device stream routed to using getDevice method on IAudioVoiceStream.
13. Application receives synchronous Status which indicates if the getDevice request was sent successfully.
14. Application is notified of the Status of the getDevice request (either SUCCESS or FAILED) via the application-supplied callback, along with device types.
15. Application requests stop audio stream using stopAudio method on IAudioVoiceStream.
16. Application receives synchronous Status which indicates if the stopAudio request was sent successfully.
17. Application is notified of the Status of the stopAudio request (either SUCCESS or FAILED) via the application-supplied callback.
18. Application requests delete audio stream using deleteStream method.
19. Application receives synchronous Status which indicates if the deleteStream request was sent successfully.
20. Application is notified of the Status of the deleteStream request (either SUCCESS or FAILED) via the application-supplied callback.

### 3.7.4 Audio Voice Call Volume/Mute control call flow

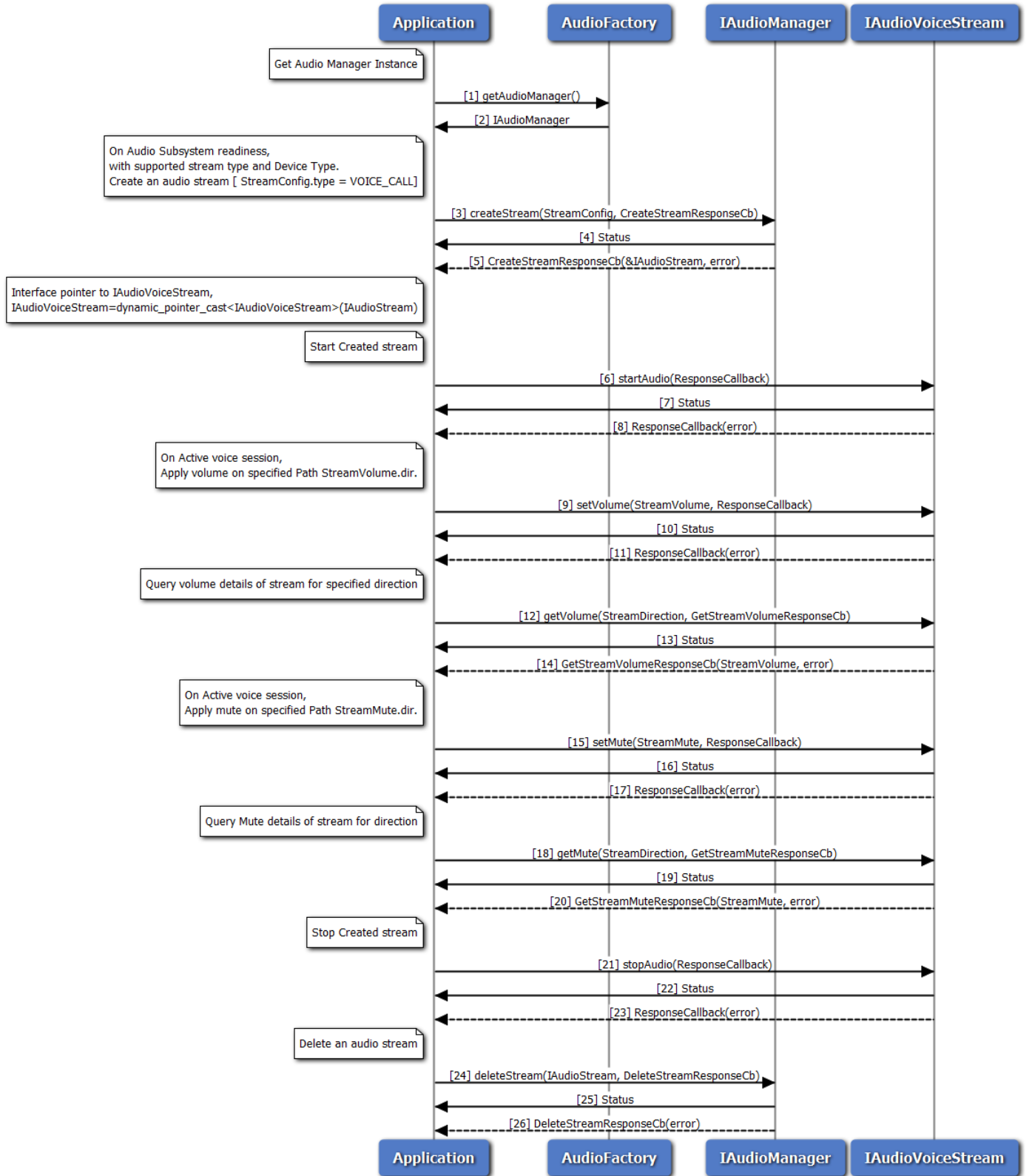


Figure 3-67 Audio Voice Call Volume/Mute control call flow

1. Application requests Audio factory for an Audio Manager.
2. Audio factory return IAudioManager object to application.
3. On Readiness, Application requests create audio voice stream using createStream method with streamType as VOICE\_CALL.
4. Application receives synchronous Status which indicates if the createStream request was sent successfully.
5. Application is notified of the Status of the createStream request (either SUCCESS or FAILED) via the application-supplied callback, with pointer to stream interface referring to IAudioVoiceStream.
6. Application requests start audio stream using startAudio method on IAudioVoiceStream.
7. Application receives synchronous Status which indicates if the startAudio request was sent successfully.
8. Application is notified of the Status of the startAudio request (either SUCCESS or FAILED) via the application-supplied callback.
9. Application requests new volume on stream using setVolume method on IAudioVoiceStream for specified direction.
10. Application receives synchronous Status which indicates if the setVolume request was sent successfully.
11. Application is notified of the Status of the setVolume request (either SUCCESS or FAILED) via the application-supplied callback.
12. Application query volume on stream using getVolume method on IAudioVoiceStream for specified direction.
13. Application receives synchronous Status which indicates if the getVolume request was sent successfully.
14. Application is notified of the Status of the getVolume request (either SUCCESS or FAILED) via the application-supplied callback for specified direction with volume details.
15. Application requests new mute on stream using setMute method on IAudioVoiceStream for specified direction.
16. Application receives synchronous Status which indicates if the setMute request was sent successfully.
17. Application is notified of the Status of the setMute request (either SUCCESS or FAILED) via the application-supplied callback.
18. Application query mute details on stream using getMute method on IAudioVoiceStream for specified direction.
19. Application receives synchronous Status which indicates if the getMute request was sent successfully.
20. Application is notified of the Status of the getMute request (either SUCCESS or FAILED) via the application-supplied callback for specified direction with mute details.
21. Application requests stop audio stream using stopAudio method on IAudioVoiceStream.
22. Application receives synchronous Status which indicates if the stopAudio request was sent successfully.
23. Application is notified of the Status of the stopAudio request (either SUCCESS or FAILED) via the application-supplied callback.

- 24. Application requests delete audio stream using deleteStream method.
- 25. Application receives synchronous Status which indicates if the deleteStream request was sent successfully.
- 26. Application is notified of the Status of the deleteStream request (either SUCCESS or FAILED) via the application-supplied callback.

### 3.7.5 Call flow to play DTMF tone

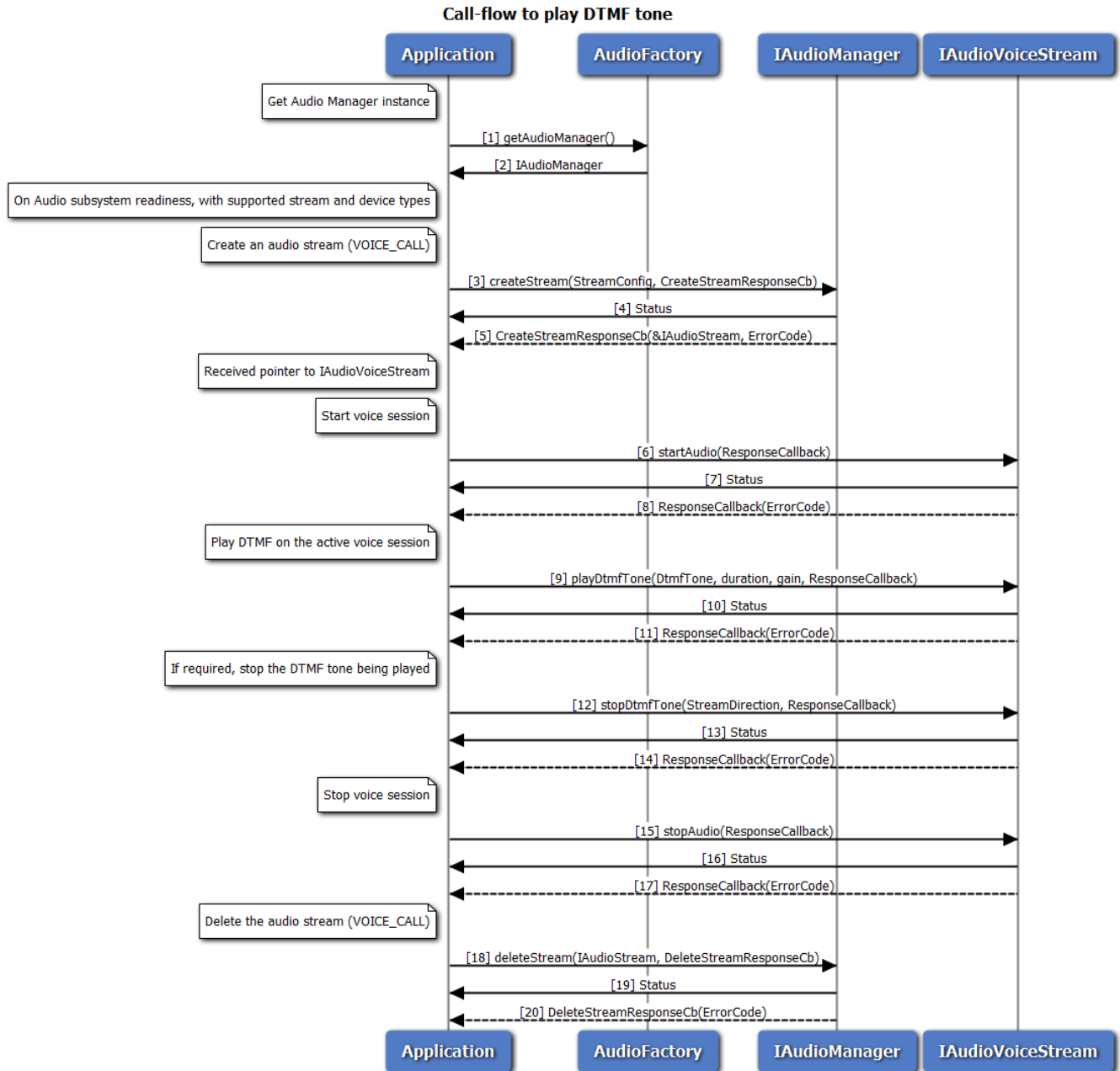
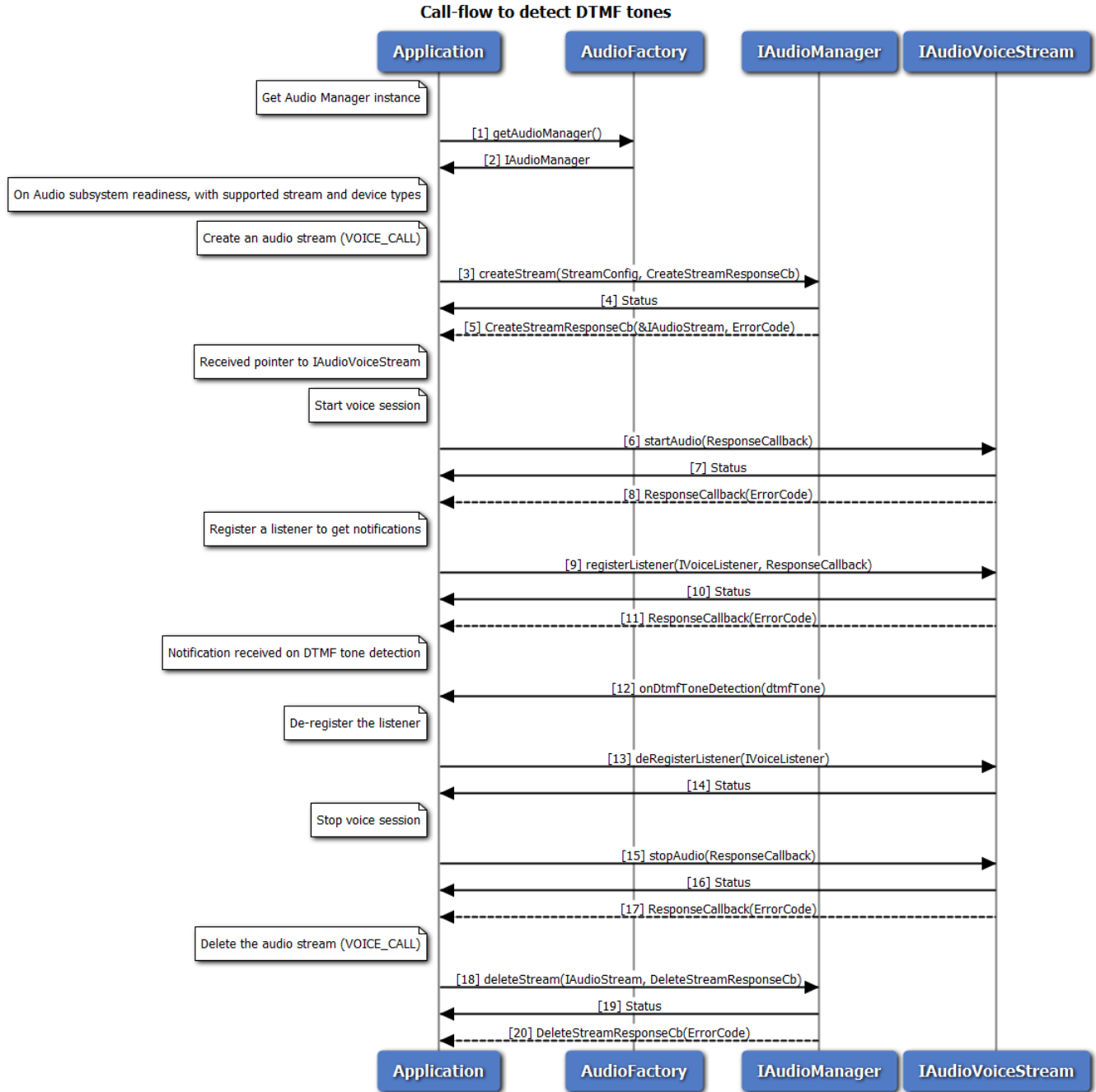


Figure 3-68 Call flow to play DTMF tone

1. Application requests Audio factory for an Audio Manager.
2. Audio factory return IAudioManager object to application.
3. On Readiness, Application requests to create a voice stream with streamType as VOICE\_CALL.
4. Application receives synchronous status which indicates if the createStream request was sent successfully.
5. Application is notified of the createStream request status (either SUCCESS or FAILED) via the application-supplied callback, with pointer to stream interface referring to IAudioVoiceStream.
6. Application requests to start voice session using startAudio method on IAudioVoiceStream.
7. Application receives synchronous status which indicates if the startAudio request was sent successfully.
8. Application is notified of the startAudio request status (either SUCCESS or FAILED) via the application-supplied callback.
9. Application requests to play a DTMF tone associated with the voice session
10. Application receives synchronous status which indicates if the playDtmfTone request was sent successfully.
11. Application is notified of the playDtmfTone request status (either SUCCESS or FAILED) via the application-supplied callback.
12. Application can optionally stop the DTMF tone being played, before its duration expires.
13. Application receives synchronous status which indicates if the stopDtmfTone request was sent successfully.
14. Application is notified of the stopDtmfTone request status (either SUCCESS or FAILED) via the application-supplied callback.
15. Application requests to stop the voice session using stopAudio method on IAudioVoiceStream.
16. Application receives synchronous Status which indicates if the stopAudio request was sent successfully.
17. Application is notified of the stopAudio request status(either SUCCESS or FAILED) via the application-supplied callback.
18. Application requests delete audio stream using deleteStream method.
19. Application receives synchronous Status which indicates if the deleteStream request was sent successfully.
20. Application is notified of the deleteStream request status(either SUCCESS or FAILED) via the application-supplied callback.

### 3.7.6 Call flow to detect DTMF tones



**Figure 3-69 Call flow to detect DTMF tone**

1. Application requests Audio factory for an Audio Manager.
2. Audio factory return IAudioManager object to application.
3. On Readiness, Application requests to create a voice stream with streamType as VOICE\_CALL.
4. Application receives synchronous status which indicates if the createStream request was sent successfully.

5. Application is notified of the createStream request status (either SUCCESS or FAILED) via the application-supplied callback, with pointer to stream interface referring to IAudioVoiceStream.
6. Application requests to start voice session using startAudio method on IAudioVoiceStream.
7. Application receives synchronous status which indicates if the startAudio request was sent successfully.
8. Application is notified of the startAudio request status (either SUCCESS or FAILED) via the application-supplied callback.
9. Application registers a listener for getting notifications when DTMF tones are detected
10. Application receives synchronous status which indicates if the registerListener request was sent successfully.
11. Application is notified of the registerListener request status (either SUCCESS or FAILED) via the application-supplied callback.
12. Application receives onDtmfToneDetection notification when a DTMF tone is detected in the active voice call session
13. Application deregisters a listener to stop getting notifications
14. Application receives synchronous status which indicates if the deRegisterListener request was sent successfully.
15. Application requests to stop the voice session using stopAudio method on IAudioVoiceStream.
16. Application receives synchronous Status which indicates if the stopAudio request was sent successfully.
17. Application is notified of the stopAudio request status(either SUCCESS or FAILED) via the application-supplied callback.
18. Application requests delete audio stream using deleteStream method.
19. Application receives synchronous Status which indicates if the deleteStream request was sent successfully.
20. Application is notified of the deleteStream request status(either SUCCESS or FAILED) via the application-supplied callback.

### 3.7.7 Audio Playback call flow

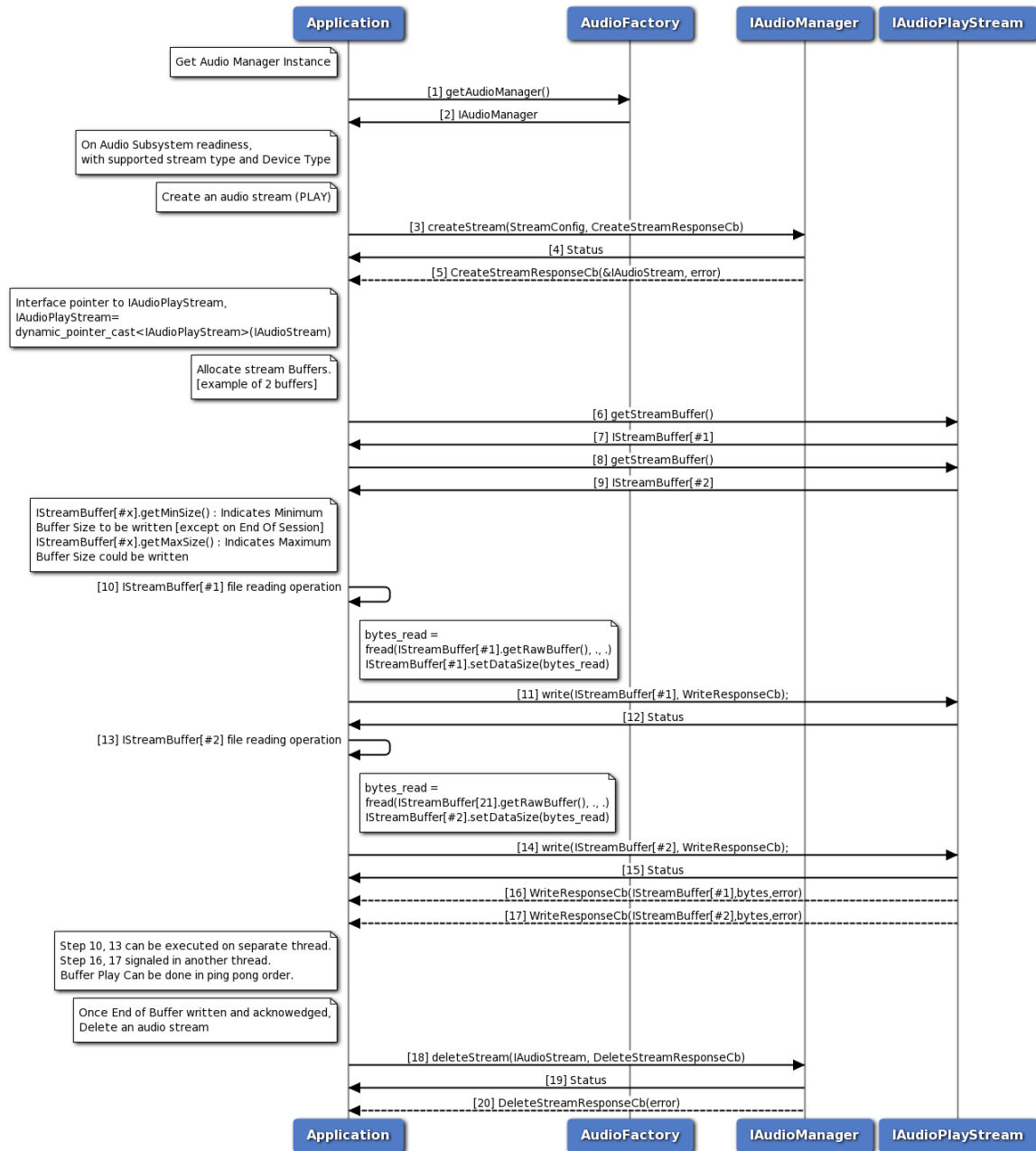


Figure 3-70 Audio Playback call flow

1. Application requests Audio factory for an Audio Manager.
2. Audio factory return IAudioManager object to application.
3. On Readiness, Application requests create audio playback stream using createStream method with streamType as PLAY.
4. Application receives synchronous Status which indicates if the createStream request was sent successfully.
5. Application is notified of the Status of the createStream request (either SUCCESS or FAILED) via



- the application-supplied callback, with pointer to stream interface referring to IAudioPlayStream.
6. Application requests stream buffer#1 using getStreamBuffer method on IAudioPlayStream.
  7. Application receives IStreamBuffer if Success.
  8. Application requests stream buffer#2 using getStreamBuffer method on IAudioPlayStream.
  9. Application receives IStreamBuffer if Success.
  10. Application writes audio samples on buffer#1 using getRawBuffer method on IStreamBuffer.
  11. Application writes buffer#1 on Playback session using write method on IAudioPlayStream.
  12. Application receives synchronous Status which indicates if the write request was sent successfully.
  13. Application writes audio samples on buffer#2 using getRawBuffer method on IStreamBuffer.
  14. Application writes buffer#2 on Playback session using write method on IAudioPlayStream.
  15. Application receives synchronous Status which indicates if the write request was sent successfully.
  16. Application is notified of the buffer#1 write Status (either SUCCESS or FAILED) via the application-supplied write callback with successful bytes written.
  17. Application is notified of the buffer#2 write Status (either SUCCESS or FAILED) via the application-supplied write callback with successful bytes written.
  18. Application requests delete audio stream using deleteStream method.
  19. Application receives synchronous Status which indicates if the deleteStream request was sent successfully.
  20. Application is notified of the Status of the deleteStream request (either SUCCESS or FAILED) via the application-supplied callback.

### 3.7.8 Audio Capture call flow

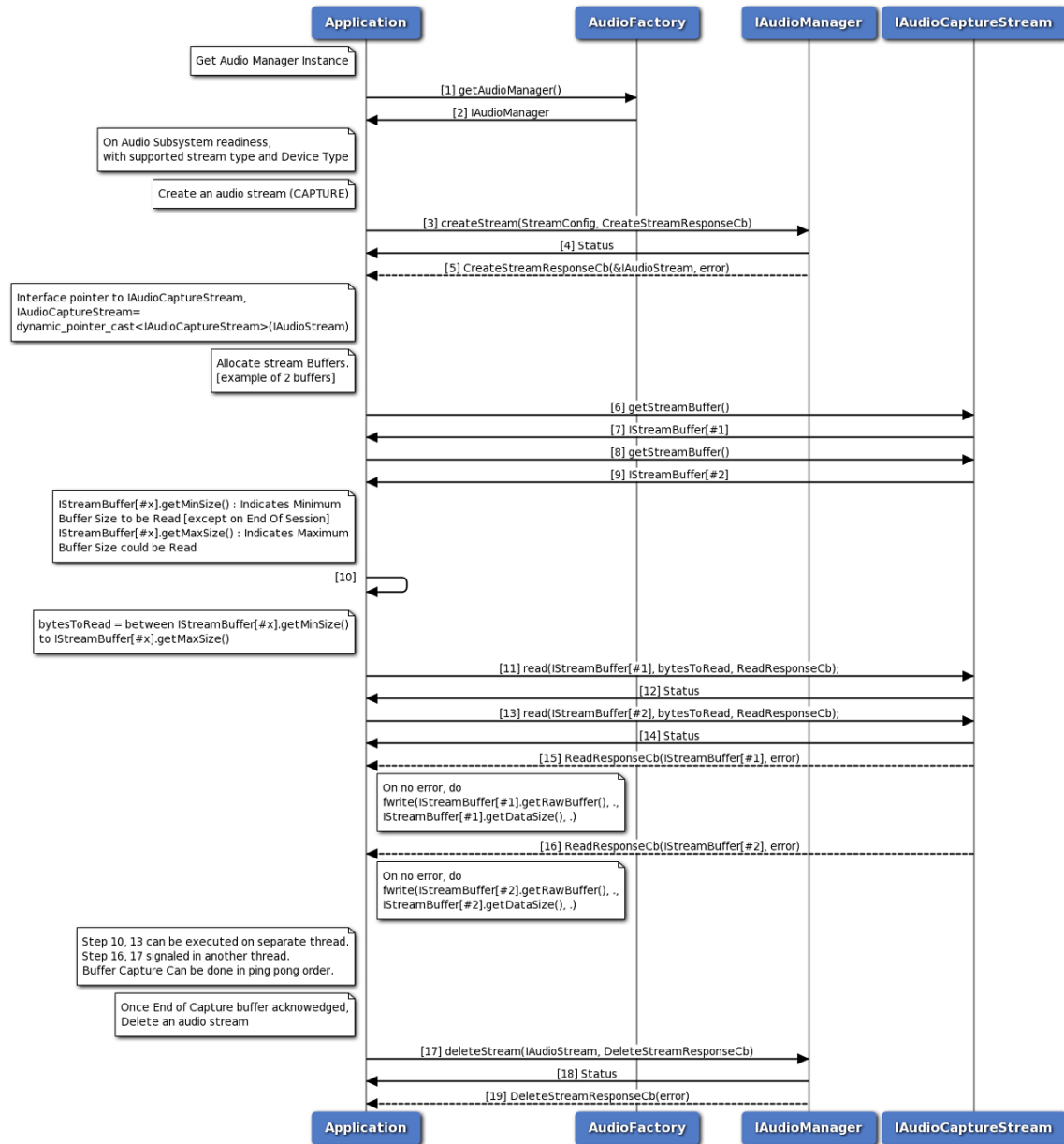
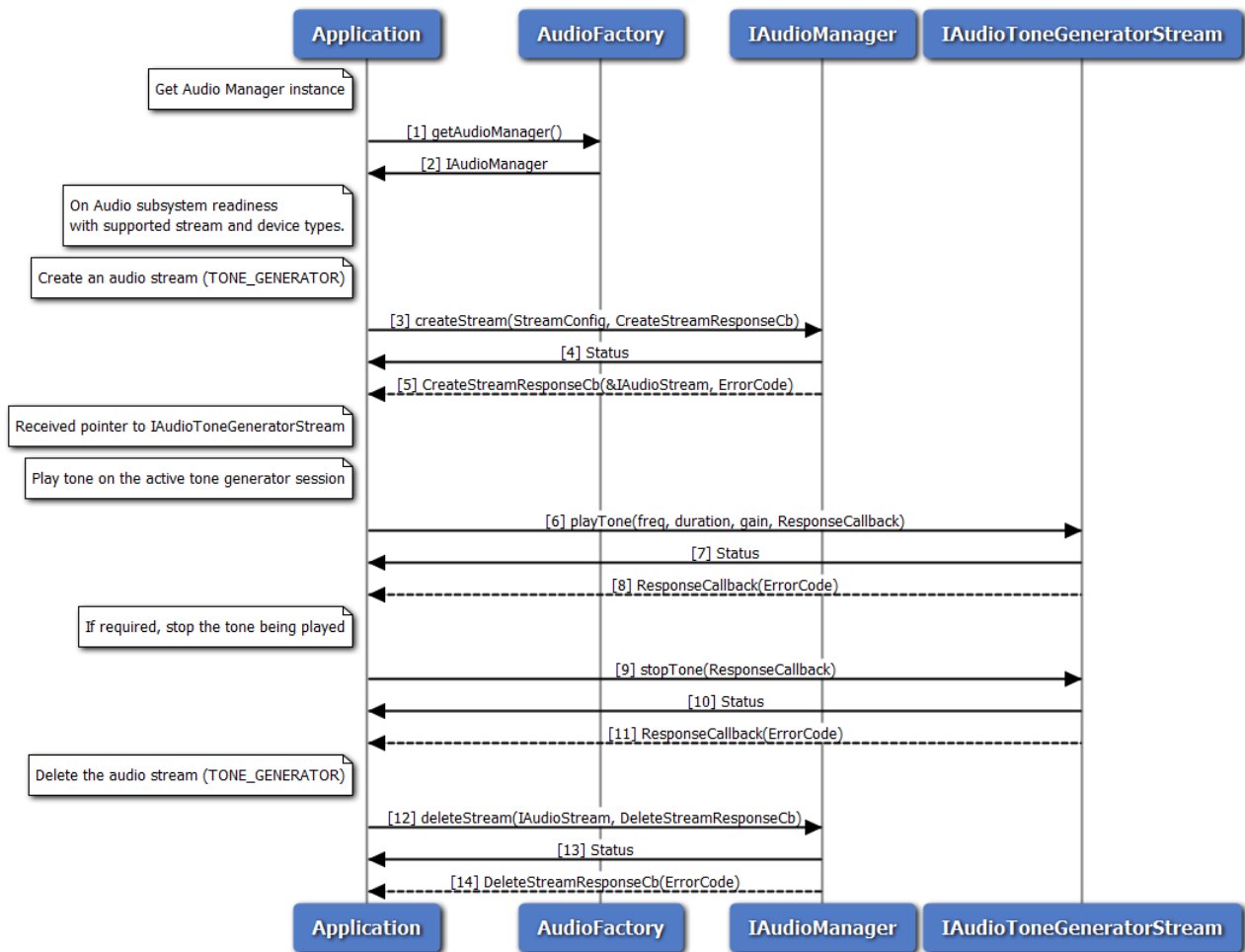


Figure 3-71 Audio Capture call flow

1. Application requests Audio factory for an Audio Manager.
2. Audio factory return IAudioManager object to application.
3. On Readiness, Application requests create audio capture stream using createStream method with streamType as CAPTURE.
4. Application receives synchronous Status which indicates if the createStream request was sent successfully.
5. Application is notified of the Status of the createStream request (either SUCCESS or FAILED) via the application-supplied callback, with pointer to stream interface referring to IAudioCaptureStream.

6. Application requests stream buffer#1 using `getStreamBuffer` method on `IAudioCaptureStream`.
7. Application receives `IStreamBuffer` if Success.
8. Application requests stream buffer#2 using `getStreamBuffer` method on `IAudioCaptureStream`.
9. Application receives `IStreamBuffer` if Success.
10. Application decides read sample size.
11. Application issue read audio samples on buffer#1 using `read` method on `IAudioCaptureStream`.
12. Application receives synchronous Status which indicates if the read request was sent successfully.
13. Application issue read audio samples on buffer#2 using `read` method on `IAudioCaptureStream`.
14. Application receives synchronous Status which indicates if the read request was sent successfully.
15. Application is notified of the buffer#1 write Status (either `SUCCESS` or `FAILED`) via the application-supplied read callback with successful bytes read.
16. Application is notified of the buffer#2 write Status (either `SUCCESS` or `FAILED`) via the application-supplied read callback with successful bytes read.
17. Application requests delete audio stream using `deleteStream` method.
18. Application receives synchronous Status which indicates if the `deleteStream` request was sent successfully.
19. Application is notified of the Status of the `deleteStream` request (either `SUCCESS` or `FAILED`) via the application-supplied callback.

### 3.7.9 Audio Tone Generator call flow



**Figure 3-72 Call flow to play/stop tone on a sink device**

1. Application requests Audio factory for an Audio Manager.
2. Audio factory return IAudioManager object to application.
3. On Readiness, Application requests to create a tone generator stream with streamType as TONE\_GENERATOR.
4. Application receives synchronous status which indicates if the createStream request was sent successfully.
5. Application is notified of the createStream request status (either SUCCESS or FAILED) via the application-supplied callback, with pointer to stream interface referring to IAudioToneGeneratorStream.
6. Application requests to play tone using playTone method on IAudioToneGeneratorStream.
7. Application receives synchronous status which indicates if the playTone request was sent successfully.

8. Application is notified of the playTone request status (either SUCCESS or FAILED) via the application-supplied callback.
9. Application can optionally stop the tone being played, before its duration expires.
10. Application receives synchronous status which indicates if the stopTone request was sent successfully.
11. Application is notified of the stopTone request status (either SUCCESS or FAILED) via the application-supplied callback.
12. Application requests delete audio stream using deleteStream method.
13. Application receives synchronous Status which indicates if the deleteStream request was sent successfully.
14. Application is notified of the deleteStream request status (either SUCCESS or FAILED) via the application-supplied callback.

### 3.7.10 Audio Loopback call flow

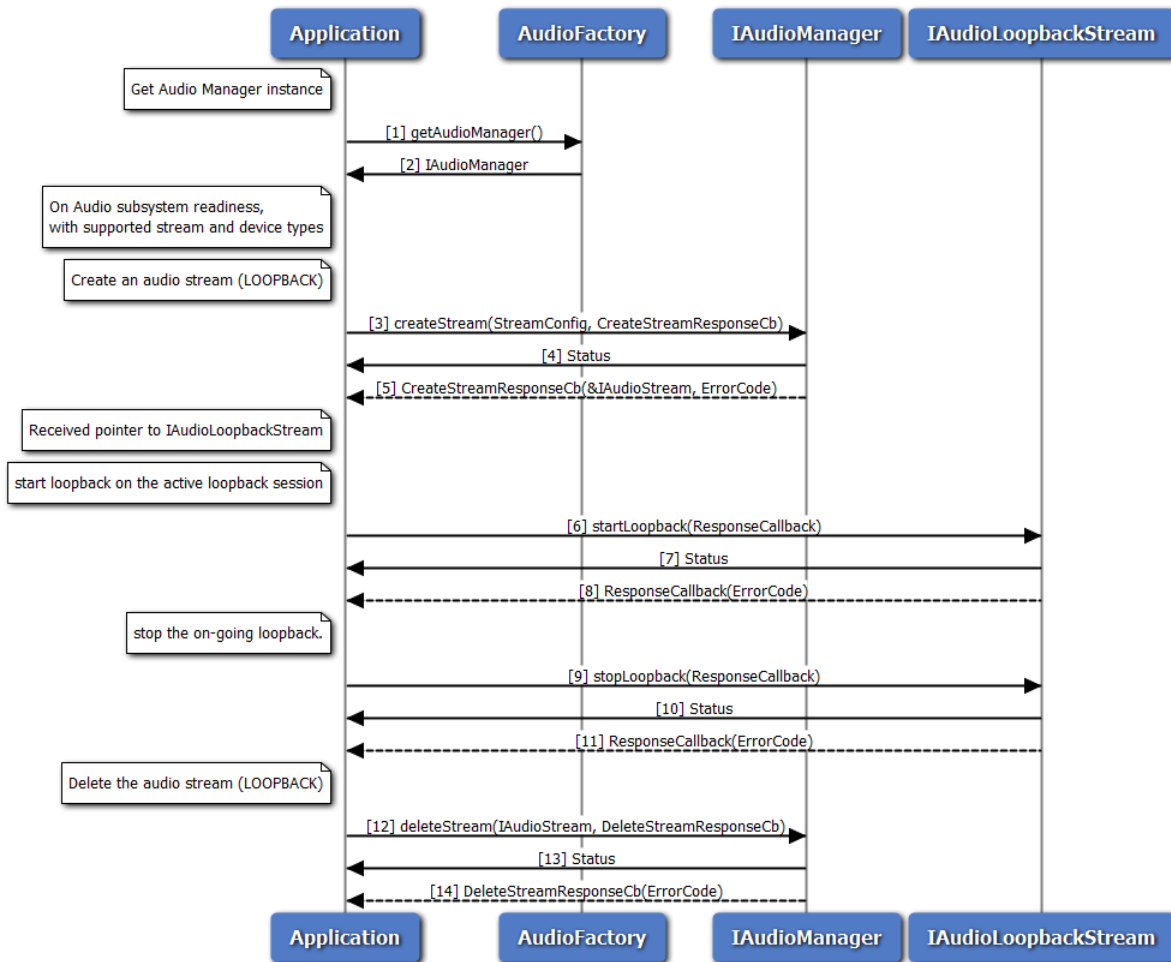


Figure 3-73 Call flow to start/stop loopback between source and sink devices

1. Application requests Audio factory for an Audio Manager.
2. Audio factory return IAudioManager object to application.
3. On Readiness, Application requests to create a loopback stream with streamType as LOOPBACK.
4. Application receives synchronous status which indicates if the createStream request was sent successfully.
5. Application is notified of the createStream request status (either SUCCESS or FAILED) via the application-supplied callback, with pointer to stream interface referring to IAudioLoopbackStream.
6. Application requests to start loopback using startLoopback method on IAudioLoopbackStream.
7. Application receives synchronous status which indicates if the startLoopback request was sent successfully.
8. Application is notified of the startLoopback request status (either SUCCESS or FAILED) via the application-supplied callback.
9. Application requests to stop loopback using stopLoopback method on IAudioLoopbackStream.
10. Application receives synchronous status which indicates if the stopLoopback request was sent successfully.
11. Application is notified of the stopLoopback request status (either SUCCESS or FAILED) via the application-supplied callback.
12. Application requests delete audio stream using deleteStream method.
13. Application receives synchronous Status which indicates if the deleteStream request was sent successfully.
14. Application is notified of the deleteStream request status(either SUCCESS or FAILED) via the application-supplied callback.

### 3.7.11 Compressed audio format playback call flow

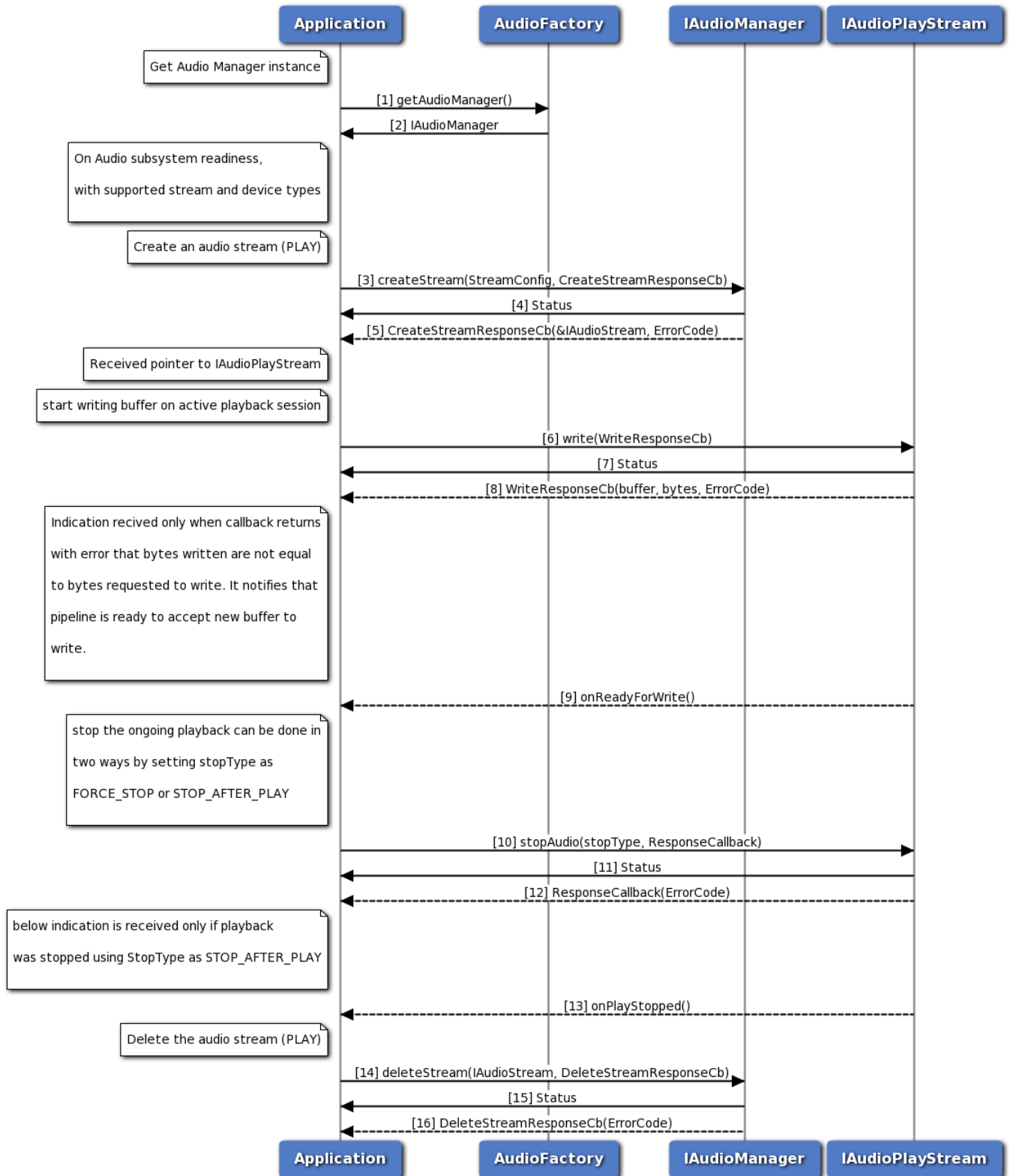


Figure 3-74 Call flow to play Compressed audio format

1. Application requests Audio factory for an Audio Manager.
2. Audio factory return IAudioManager object to application.
3. On Readiness, Application requests to create a play stream with streamType as PLAY.
4. Application receives synchronous status which indicates if the createStream request was sent successfully.
5. Application is notified of the createStream request status (either SUCCESS or FAILED) via the application-supplied callback, with pointer to stream interface referring to IAudioPlayStream.
6. Application requests to write buffer using write method on IAudioPlayStream.
7. Application receives synchronous status which indicates if the write request was sent successfully.
8. Application is notified of the write request status (either SUCCESS or FAILED) via the application-supplied callback along with number of bytes written.
9. Application is notified of when pipeline is ready to accept new buffer if callback returns with error that number of bytes written are not equal to bytes requested.
10. Application send request to stop playback using stopAudio method of IAudioPlayStream.
11. Application receives synchronous status which indicates if the stopAudio request was sent successfully.
12. Application is notified of the stopAudio request status (either SUCCESS or FAILED) via the application-supplied callback.
13. Application is notified via indication that playback is stopped if StopType is STOP\_AFTER\_PLAY.
14. Application requests delete audio stream using deleteStream method.
15. Application receives synchronous Status which indicates if the deleteStream request was sent successfully.
16. Application is notified of the deleteStream request status(either SUCCESS or FAILED) via the application-supplied callback.



### 3.7.12 Audio Transcoding Operation Callflow

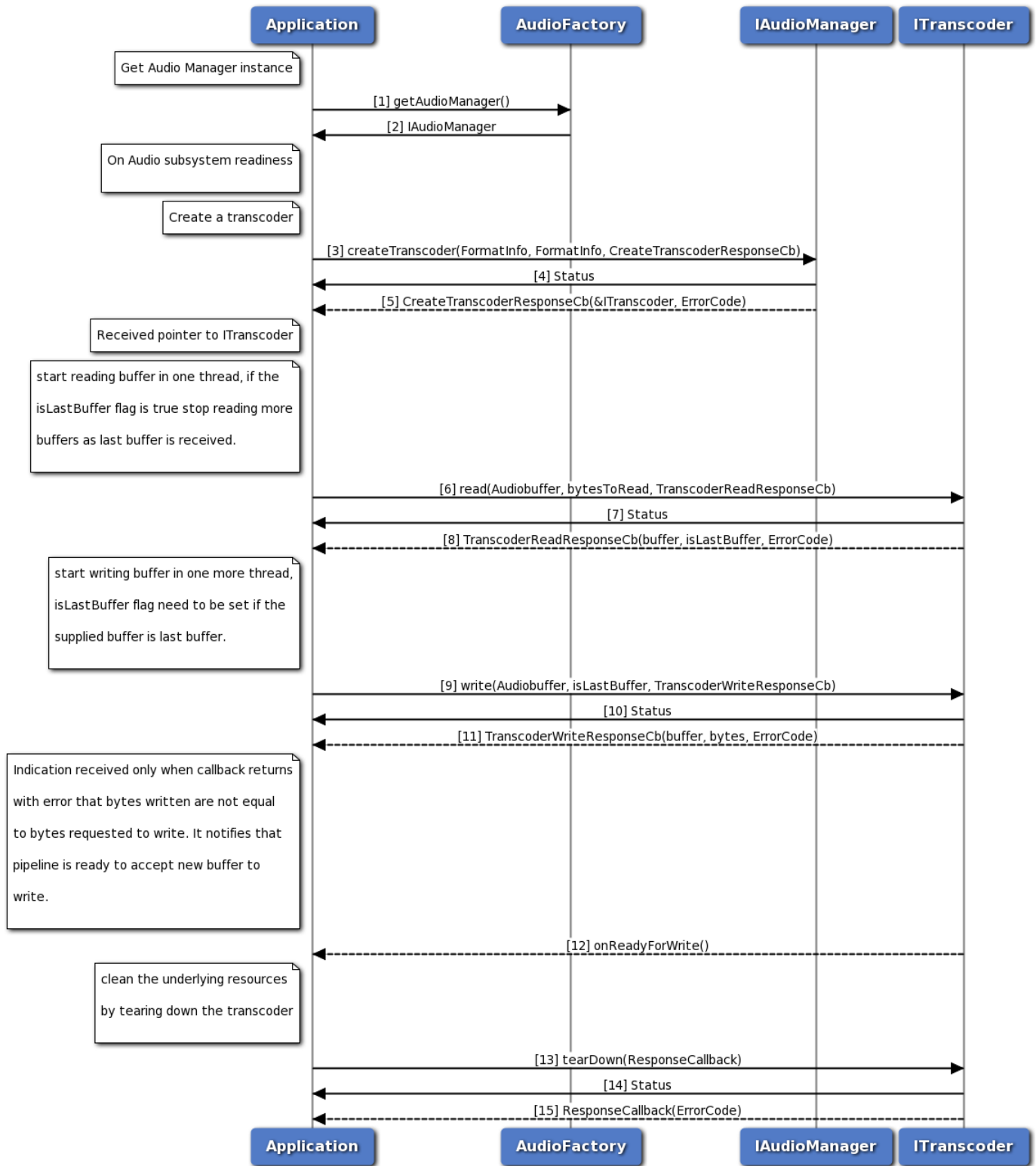


Figure 3-75 Audio Transcoding Operation Callflow

1. Application requests Audio factory for an Audio Manager.
2. Audio factory return IAudioManager object to application.
3. On Readiness, Application requests to create a transcoder.
4. Application receives synchronous status which indicates if the createTranscoder request was sent successfully.
5. Application is notified of the createTranscoder request status (either SUCCESS or FAILED) via the application-supplied callback, with pointer to transcoder interface referring to ITranscoder.
6. Application requests to read buffer using read method on ITranscoder.
7. Application receives synchronous status which indicates if the read request was sent successfully.
8. Application is notified of the read request status (either SUCCESS or FAILED) via the application-supplied callback along with isLastBuffer flag which indicates whether the buffer is last buffer to read or not.
9. Application requests to write buffer using write method on ITranscoder.
10. Application receives synchronous status which indicates if the write request was sent successfully. Application need to mark the isLastBuffer flag, whenever it is providing the last buffer to be write.
11. Application is notified of the write request status (either SUCCESS or FAILED) via the application-supplied callback along with number of bytes written.
12. Application is notified of when pipeline is ready to accept new buffer if callback returns with error that number of bytes written are not equal to bytes requested.
13. Once transcoding done, Application requests to tearDown transcoder as transcoder can not be used for multiple transcoding operations.
14. Application receives synchronous status which indicates if the tearDown request was sent successfully.
15. Application is notified of the tearDown request status (either SUCCESS or FAILED) via the application-supplied callback.

### 3.7.13 Compressed audio format playback on Voice Paths Callflow

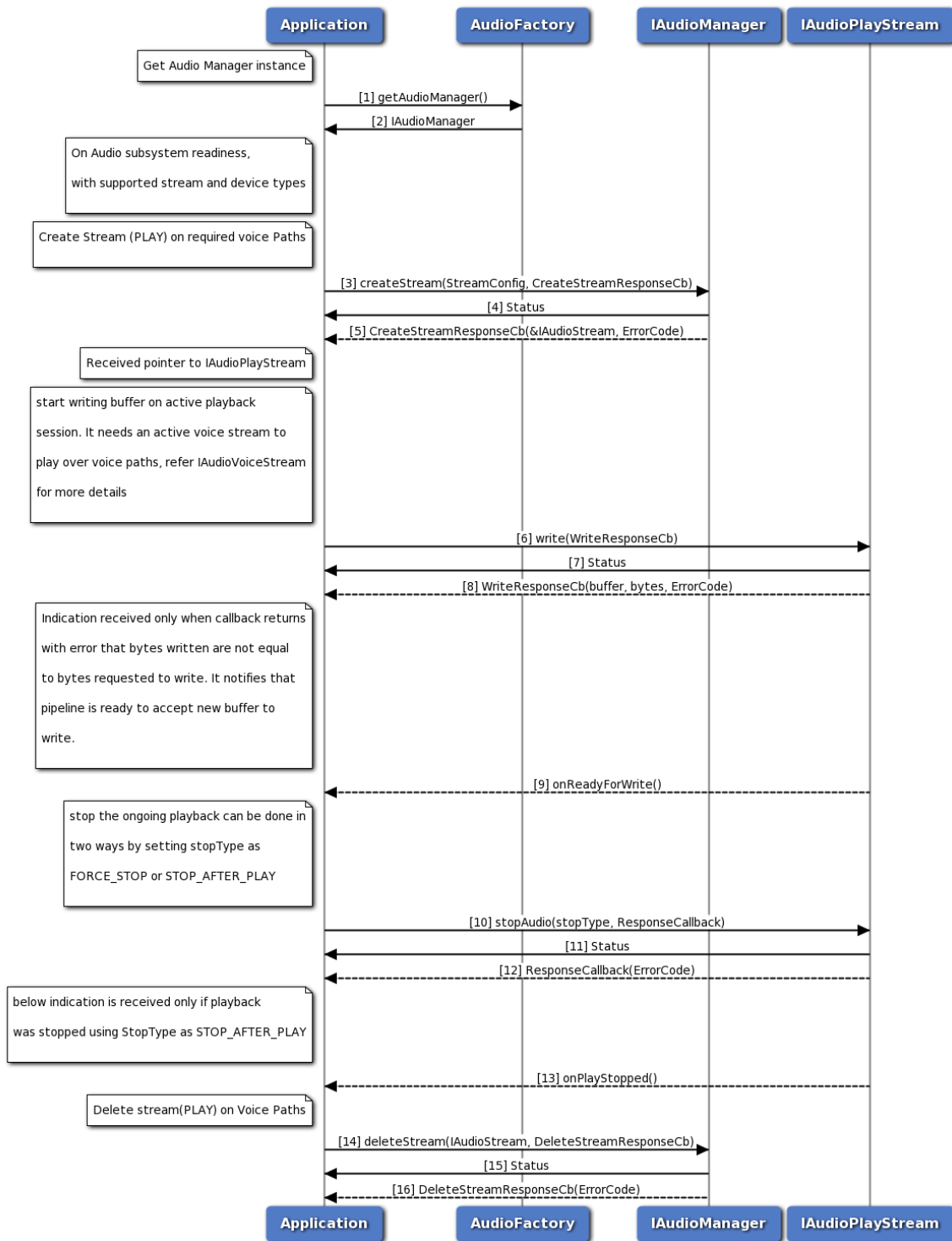


Figure 3-76 Compressed audio format playback on Voice Paths Callflow

1. Application requests Audio factory for an Audio Manager.
2. Audio factory return IAudioManager object to application.

3. On Readiness, Application requests to create a play stream with streamType as PLAY, voicePaths direction as TX or RX and no device is selected.
4. Application receives synchronous status which indicates if the createStream request was sent successfully.
5. Application is notified of the createStream request status (either SUCCESS or FAILED) via the application-supplied callback, with pointer to stream interface referring to IAudioPlayStream.
6. Application requests to write buffer using write method on IAudioPlayStream. It needs an active voice session to play over voice paths, refer IAudioVoiceStream for more details on how to create voice stream.
7. Application receives synchronous status which indicates if the write request was sent successfully.
8. Application is notified of the write request status (either SUCCESS or FAILED) via the application-supplied callback along with number of bytes written.
9. Application is notified of when pipeline is ready to accept new buffer if callback returns with error that number of bytes written are not equal to bytes requested.
10. Application send request to stop playback using stopAudio method of IAudioPlayStream.
11. Application receives synchronous status which indicates if the stopAudio request was sent successfully.
12. Application is notified of the stopAudio request status (either SUCCESS or FAILED) via the application-supplied callback.
13. Application is notified via indication that playback is stopped if StopType is STOP\_AFTER\_PLAY.
14. Application requests delete audio play stream using deleteStream method.
15. Application receives synchronous Status which indicates if the deleteStream request was sent successfully.
16. Application is notified of the deleteStream request status(either SUCCESS or FAILED) via the application-supplied callback.

### 3.8 Thermal manager call flow

Thermal manager provides APIs to get list of thermal zones and cooling devices. It also contains APIs to get a particular thermal zone and a particular cooling device details with the given Id.

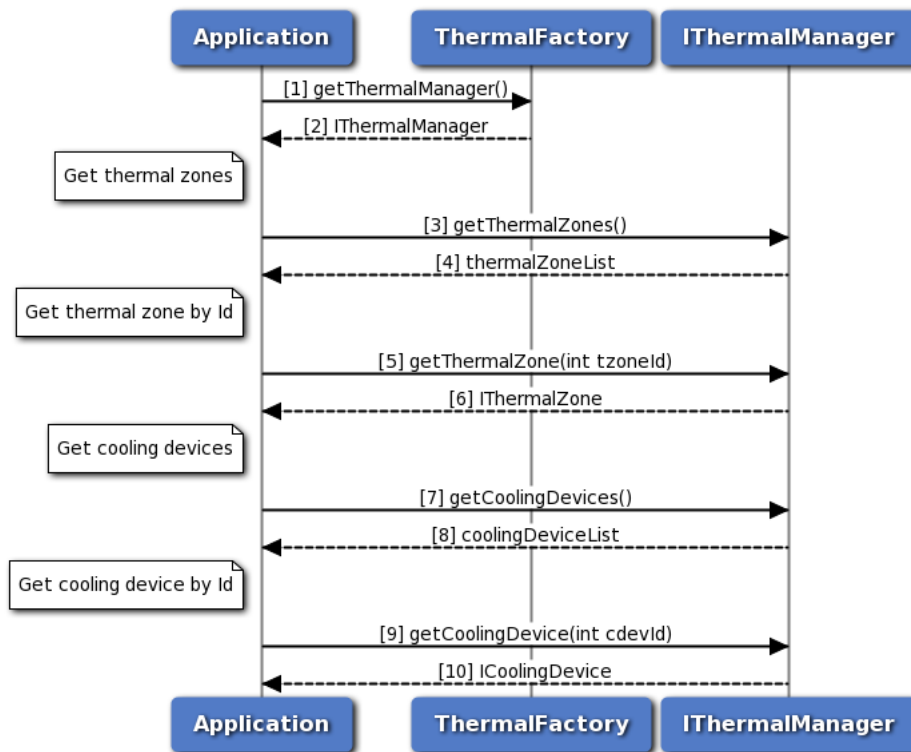


Figure 3-77 Thermal manager call flow

1. Application requests Thermal factory for Thermal Manager.
2. Thermal factory returns IThermalManager object to application.
3. Application sends request to get all thermal zones using IThermalManager object.
4. Thermal manager returns the list of thermal zones to the application.
5. Application requests for a particular thermal zone details by mentioning the thermal zone Id.
6. Application receives thermal zone details with the given Id from thermal manager.
7. Application sends request to get all cooling devices using IThermalManager object.
8. Thermal manager returns the list of cooling devices to the application.
9. Application requests for a particular cooling device details by passing the cooling device Id.
10. Thermal Manager sends cooling device details with the given Id to the application.

### 3.9 Thermal shutdown management

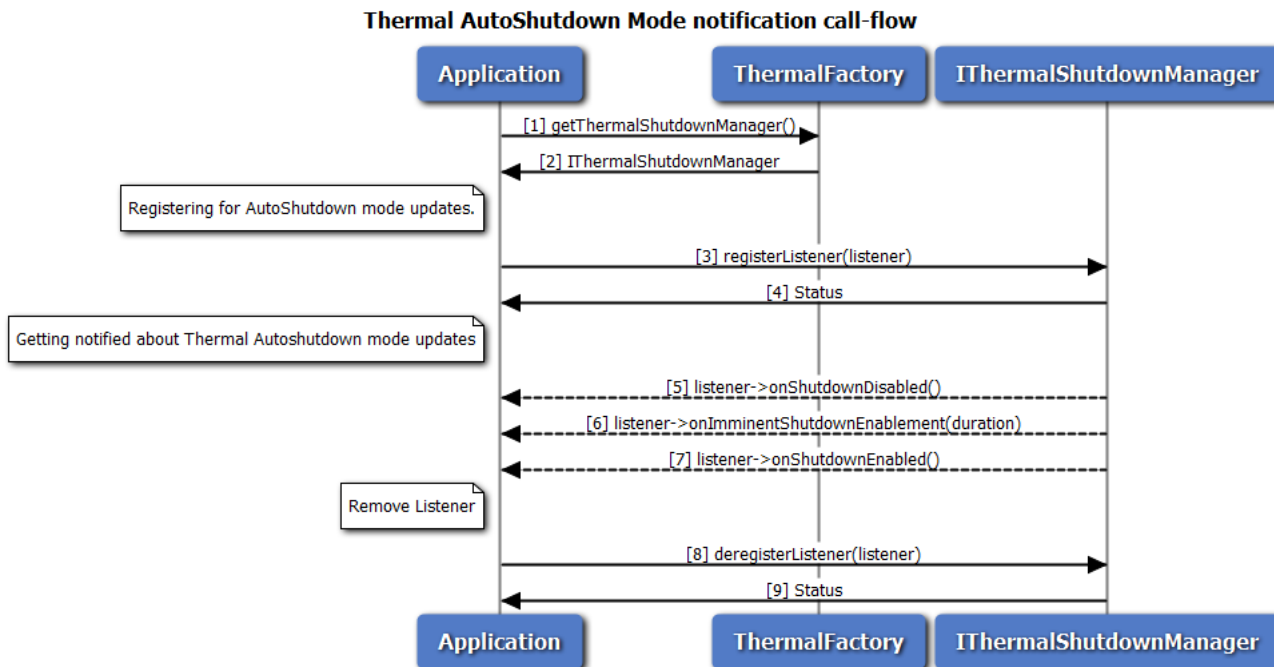
Thermal shutdown manager provides APIs to set/get auto thermal shutdown modes. It also has listener interface for notifications. Application will get the Thermal-shutdown manager object from thermal factory.

The application can register a listener for updates in thermal auto shutdown modes and its management service status. Also there is provision to set the desired thermal auto-shutdown mode.

When application is notified of service being unavailable, the thermal auto-shutdown mode updates are inactive. After service becomes available, the existing listener registrations will be maintained.

As a reference, auto-shutdown management in an eCall application is described in the below sections.

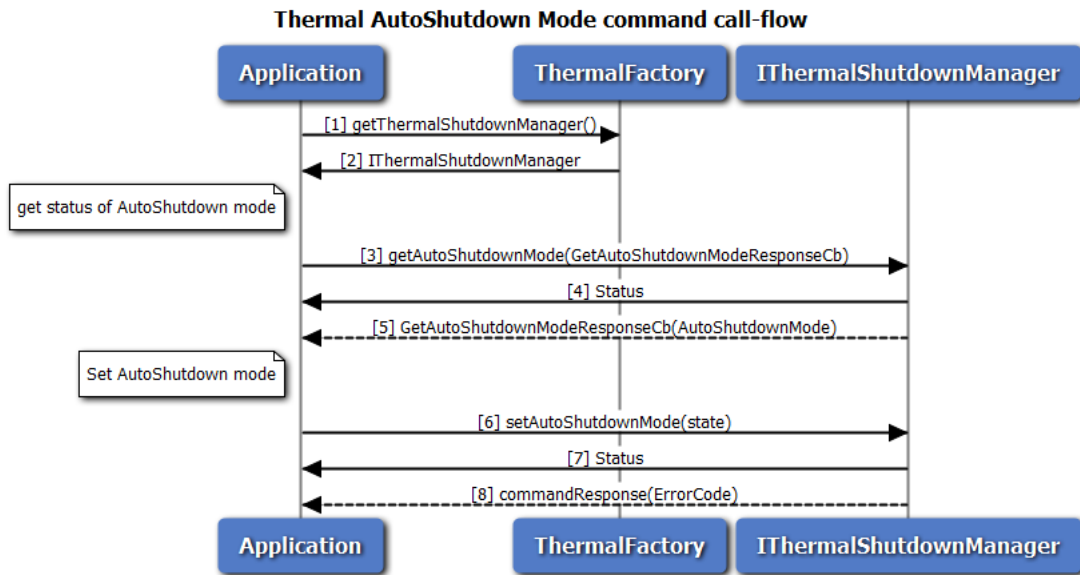
### 3.9.1 Call flow to register/remove listener for Thermal auto-shutdown mode updates.



**Figure 3-78 Call flow to register/remove listener for Thermal shutdown manager**

1. Application requests thermal factory for Thermal Shutdown Manager.
2. Thermal factory returns IThermalShutdownManager object using which application will register or remove a listener.
3. Application can register a listener for getting notifications on Thermal auto-shutdown mode updates.
4. Status of register listener i.e. either SUCCESS or FAILED will be returned to the application.
5. Application receives a notification that thermal auto-shutdown mode is disabled.
6. Application receives a notification that thermal auto-shutdown mode is going to be enabled soon. The exact duration is also received as part of notification.
7. Application receives a notification that thermal auto-shutdown mode is enabled.
8. Application can remove listener.
9. Status of remove listener i.e. either SUCCESS or FAILED will be returned to the application.

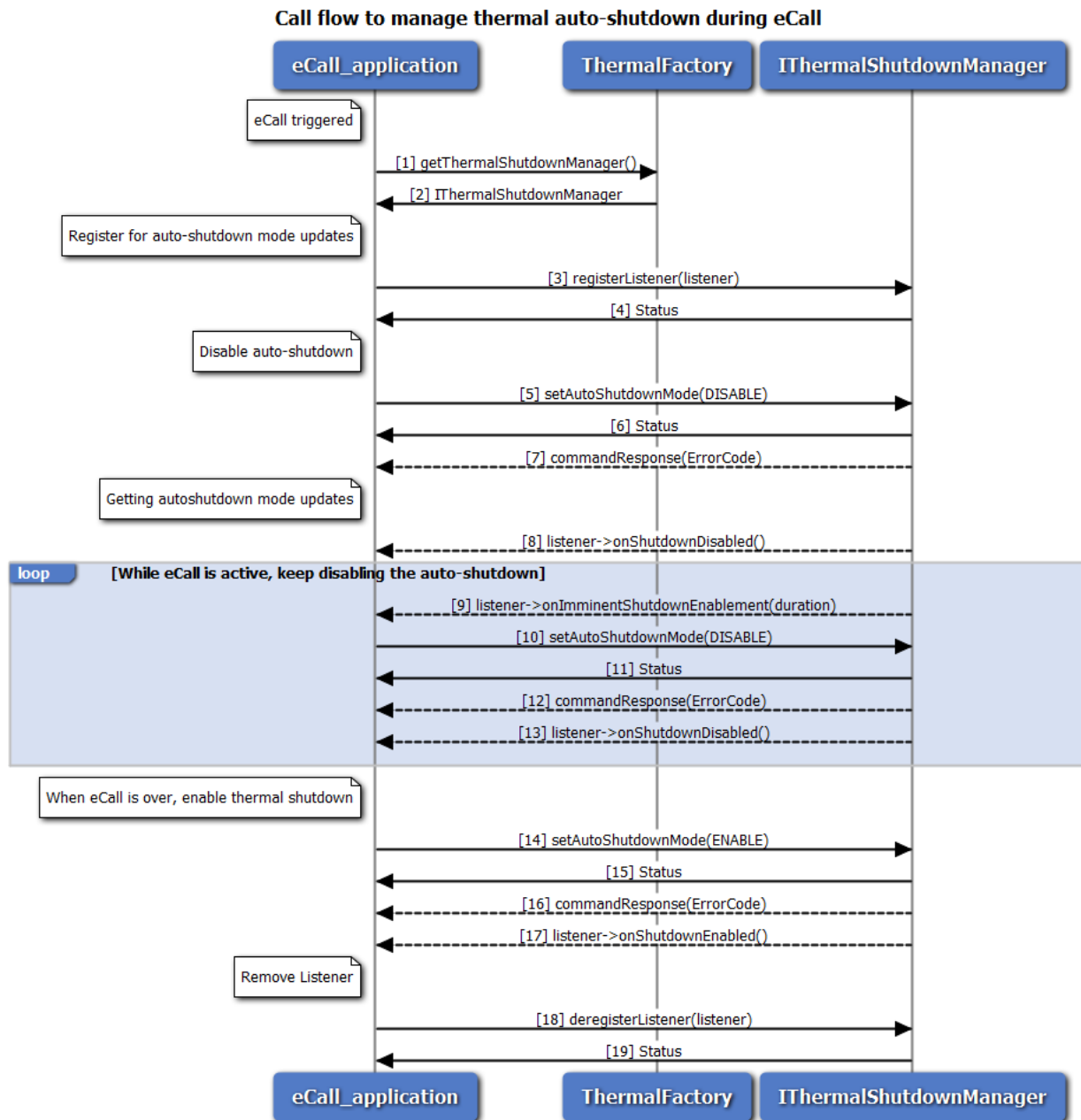
### 3.9.2 Call flow to set/get the Thermal auto-shutdown mode



**Figure 3-79 Call flow to set/get the Thermal auto-shutdown mode**

1. Application requests thermal factory for Thermal Shutdown Manager object using which application will set/get the thermal auto-shutdown mode.
2. Thermal factory returns `IThermalShutdownManager` object.
3. Application can query the thermal auto-shutdown mode.
4. Application receives synchronous status which indicates if the request was sent successfully.
5. Application receives the auto-shutdown mode asynchronously.
6. Application can set the thermal auto-shutdown mode to ENABLE or DISABLE.
7. Application receives synchronous status which indicates if the request was sent successfully.
8. Optionally, the response to `setAutoShutdownMode` request can be received by the application.

### 3.9.3 Call flow to manage thermal auto-shutdown from an eCall application.



**Figure 3-80 Call flow to manage thermal auto-shutdown from an eCall application**

1. When eCall is triggered, application requests thermal factory for Thermal Shutdown Manager.
2. Thermal factory returns IThermalShutdownManager object.
3. Application can register a listener for getting notifications on Thermal auto-shutdown mode updates.
4. Status of register listener i.e. either SUCCESS or FAILED will be returned to the application.
5. Application disables auto-shutdown using setAutoShutdownMode API, to prevent a possible thermal

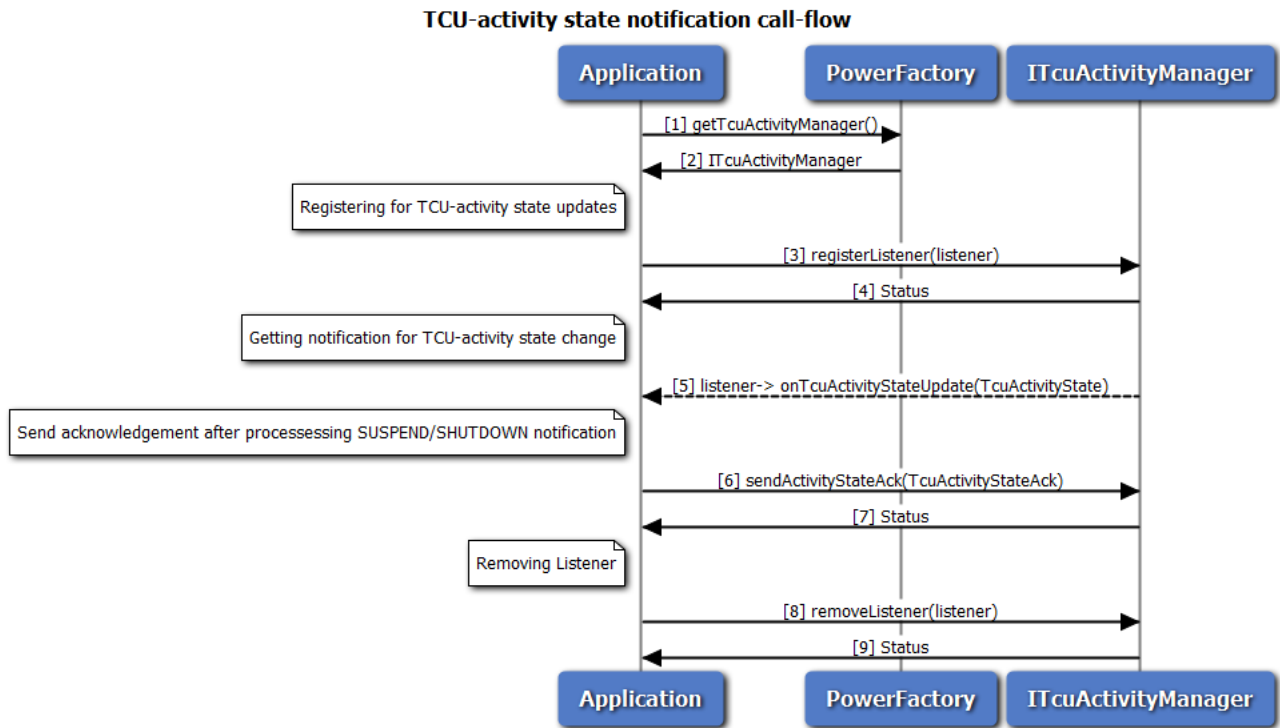


- auto-shutdown during eCall.
6. Application receives synchronous status which indicates if the request was sent successfully.
  7. Optionally, the response to setAutoShutdownMode request can be received by the application.
  8. Application receives a notification that thermal auto-shutdown mode is disabled.
  9. Application receives an imminent auto-shutdown enable notification and system will attempt to enable auto-shutdown after a certain period. This notification is received if application does not enable auto-shutdown due to an active eCall.
  10. If the eCall is still active, the application disables auto-shutdown before it gets enabled automatically.
  11. Application receives synchronous status which indicates if the request was sent successfully.
  12. Optionally, the response to setAutoShutdownMode request can be received by the application.
  13. Application receives a notification that thermal auto-shutdown mode is disabled. Steps 9 to 13 are repeated as long as the eCall is active.
  14. When the eCall is completed, the application immediately enables auto-shutdown using setAutoShutdownMode API.
  15. Application receives synchronous status which indicates if the request was sent successfully.
  16. Optionally, the response to setAutoShutdownMode request can be received by the application.
  17. Application receives a notification that thermal auto-shutdown mode is enabled.
  18. Application can remove listener.
  19. Status of remove listener i.e. either SUCCESS or FAILED will be returned to the application.

### 3.10 TCU Activity Management

Application will get the TCU-activity manager object from power factory. The application can register a listener for updates on TCU-activity state and its management service status. The application can also set the system to a desired activity state. When the application is notified about the service being unavailable, the TCU-activity state notifications will be inactive. After the service becomes available, the existing listener registrations will be maintained.

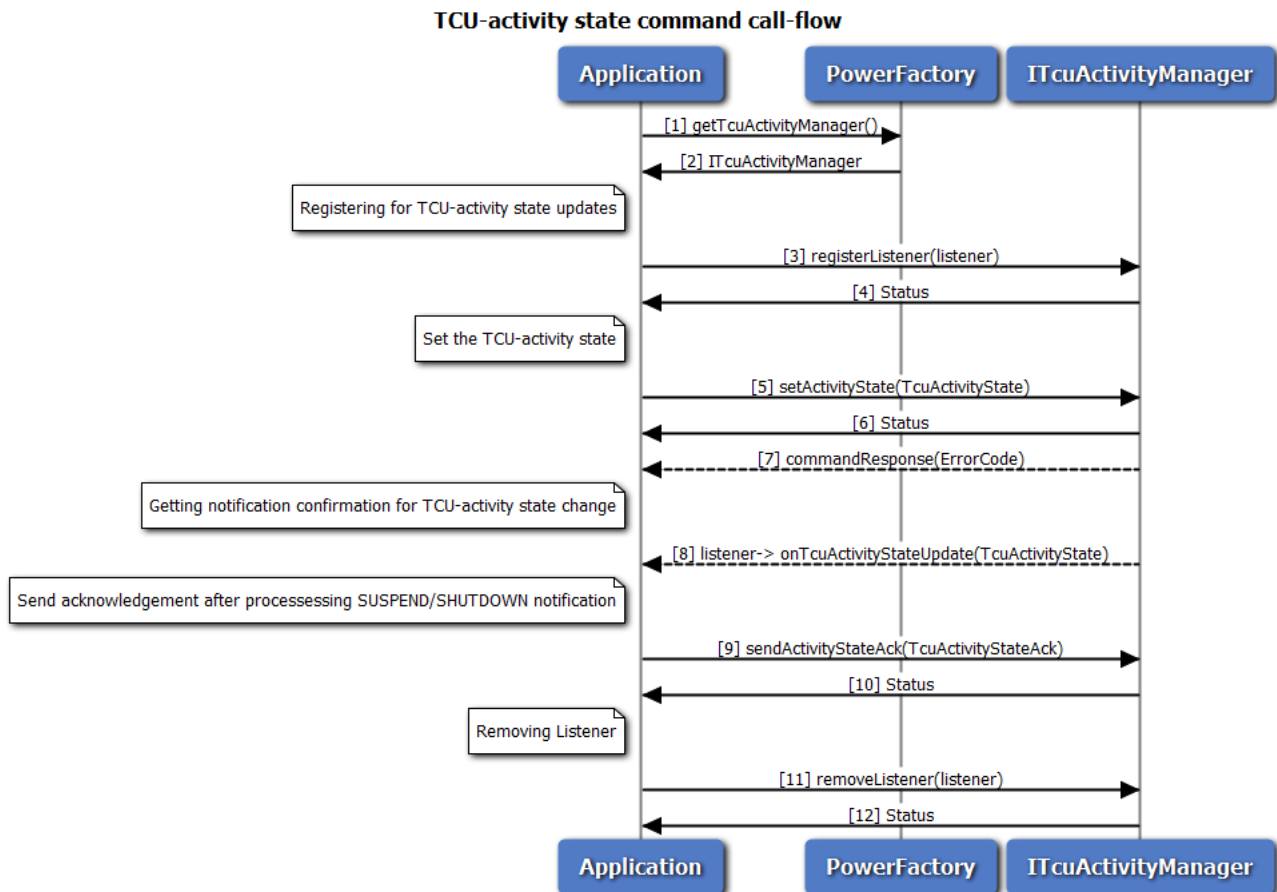
### 3.10.1 Call flow to register/remove listener for TCU-activity manager



**Figure 3-81 Call flow to register/remove listener for TCU-activity manager**

1. Application requests power factory for TCU-activity manager object.
2. Power factory returns ITcuActivityManager object using which application will register or remove a listener.
3. Application can register a listener for getting notifications on TCU-activity state updates.
4. Status of register listener i.e. either SUCCESS or FAILED will be returned to the application.
5. Application will get TCU-activity state notifications like SUSPEND, RESUME and SHUTDOWN.
6. Application will send one(despite multiple listeners) acknowledgement, after processing(save any required information) SUSPEND/SHUTDOWN notifications. This indicates the readiness of application for state-transition. However the TCU-activity management service doesn't wait for acknowledgement indefinitely, before performing the state transition.
7. Application receives synchronous status which indicates if the acknowledgement was sent successfully.
8. Application can remove listener.
9. Status of remove listener i.e. either SUCCESS or FAILED will be returned to the application.

### 3.10.2 Call flow to set the TCU-activity state



**Figure 3-82 Call flow to set the TCU-activity state**

1. Application requests power factory for TCU-activity manager object.
2. Power factory returns ITcuActivityManager object using which application will set the TCU-activity state.
3. Application can register a listener for getting notifications on TCU-activity state.
4. Status of register listener i.e. either SUCCESS or FAILED will be returned to the application
5. Application can set the TCU-activity state to SUSPEND, RESUME or SHUTDOWN.
6. Application receives synchronous status which indicates if the request was sent successfully.
7. Optionally, the response to setActivityState request can be received by the application.
8. Application waits for TCU-activity state update to confirm the state change.
9. Application will send one(despite multiple listeners) acknowledgement, after processing(save any required information) SUSPEND/SHUTDOWN notifications. This indicates the readiness of application for state-transition. However the TCU-activity management service doesn't wait for acknowledgement indefinitely, before performing the state transition.
10. Application receives synchronous status which indicates if the acknowledgement was sent

successfully.

11. Application can remove listener.
12. Status of remove listener i.e. either SUCCESS or FAILED will be returned to the application.

### 3.11 Remote SIM call flow

Application will get the remote SIM manager object from phone factory. The application must register a listener to receive commands/messages from the modem to send to the SIM. After sending the connection available message, a onCardConnect() notification tells the application to connect to the SIM and perform an Answer to Reset. After sending the card reset message (with the AtR bytes), APDU messages will begin to be sent/received.

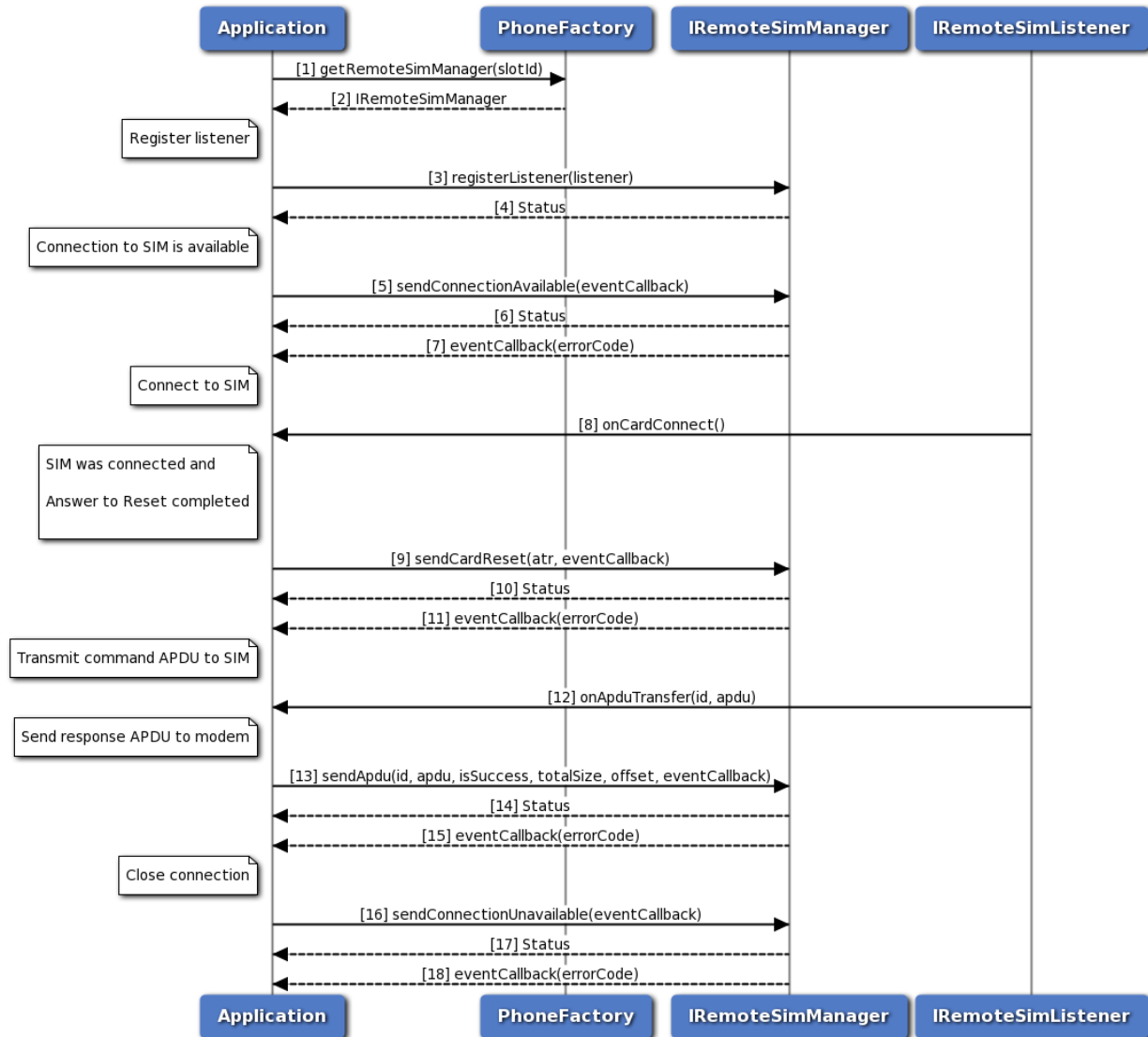


Figure 3-83 Remote SIM call flow

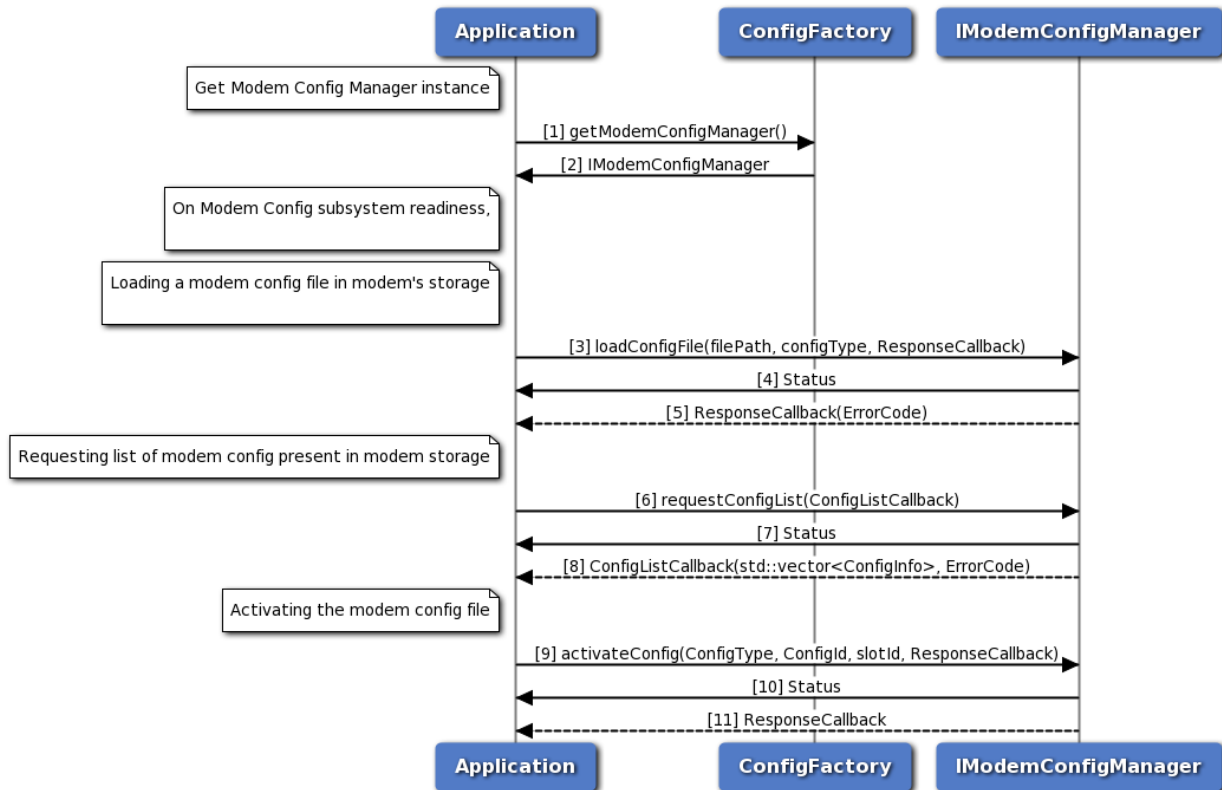
1. Application requests remote SIM manager object from phone factory, specifying a slot id.

2. Phone factory returns IRemoteSimManager object.
3. Application registers a listener to receive commands/messages from the modem to send to the SIM.
4. Status of register listener i.e. either SUCCESS or FAILED will be returned to the application.
5. Application sends a connection available message indicating that a SIM is available for use.
6. Status of send connection available i.e. either SUCCESS or FAILED will be returned to the application.
7. Optionally, the response to send connection available request can be received by the application.
8. Application will receive a card connect notification by the listener.
9. After the application successfully connects to the SIM and requests an AtR, it sends a card reset message with the AtR bytes.
10. Status of send card reset i.e. either SUCCESS or FAILED will be returned to the application.
11. Optionally, the response to send card reset request can be received by the application.
12. Application will receive an APDU transfer notification by the listener (with APDU message id).
13. After forwarding the APDU transfer to the SIM and receiving the response, application will send APDU response.
14. Status of send APDU i.e. either SUCCESS or FAILED will be returned to the application.
15. Optionally, the response to send APDU request can be received by the application.
16. To close the connection, application will send connection unavailable message.
17. Status of send connection unavailable i.e. either SUCCESS or FAILED will be returned to the application.
18. Optionally, the response to send connection unavailable can be received by the application.

## 3.12 Modem Config Call Flow

Modem Config manager provides APIs to request all configs from modem, load/delete modem config files from modem's storage, activate/deactivate a modem config file, get the active config details, set and get auto config selection mode. It also has listener interface for notifications for config activation update status. Application will get the Modem Config manager object from config factory. The application can register a listener for updates regarding modem config activation.

### 3.12.1 Call flow to load and activate a modem config file.

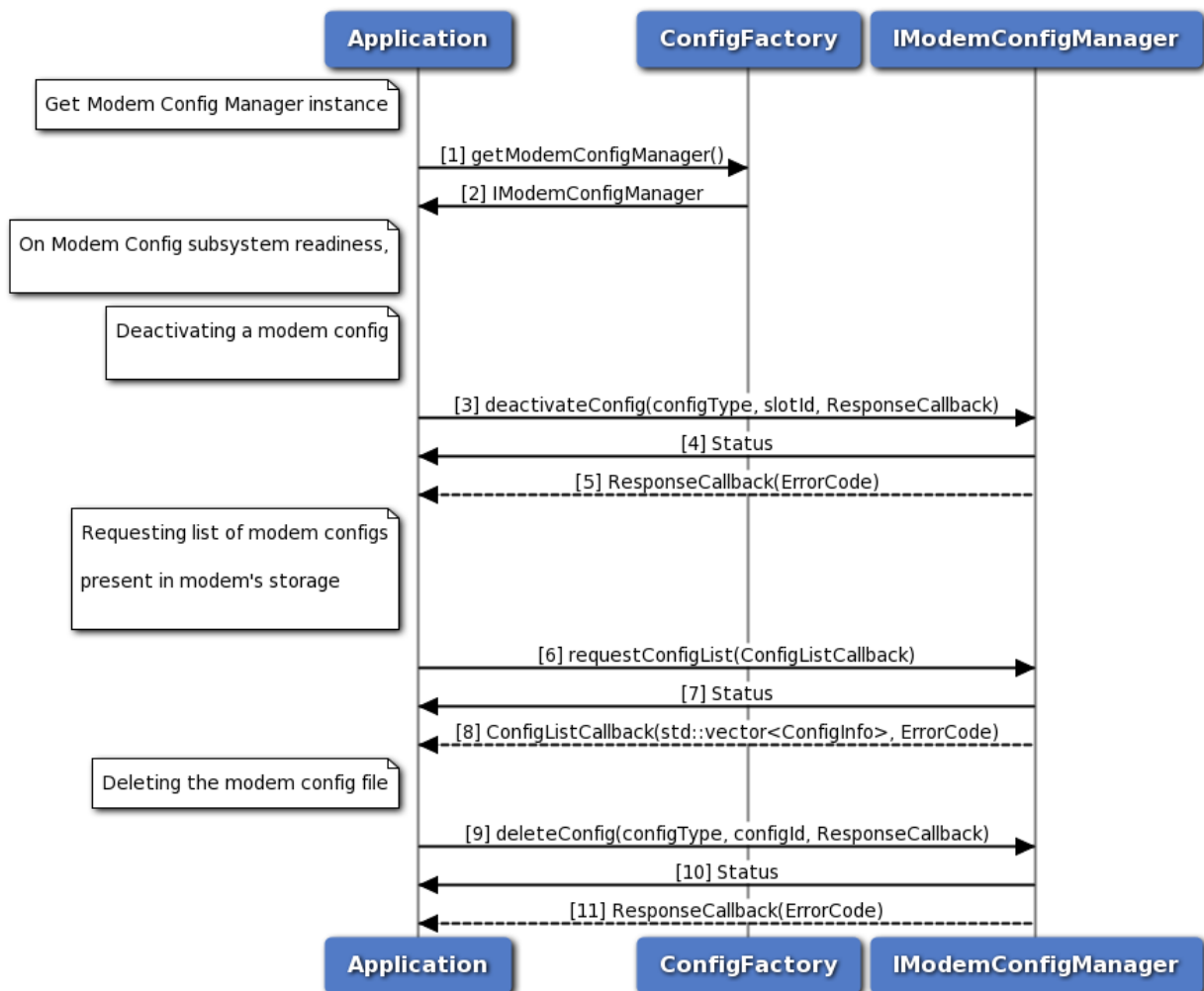


**Figure 3-84 Modem Config load and activate call flow**

1. Application requests modem config manager object from config factory.
2. Config factory returns IModemConfigManager object.
3. Application sends a request to load config file in modem's storage.
4. Application receives synchronous Status which indicates if the request to load config file was sent successfully.
5. Application is notified of the Status of the loadConfigFile request (either SUCCESS or FAILED) via the application-supplied callback.
6. Application sends a request to get list of all modem configs from modem's storage.
7. Application receives synchronous Status which indicates if the request to get config list was sent successfully.
8. Application is notified of the Status of the requestConfigList request (either SUCCESS or FAILED) via the application-supplied callback along with list of modem configs.
9. Application sends a request to activate config file.
10. Application receives synchronous Status which indicates if the request to activate config file was sent successfully.

- Application is notified of the Status of the activateConfig request (either SUCCESS or FAILED) via the application-supplied callback.

### 3.12.2 Call flow to deactivate and delete a modem config file.

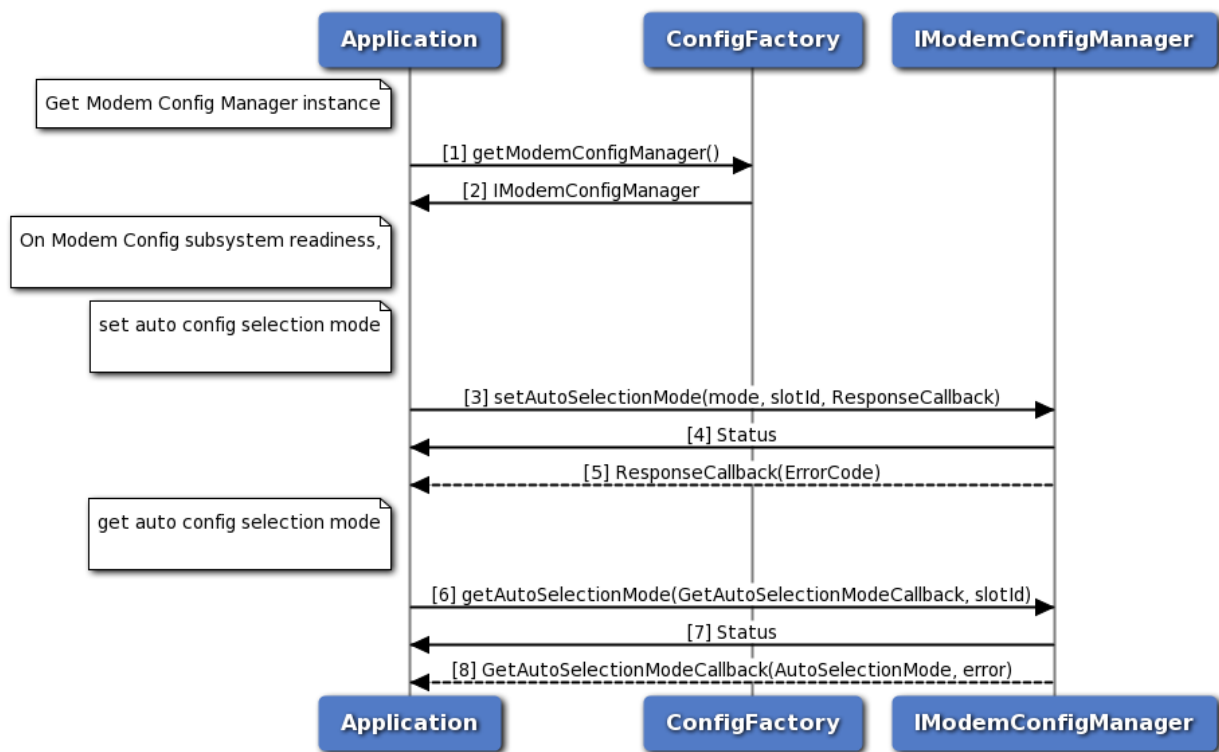


**Figure 3-85 Modem Config deactivate and delete Call Flow**

- Application requests modem config manager object from config factory.
- Config factory returns IModemConfigManager object.
- Application sends a request to deactivate config file.
- Application receives synchronous Status which indicates if the request to deactivate config file was sent successfully.
- Application is notified of the Status of the deactivateConfig request (either SUCCESS or FAILED) via the application-supplied callback.
- Application sends a request to get list of all modem configs from modem's storage.

7. Application receives synchronous Status which indicates if the request to get config list was sent successfully.
8. Application is notified of the Status of the requestConfigList request (either SUCCESS or FAILED) via the application-supplied callback along with list of modem configs.
9. Application sends a request to delete config file.
10. Application receives synchronous Status which indicates if the request to delete config file was sent successfully.
11. Application is notified of the Status of the deleteConfig request (either SUCCESS or FAILED) via the application-supplied callback.

### 3.12.3 Call flow to set and get config auto selection mode



**Figure 3-86 Modem Config get and set Auto Selection Mode Call Flow**

1. Application requests modem config manager object from config factory.
2. Config factory returns IModemConfigManager object.
3. Application sends a request to set config auto selection mode.
4. Application receives synchronous Status which indicates if the request to set config auto selection mode was sent successfully.
5. Application is notified of the Status of the request setAutoSelectionMode (either SUCCESS or FAILED) via the application-supplied callback.



6. Application sends a request to get config auto selection mode.
7. Application receives synchronous Status which indicates if the request to get config auto selection mode was sent successfully.
8. Application is notified of the Status of the request setAutoSelectionMode (either SUCCESS or FAILED) via the application-supplied callback, along with mode and slot id.

### **3.13 WLAN**

### 3.13.1 Call flow to Modify WLAN Configuration

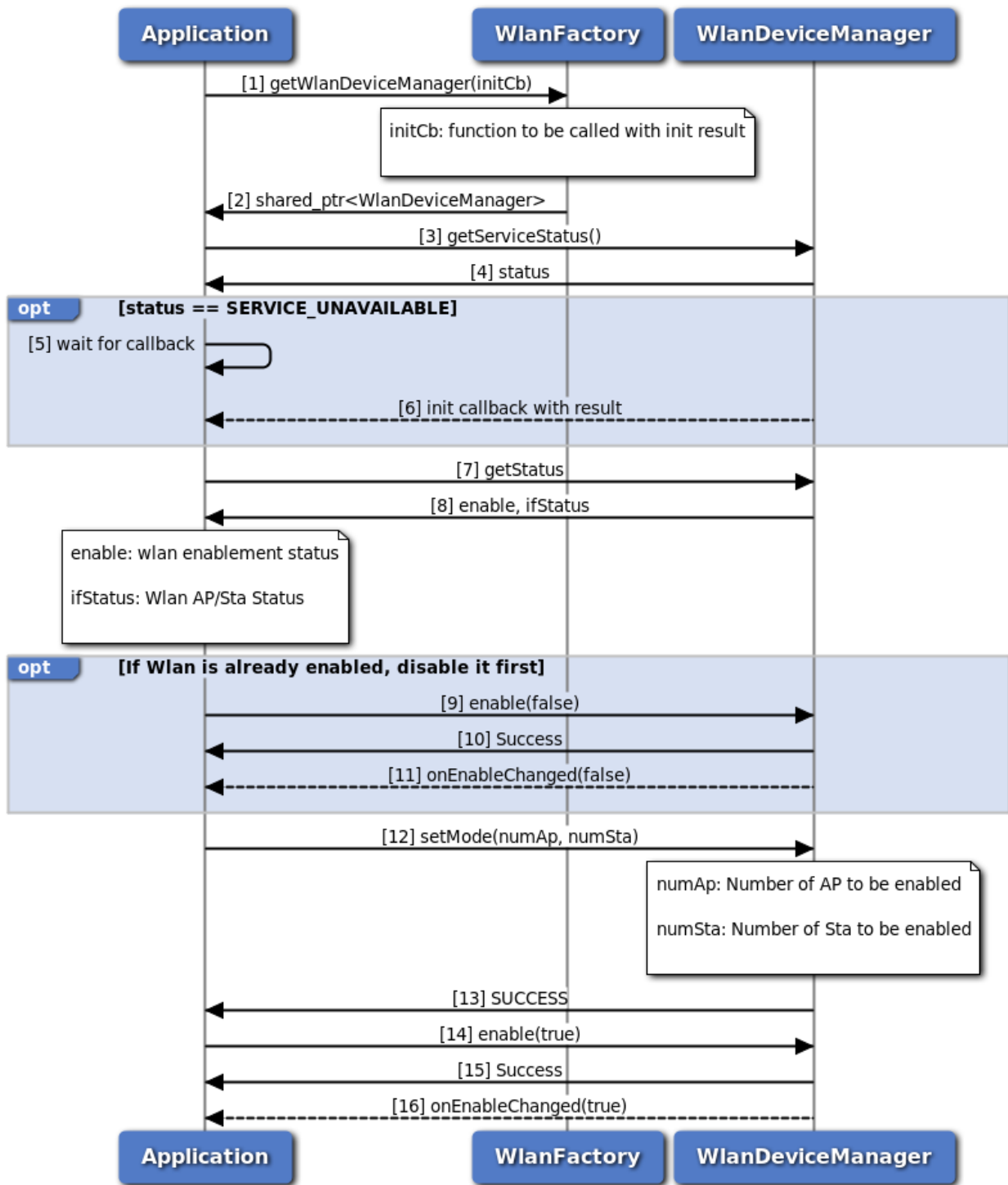
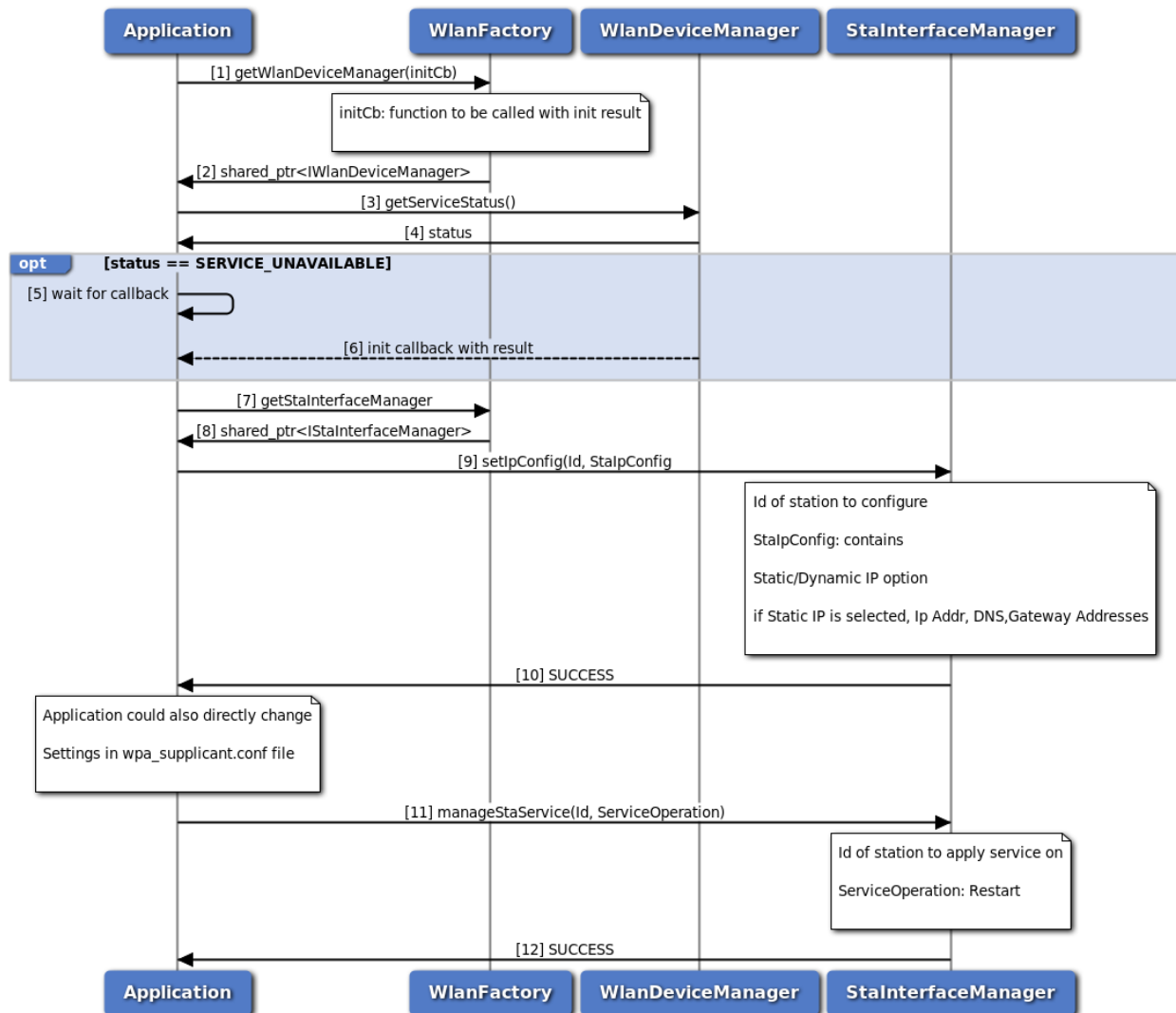


Figure 3-87 Modify WLAN Configuration Call Flow

1. Application requests IWlanDeviceManager object from Wlan factory.
2. Wlan factory returns shared pointer to IWlandeviceManager to the application.

3. Application can use `IWlanDeviceManager::getServiceStatus` to determine if the system is ready.
4. The application receives the Status i.e. either `SERVICE_AVAILABLE` or `SERVICE_UNAVAILABLE` to indicate whether sub-system is ready or not.
  - (a) If subsystem is not ready, then the application could wait for callback provided in step 1 for subsystem initialization status.
  - (b) Application provided callback is invoked with subsystem status (`SERVICE_AVAILABLE/SERVICE_FAILED`).
5. On Readiness, application calls `IWlanDeviceManager::getStatus` to get Wlan enablement status.
6. Application receives Wlan enablement status.
  - (a) If Wlan is enabled, application calls `IWlanDeviceManager::enable(false)` to disable Wlan.
  - (b) Application receives wlan disablement response and wait for Wlan status indication.
  - (c) Application receives Wlan status indication `IWlanDeviceManager::onEnableChanged(false)` to indicate Wlan is disabled.
7. Application sets desired configuration using `IWlanDeviceManager::setMode` to set desired number of access points and stations to be enabled.
8. Application receives response.
9. Application calls `IWlanDeviceManager::enable(true)` to enable Wlan.
10. Application receives wlan enablement response and wait for Wlan status indication.
11. Application receives Wlan status indication `IWlanDeviceManager::onEnableChanged(true)` to indicate Wlan is enabled.

### 3.13.2 Call flow to Modify WLAN Station Configuration



**Figure 3-88 Modify WLAN Station Configuration Call Flow**

1. Application requests IWlanDeviceManager object from Wlan factory.
2. Wlan factory returns shared pointer to IWlandeviceManager to the application.
3. Application can use IWlanDeviceManager::getServiceStatus to determine if the system is ready.
4. The application receives the Status i.e. either SERVICE\_AVAILABLE or SERVICE\_UNAVAILABLE to indicate whether sub-system is ready or not.
  - (a) If subsystem is not ready, then the application could wait for callback provided in step 1 for subsystem initialization status.
  - (b) Application provided callback is invoked with subsystem status (SERVICE\_AVAILABLE/SERVICE\_FAILED).
5. On Readiness, requests IStaInterfaceManager object from Wlan factory.

6. Wlan factory returns shared pointer to IStaInterfaceManager to the application.
7. Application calls IStaInterfaceManager::setIpConfig to set desired IP configurations.
8. Application receives response set IP configuration.
9. Application calls IStaInterfaceManager::managerStaService to restart wpa\_supplicant daemon.
10. Application receives response to restart wpa\_supplicant daemon and station configuration shall be active at this stage.

### 3.13.3 Call flow to Modify WLAN Access Point Configuration

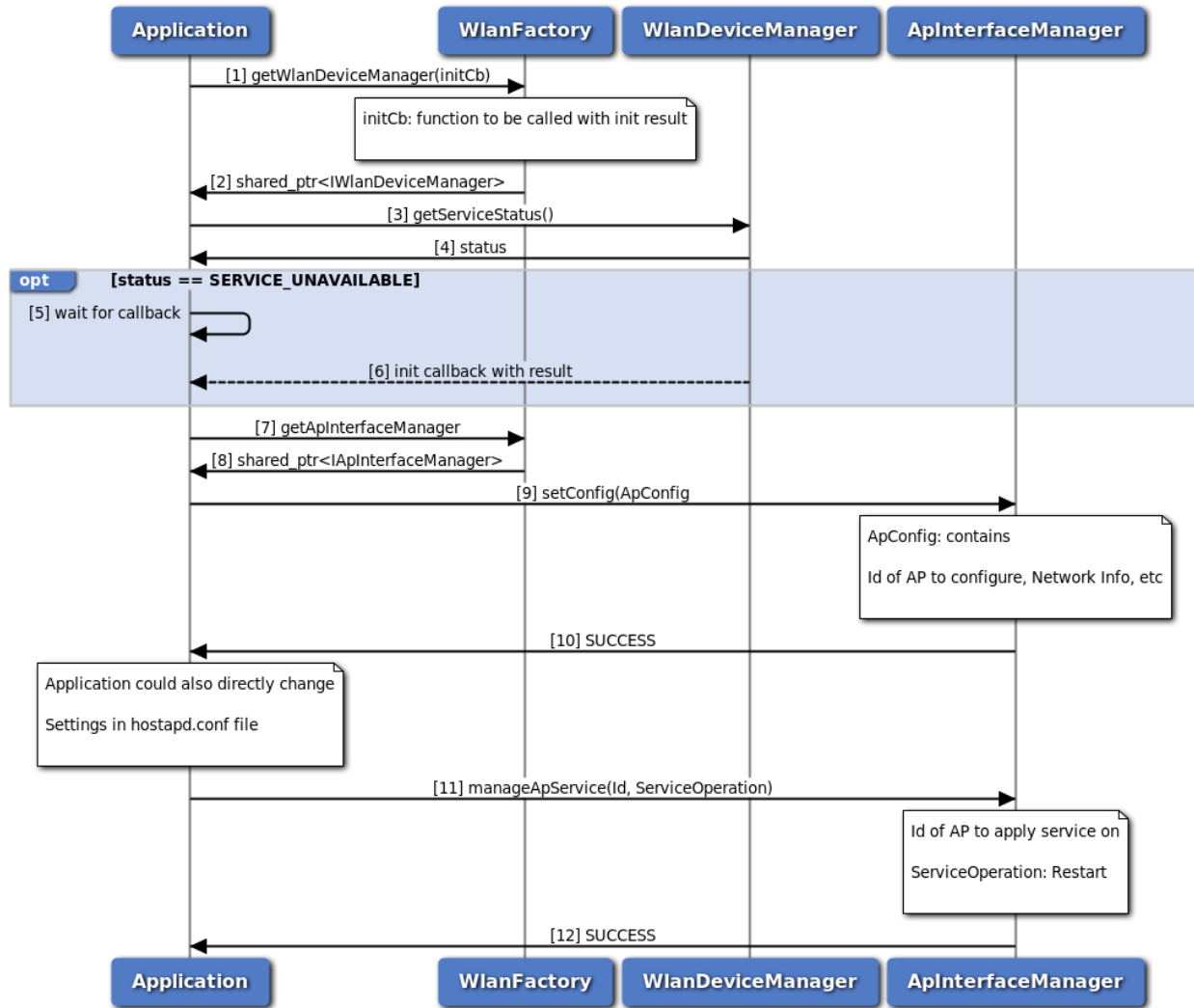


Figure 3-89 Modify WLAN Access Point Configuration Call Flow

1. Application requests IWlanDeviceManager object from Wlan factory.
2. Wlan factory returns shared pointer to IWlanDeviceManager to the application.
3. Application can use IWlanDeviceManager::getServiceStatus to determine if the system is ready.

4. The application receives the Status i.e. either SERVICE\_AVAILABLE or SERVICE\_UNAVAILABLE to indicate whether sub-system is ready or not.
  - (a) If subsystem is not ready, then the application could wait for callback provided in step 1 for subsystem initialization status.
  - (b) Application provided callback is invoked with subsystem status (SERVICE\_AVAILABLE/SERVICE\_FAILED).
5. On Readiness, requests IApInterfaceManager object from Wlan factory.
6. Wlan factory returns shared pointer to IApInterfaceManager to the application.
7. Application calls IApInterfaceManager::setConfig to set desired IP configurations.
8. Application receives response set configuration.
9. Application calls IApInterfaceManager::managerApService to restart hostapd daemon.
10. Application receives response to restart hostapd daemon and access point configuration shall be active at this stage.

# 4 Deprecated List

---

**Global `telux::cv2x::ICv2xListener::onStatusChanged (Cv2xStatus status)`**

use `onStatusChanged(Cv2xStatusEx status)`

**Global `telux::cv2x::ICv2xRadio::getCapabilities () const =0`**

Use `requestCapabilities()` API

**Global `telux::cv2x::ICv2xRadioListener::onSpsOffsetChanged (int spsId, MacDetails details)`**

use `onSpsSchedulingChanged`

**Global `telux::cv2x::ICv2xRadioListener::onStatusChanged (Cv2xStatus status)`**

use `onStatusChanged` in `Cv2xListener`

**Global `telux::cv2x::ICv2xRadioListener::onStatusChanged (Cv2xStatusEx status)`**

use `onStatusChanged` in `Cv2xListener`

**Global `telux::cv2x::ICv2xRadioManager::requestCv2xStatus (RequestCv2xStatusCallback cb)=0`**

use `requestCv2xStatus(RequestCv2xCalbackEx)`

**Global `telux::cv2x::ICv2xRadioManager::updateConfiguration (const std::string &configFilePath, UpdateConfigurationCallback cb)=0`**

Use `ICv2xConfig` instead

**Global `telux::cv2x::RequestCv2xStatusCallback`**

use `RequestCv2xStatusCallbackEx`

**Global `telux::cv2x::TrustedUEInfo::timeConfidenceLevel`**

Use `timeUncertainty` Time confidence level. Range from 0 to 127 with 0 being invalid/unavailable and 127 being the most confident.

**Global `telux::data::net::DmzEntriesCb`**

`telux::data::DmzEntryInfoCb` callback is used to get DMZ entries

**Global `telux::data::net::FirewallEntriesCb`**

`telux::data::FirewallEntryInfoCb` callback is used to get Firewall entries

**Global `telux::data::net::FirewallStatusCb`**

`telux::data::FirewallConfigCb` callback is used to get Firewall configuration

**Global `telux::data::net::IFirewallManager::addFirewallEntry`** (int profileId, std::shared\_ptr< I← FirewallEntry > entry, `telux::common::ResponseCallback` callback=nullptr)=0

Use `telux::data::addFirewallEntry` API to add firewall rule on any backhaul

**Global `telux::data::net::IFirewallManager::disableDmz`** (int profileId, const `telux::data::IpFamily`← Type ipType, `telux::common::ResponseCallback` callback=nullptr)=0

Use `telux::data::disableDmz` API to Disable DMZ on any backhaul

**Global `telux::data::net::IFirewallManager::enableDmz`** (int profileId, const std::string ipAddr, `telux::common::ResponseCallback` callback=nullptr)=0

Use `telux::data::enableDmz` API to enable DMZ on any backhaul

**Global `telux::data::net::IFirewallManager::removeFirewallEntry`** (int profileId, uint32\_t handle, `telux::common::ResponseCallback` callback=nullptr)=0

Use `telux::data::removeFirewallEntry` API to remove firewall rule from any backhaul

**Global `telux::data::net::IFirewallManager::requestDmzEntry`** (int profileId, DmzEntriesCb dmz← Cb)=0

Use `telux::data::requestDmzEntry` API to request DMZ on any backhaul

**Global `telux::data::net::IFirewallManager::requestFirewallEntries`** (int profileId, FirewallEntriesCb callback)=0

Use `telux::data::requestFirewallEntries` API to request firewall rules on any backhaul

**Global `telux::data::net::IFirewallManager::requestFirewallStatus`** (int profileId, FirewallStatusCb callback)=0

Use `telux::data::requestFirewallConfig` API to request firewall status on any backhaul

**Global `telux::data::net::IFirewallManager::setFirewall`** (int profileId, bool enable, bool allowPackets, `telux::common::ResponseCallback` callback=nullptr)=0

Use `telux::data::setFirewallConfig` API to set firewall on any backhaul

**Global `telux::data::net::IVlanManager::bindWithProfile`** (int profileId, int vlanId, `telux::common::`← `ResponseCallback` callback)=0

Use `bindToBackhaul()` API below to bind VLAN to backhaul

**Global `telux::data::net::IVlanManager::queryVlanMappingList`** (VlanMappingResponseCb call← back)=0

Use `queryVlanToBackhaulBindings()` API below to request VLAN to backhaul mapping

**Global `telux::data::net::IVlanManager::unbindFromProfile`** (int profileId, int vlanId, `telux`← `::common::ResponseCallback` callback)=0

Use `unbindFromBackhaul()` API below to unbind VLAN to backhaul

**Global `telux::loc::COMPASS`**

constellation type is not supported.

**Global `telux::loc::GNSS_LOC_SV_SYSTEM_COMPASS`**



constellation type is not supported.

**Global `telux::tel::GsmSignalStrengthInfo::getGsmBitErrorRate () const`**

This API not being supported

**Global `telux::tel::GsmSignalStrengthInfo::getTimingAdvance ()`**

This API not being supported

**Global `telux::tel::ICallListener::onECallMsdTransmissionStatus (int phoneId, telux::common::I←  
ErrorCode errorCode)`**

Use another `onECallMsdTransmissionStatus()` API with argument `ECallMsdTransmissionStatus`

**Global `telux::tel::IPhone::getRadioState ()=0`**

Use `IPhoneManager::requestOperatingMode()` API instead

**Global `telux::tel::IPhone::getServiceState ()=0`**

Use `requestVoiceServiceState()` API

**Global `telux::tel::IPhone::setRadioPower (bool enable, std::shared_ptr< telux::common::I←  
CommandResponseCallback > callback=nullptr)=0`**

Use `IPhoneManager::setOperatingMode()` API instead

**Global `telux::tel::IPhoneListener::onRadioStateChanged (int phoneId, RadioState radioState)`**

Use `onOperatingModeChanged()` API instead

**Global `telux::tel::IPhoneListener::onServiceStateChanged (int phoneId, ServiceState state)`**

Use `onVoiceServiceStateChanged()` listener

**Global `telux::tel::ISapCardManager::getState (SapState &sapState)=0`**

Use `requestSapState()` API below to get SAP state

**Global `telux::tel::ISmsManager::sendSms (const std::string &message, const std::string &receiver←  
Address, std::shared_ptr< telux::common::ICommandResponseCallback > callback=nullptr, std←  
::shared_ptr< telux::common::ICommandResponseCallback > deliveryCallback=nullptr)=0`**

Use API `ISmsManager::sendSms(const std::string &message, const std::string &receiverAddress, bool  
deliveryReportNeeded = true, SmsResponseCb callback = nullptr, std::string smscAddr = "")`

**Global `telux::tel::ISubscription::getMcc ()=0`**

Use `telux::tel::ISubscription::getMobileCountryCode()` API instead

**Global `telux::tel::ISubscription::getMnc ()=0`**

Use `telux::tel::ISubscription::getMobileNetworkCode()` API instead

**Global `telux::tel::LteSignalStrengthInfo::getLteChannelQualityIndicator () const`**

This API not being supported

As of version 1.54.0 this API is no longer supported.

**Global [telx::tel::LteSignalStrengthInfo::getTimingAdvance](#) () const**

This API not being supported

**Global [telx::tel::ServiceState](#)**

Use [requestVoiceServiceState\(\)](#) API or to know the status of phone

**Global [telx::tel::SmsMessage::getPdu](#) () const**

Use API [SmsMessage::getRawPdu](#)

**Global [telx::tel::WcdmaSignalStrengthInfo::getBitErrorRate](#) () const**

This API not being supported

# 5 Interfaces

---

## 5.1 Telematics SDK APIs

- Phone Factory
- Phone
- Call
- SMS
- SIM Card Services
- Location Services
- Data Services
- Subscription Management
- Network Selection
- Serving System
- Common
- C-V2X
- Audio
- Thermal Management
- TCU Activity Manager
- Remote SIM Provisioning
- Remote SIM
- Modem Config
- Wlan

## 5.2 Phone Factory

This section contains APIs related to [PhoneFactory](#).

### 5.2.1 Data Structure Documentation

#### 5.2.1.1 class telux::tel::PhoneFactory

[PhoneFactory](#) is the central factory to create all Telephony SDK Classes and services.

##### Public member functions

- `std::shared_ptr< IPhoneManager > getPhoneManager ()`
- `std::shared_ptr< ISmsManager > getSmsManager (int phoneId=DEFAULT_PHONE_ID, telux::common::InitResponseCb callback=nullptr)`
- `std::shared_ptr< ICallManager > getCallManager (telux::common::InitResponseCb callback=nullptr)`
- `std::shared_ptr< ICardManager > getCardManager ()`
- `std::shared_ptr< ISapCardManager > getSapCardManager (int slotId=DEFAULT_SLOT_ID)`
- `std::shared_ptr< ISubscriptionManager > getSubscriptionManager ()`
- `std::shared_ptr< IServingSystemManager > getServingSystemManager (int slotId=DEFAULT_SLOT_ID)`
- `std::shared_ptr< INetworkSelectionManager > getNetworkSelectionManager (int slotId=DEFAULT_SLOT_ID)`
- `std::shared_ptr< IRemoteSimManager > getRemoteSimManager (int slotId=DEFAULT_SLOT_ID)`
- `std::shared_ptr< IMultiSimManager > getMultiSimManager ()`
- `std::shared_ptr< ISimProfileManager > getSimProfileManager ()`
- `std::shared_ptr< ICellBroadcastManager > getCellBroadcastManager (SlotId slotId=DEFAULT_SLOT_ID)`
- `std::shared_ptr< IEcallManager > getEcallManager (telux::common::InitResponseCb callback=nullptr)`
- `std::shared_ptr< IimsSettingsManager > getImsSettingsManager (telux::common::InitResponseCb callback=nullptr)`
- `std::shared_ptr< IimsServingSystemManager > getImsServingSystemManager (SlotId slotId, telux::common::InitResponseCb callback=nullptr)`

##### Static Public Member Functions

- `static PhoneFactory & getInstance ()`

### 5.2.1.1.1 Member Function Documentation

#### 5.2.1.1.1.1 static PhoneFactory& telux::tel::PhoneFactory::getInstance ( ) [static]

Get Phone Factory instance.

#### 5.2.1.1.1.2 std::shared\_ptr<IPhoneManager> telux::tel::PhoneFactory::getPhoneManager ( )

Get Phone Manager instance. Phone Manager is the main entry point into the telephony subsystem.

#### Returns

Pointer of [IPhoneManager](#) object.

#### 5.2.1.1.1.3 std::shared\_ptr<ISmsManager> telux::tel::PhoneFactory::getSmsManager ( int *phoneId* = *DEFAULT\_PHONE\_ID*, telux::common::InitResponseCb *callback* = *nullptr* )

Get SMS Manager instance for Phone ID. SMSManager used to send and receive SMS messages.

#### Parameters

in	<i>phoneId</i>	Unique identifier for the phone
in	<i>callback</i>	Optional callback pointer to get response of SMS Manager initialization. It will be invoked when initialization is either succeeded or failed. In case of failure response, the provided SMS Manager object will no more be a valid object.

#### Returns

Pointer of [ISmsManager](#) object or nullptr in case of failure.

#### 5.2.1.1.1.4 std::shared\_ptr<ICallManager> telux::tel::PhoneFactory::getCallManager ( telux↔::common::InitResponseCb *callback* = *nullptr* )

Get Call Manager instance to determine state of active calls and perform other functions like dial, conference, swap call.

#### Parameters

in	<i>callback</i>	Optional callback pointer to get response of CallManager initialization. It will be invoked when initialization is either succeeded or failed. In case of failure response, the provided Call Manager object will no more be a valid object.
----	-----------------	--

#### Returns

Pointer of [ICallManager](#) object or nullptr in case of failure.

#### 5.2.1.1.1.5 `std::shared_ptr<ICardManager> telux::tel::PhoneFactory::getCardManager ( )`

Get Card Manager instance to handle services such as transmitting APDU, SIM IO and more.

#### Returns

Pointer of [ICardManager](#) object.

#### 5.2.1.1.1.6 `std::shared_ptr<ISapCardManager> telux::tel::PhoneFactory::getSapCardManager ( int slotId = DEFAULT_SLOT_ID )`

Get Sap Card Manager instance associated with the provided slot id. This object will handle services in SAP mode such as APDU, SIM Power On/Off and SIM reset.

#### Parameters

in	<i>slotId</i>	Unique identifier for the SIM slot
----	---------------	------------------------------------

#### Returns

Pointer of [ISapCardManager](#) object.

#### 5.2.1.1.1.7 `std::shared_ptr<ISubscriptionManager> telux::tel::PhoneFactory::getSubscriptionManager ( )`

Get Subscription Manager instance to get device subscription details

#### Returns

Pointer of [ISubscriptionManager](#) object.

#### 5.2.1.1.1.8 `std::shared_ptr<IServingSystemManager> telux::tel::PhoneFactory::getServingSystemManager ( int slotId = DEFAULT_SLOT_ID )`

Get Serving System Manager instance to get and set preferred network type.

#### Parameters

in	<i>slotId</i>	Unique identifier for the SIM slot
----	---------------	------------------------------------

#### Returns

Pointer of [IServingSystemManager](#) object.

#### 5.2.1.1.1.9 `std::shared_ptr<INetworkSelectionManager> telux::tel::PhoneFactory::getNetworkSelectionManager ( int slotId = DEFAULT_SLOT_ID )`

Get Network Selection Manager instance to get and set selection mode, get and set preferred networks and scan available networks.

in	<i>slotId</i>	Unique identifier for the SIM slot
----	---------------	------------------------------------

**Returns**

Pointer of [INetworkSelectionManager](#) object.

**5.2.1.1.1.10** `std::shared_ptr<IRemoteSimManager> telux::tel::PhoneFactory::getRemoteSimManager ( int slotId = DEFAULT_SLOT_ID )`

Get Remote SIM Manager instance to handle services like exchanging APDU, SIM Power On/Off, etc.

**Parameters**

in	<i>slotId</i>	Unique identifier for the SIM slot
----	---------------	------------------------------------

**Returns**

Pointer of [IRemoteSimManager](#) object.

**5.2.1.1.1.11** `std::shared_ptr<IMultiSimManager> telux::tel::PhoneFactory::getMultiSimManager ( )`

Get Multi SIM Manager instance to handle operations like high capabilty switch.

**Returns**

Pointer of [IMultiSimManager](#) object.

**5.2.1.1.1.12** `std::shared_ptr<ISimProfileManager> telux::tel::PhoneFactory::getSimProfileManager ( )`

Get SimProfileManager. SimProfileManager is a primary interface for remote eUICC(eSIM) provisioning and local profile assistance.

**Returns**

Pointer of [ISimProfileManager](#) object or nullptr in case of failure.

**5.2.1.1.1.13** `std::shared_ptr<ICellBroadcastManager> telux::tel::PhoneFactory::getCellBroadcastManager ( SlotId slotId = DEFAULT_SLOT_ID )`

**5.2.1.1.1.14** `std::shared_ptr<IEcallManager> telux::tel::PhoneFactory::getEcallManager ( telux::common::InitResponseCb callback = nullptr )`

**5.2.1.1.1.15** `std::shared_ptr<IImSettingsManager> telux::tel::PhoneFactory::getImSettingsManager ( telux::common::InitResponseCb callback = nullptr )`

Get ImSettings Manager instance to handle IMS service enable configuration parameters like enable/disable voIMS.

in	<i>callback</i>	Optional callback pointer to get the response of the manager initialisation.
----	-----------------	--

**Returns**

Pointer of [ImsSettingsManager](#) object.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

```
5.2.1.1.16 std::shared_ptr<ImsServingSystemManager> telux::tel::PhoneFactory::getImsServingSystemManager ( SlotId slotId, telux::common::InitResponseCb callback = nullptr )
```

Get IMS Serving System Manager instance to query IMS registration status

**Returns**

Pointer of [ImsServingSystemManager](#) object or nullptr in case of failure.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.



## 5.3 Phone

This section contains APIs related to Phone, Signal Strength and interfaces to register global listeners to event notifications.

### 5.3.1 Data Structure Documentation

#### 5.3.1.1 class `telux::tel::GsmCellIdentity`

`GsmCellIdentity` class provides methods to get mobile country code, mobile network code, location area code, cell identity, absolute RF channel number and base station identity code.

##### Public member functions

- `GsmCellIdentity` (int *mcc*, int *mnc*, int *lac*, int *cid*, int *arfcn*, int *bsic*)
- const int `getMcc` ()
- const int `getMnc` ()
- const int `getLac` ()
- const int `getIdentity` ()
- const int `getArfcn` ()
- const int `getBaseStationIdentityCode` ()

##### 5.3.1.1.1 Constructors and Destructors

5.3.1.1.1 `telux::tel::GsmCellIdentity::GsmCellIdentity` ( int *mcc*, int *mnc*, int *lac*, int *cid*, int *arfcn*, int *bsic* )

##### 5.3.1.1.2 Member Function Documentation

5.3.1.1.2.1 const int `telux::tel::GsmCellIdentity::getMcc` ( )

Get the Mobile Country Code.

##### Returns

Mcc value.

5.3.1.1.2.2 const int `telux::tel::GsmCellIdentity::getMnc` ( )

Get the Mobile Network Code.

##### Returns

Mnc value.

#### 5.3.1.1.2.3 `const int telux::tel::GsmCellIdentity::getLac ( )`

Get the location area code.

##### Returns

Location area code.

#### 5.3.1.1.2.4 `const int telux::tel::GsmCellIdentity::getIdentity ( )`

Get the cell identity.

##### Returns

Cell identity.

#### 5.3.1.1.2.5 `const int telux::tel::GsmCellIdentity::getArfcn ( )`

Get the absolute RF channel number.

##### Returns

Absolute RF channel number.

#### 5.3.1.1.2.6 `const int telux::tel::GsmCellIdentity::getBaseStationIdentityCode ( )`

Get the base station identity code.

##### Returns

Base station identity code.

### 5.3.1.2 `class telux::tel::CdmaCellIdentity`

`CdmaCellIdentity` class provides methods to get the network identifier, system identifier, base station identifier, longitude and latitude.

##### Public member functions

- `CdmaCellIdentity` (int networkId, int systemId, int baseStationId, int longitude, int latitude)
- `const int getNid ( )`
- `const int getSid ( )`
- `const int getBaseStationId ( )`
- `const int getLongitude ( )`
- `const int getLatitude ( )`

### 5.3.1.2.1 Constructors and Destructors

**5.3.1.2.1.1** `telux::tel::CdmaCellIdentity::CdmaCellIdentity ( int networkId, int systemId, int baseStationId, int longitude, int latitude )`

### 5.3.1.2.2 Member Function Documentation

**5.3.1.2.2.1** `const int telux::tel::CdmaCellIdentity::getNid ( )`

Get the network identifier.

#### Returns

Network identifier.

**5.3.1.2.2.2** `const int telux::tel::CdmaCellIdentity::getSid ( )`

Get the system identifier.

#### Returns

System identifier.

**5.3.1.2.2.3** `const int telux::tel::CdmaCellIdentity::getBaseStationId ( )`

Get the base station identifier.

#### Returns

Base station identifier.

**5.3.1.2.2.4** `const int telux::tel::CdmaCellIdentity::getLongitude ( )`

Get the longitude.

#### Returns

Longitude.

**5.3.1.2.2.5** `const int telux::tel::CdmaCellIdentity::getLatitude ( )`

Get the latitude.

#### Returns

Latitude.

### 5.3.1.3 class telux::tel::LteCellIdentity

[LteCellIdentity](#) class provides methods to get the mobile country code, mobile network code, cell identity, physical cell identifier, tracking area code and absolute Rf channel number.

#### Public member functions

- [LteCellIdentity](#) (int mcc, int mnc, int ci, int pci, int tac, int earfcn)
- const int [getMcc](#) ()
- const int [getMnc](#) ()
- const int [getIdentity](#) ()
- const int [getPhysicalCellId](#) ()
- const int [getTrackingAreaCode](#) ()
- const int [getEarfcn](#) ()

#### 5.3.1.3.1 Constructors and Destructors

**5.3.1.3.1.1** `telux::tel::LteCellIdentity::LteCellIdentity ( int mcc, int mnc, int ci, int pci, int tac, int earfcn )`

#### 5.3.1.3.2 Member Function Documentation

**5.3.1.3.2.1** `const int telux::tel::LteCellIdentity::getMcc ( )`

Get the Mobile Country Code.

#### Returns

Mcc value.

**5.3.1.3.2.2** `const int telux::tel::LteCellIdentity::getMnc ( )`

Get the Mobile Network Code.

#### Returns

Mnc value.

**5.3.1.3.2.3** `const int telux::tel::LteCellIdentity::getIdentity ( )`

Get the cell identity.

#### Returns

Cell identity.

#### 5.3.1.3.2.4 `const int telux::tel::LteCellIdentity::getPhysicalCellId ( )`

Get the physical cell identifier.

##### Returns

Physical cell identifier.

#### 5.3.1.3.2.5 `const int telux::tel::LteCellIdentity::getTrackingAreaCode ( )`

Get the tracking area code.

##### Returns

Tracking area code.

#### 5.3.1.3.2.6 `const int telux::tel::LteCellIdentity::getEarfcn ( )`

Get the absolute RF channel number.

##### Returns

Absolute RF channel number.

### 5.3.1.4 `class telux::tel::WcdmaCellIdentity`

[WcdmaCellIdentity](#) class provides methods to get the mobile country code, mobile network code, location area code, cell identifier, primary scrambling code and absolute RF channel number.

#### Public member functions

- [WcdmaCellIdentity](#) (int mcc, int mnc, int lac, int cid, int psc, int uarfcn)
- `const int getMcc ( )`
- `const int getMnc ( )`
- `const int getLac ( )`
- `const int getIdentity ( )`
- `const int getPrimaryScramblingCode ( )`
- `const int getUarfcn ( )`

#### 5.3.1.4.1 Constructors and Destructors

5.3.1.4.1.1 `telux::tel::WcdmaCellIdentity::WcdmaCellIdentity ( int mcc, int mnc, int lac, int cid, int psc, int uarfcn )`

### 5.3.1.4.2 Member Function Documentation

#### 5.3.1.4.2.1 `const int telux::tel::WcdmaCellIdentity::getMcc ( )`

Get the Mobile Country Code.

##### Returns

Mcc value.

#### 5.3.1.4.2.2 `const int telux::tel::WcdmaCellIdentity::getMnc ( )`

Get the Mobile Network Code.

##### Returns

Mnc value.

#### 5.3.1.4.2.3 `const int telux::tel::WcdmaCellIdentity::getLac ( )`

Get the location area code.

##### Returns

Location area code.

#### 5.3.1.4.2.4 `const int telux::tel::WcdmaCellIdentity::getIdentity ( )`

Get the cell identity.

##### Returns

Cell identity.

#### 5.3.1.4.2.5 `const int telux::tel::WcdmaCellIdentity::getPrimaryScramblingCode ( )`

Get the primary scrambling code.

##### Returns

Primary scrambling code.

#### 5.3.1.4.2.6 `const int telux::tel::WcdmaCellIdentity::getUarfcn ( )`

Get the absolute RF channel number.

##### Returns

Absolute RF channel number.

### 5.3.1.5 class telux::tel::TdscdmaCellIdentity

[TdscdmaCellIdentity](#) class provides methods to get the mobile country code, mobile network code, location area code, cell identity and cell parameters identifier.

#### Public member functions

- [TdscdmaCellIdentity](#) (int mcc, int mnc, int lac, int cid, int cpid)
- const int [getMcc](#) ()
- const int [getMnc](#) ()
- const int [getLac](#) ()
- const int [getIdentity](#) ()
- const int [getParametersId](#) ()

#### 5.3.1.5.1 Constructors and Destructors

**5.3.1.5.1.1** `telux::tel::TdscdmaCellIdentity::TdscdmaCellIdentity ( int mcc, int mnc, int lac, int cid, int cpid )`

#### 5.3.1.5.2 Member Function Documentation

**5.3.1.5.2.1** `const int telux::tel::TdscdmaCellIdentity::getMcc ( )`

Get the Mobile Country Code.

#### Returns

Mcc value.

**5.3.1.5.2.2** `const int telux::tel::TdscdmaCellIdentity::getMnc ( )`

Get the Mobile Network Code.

#### Returns

Mnc value.

**5.3.1.5.2.3** `const int telux::tel::TdscdmaCellIdentity::getLac ( )`

Get the location area code

#### Returns

Location area code.

#### 5.3.1.5.2.4 `const int telux::tel::TdscdmaCellIdentity::getIdentity ( )`

Get the cell identity.

##### Returns

Cell identity.

#### 5.3.1.5.2.5 `const int telux::tel::TdscdmaCellIdentity::getParametersId ( )`

Get the cell parameters identifier.

##### Returns

Cell parameters identifier.

### 5.3.1.6 `class telux::tel::CellInfo`

[CellInfo](#) class provides cell info type and checks whether the current cell is registered or not.

#### Public member functions

- virtual [CellType](#) `getType ( )`
- virtual bool `isRegistered ( )`

#### Protected Attributes

- [CellType](#) `type_`
- int `registered_`

#### 5.3.1.6.1 Member Function Documentation

##### 5.3.1.6.1.1 `virtual CellType telux::tel::CellInfo::getType ( ) [virtual]`

Get the cell type.

##### Returns

CellType.

##### 5.3.1.6.1.2 `virtual bool telux::tel::CellInfo::isRegistered ( ) [virtual]`

Checks whether the current cell is registered or not.

##### Returns

If true cell is registered or vice-versa.



### 5.3.1.6.2 Field Documentation

5.3.1.6.2.1 `CellType` `telux::tel::CellInfo::type_` [protected]

5.3.1.6.2.2 `int` `telux::tel::CellInfo::registered_` [protected]

### 5.3.1.7 class `telux::tel::GsmCellInfo`

`GsmCellInfo` class provides methods to get cell type, cell registration status, cell identity and signal strength information.

#### Public member functions

- `GsmCellInfo` (`int` `registered`, `GsmCellIdentity` `id`, `GsmSignalStrengthInfo` `ssInfo`)
- `GsmCellIdentity` `getCellIdentity` ()
- `GsmSignalStrengthInfo` `getSignalStrengthInfo` ()

#### Additional Inherited Members

### 5.3.1.7.1 Constructors and Destructors

5.3.1.7.1.1 `telux::tel::GsmCellInfo::GsmCellInfo` ( `int` *registered*, `GsmCellIdentity` *id*, `GsmSignalStrengthInfo` *ssInfo* )

`GsmCellInfo` constructor.

#### Parameters

<code>in</code>	<i>registered</i>	- Registration status of the cell.
<code>in</code>	<i>id</i>	- GSM cell identity.
<code>in</code>	<i>ssInfo</i>	- GSM cell signal strength.

### 5.3.1.7.2 Member Function Documentation

5.3.1.7.2.1 `GsmCellIdentity` `telux::tel::GsmCellInfo::getCellIdentity` ( )

Get GSM cell identity information.

#### Returns

`GsmCellIdentity`.

5.3.1.7.2.2 `GsmSignalStrengthInfo` `telux::tel::GsmCellInfo::getSignalStrengthInfo` ( )

Get GSM cell signal strength information.

#### Returns

`GsmSignalStrengthInfo`.

### 5.3.1.8 class telux::tel::CdmaCellInfo

[CdmaCellInfo](#) class provides methods to get cell type, cell registration status, cell identity and signal strength information.

#### Public member functions

- [CdmaCellInfo](#) (int registered, [CdmaCellIdentity](#) id, [CdmaSignalStrengthInfo](#) ssInfo)
- [CdmaCellIdentity](#) getCellIdentity ()
- [CdmaSignalStrengthInfo](#) getSignalStrengthInfo ()

#### Additional Inherited Members

#### 5.3.1.8.1 Constructors and Destructors

5.3.1.8.1.1 **telux::tel::CdmaCellInfo::CdmaCellInfo ( int *registered*, [CdmaCellIdentity](#) *id*, [CdmaSignalStrengthInfo](#) *ssInfo* )**

[CdmaCellInfo](#) constructor

#### Parameters

in	<i>registered</i>	- Registration status of the cell.
in	<i>id</i>	- CDMA cell identity.
in	<i>ssInfo</i>	- CDMA cell signal strength.

#### 5.3.1.8.2 Member Function Documentation

5.3.1.8.2.1 **[CdmaCellIdentity](#) telux::tel::CdmaCellInfo::getCellIdentity ( )**

Get CDMA cell identity information.

#### Returns

[CdmaCellIdentity](#).

5.3.1.8.2.2 **[CdmaSignalStrengthInfo](#) telux::tel::CdmaCellInfo::getSignalStrengthInfo ( )**

Get CDMA cell signal strength information.

#### Returns

[CdmaSignalStrengthInfo](#).

### 5.3.1.9 class telux::tel::LteCellInfo

[LteCellInfo](#) class provides methods to get cell type, cell registration status, cell identity and signal strength information.

#### Public member functions

- [LteCellInfo](#) (int registered, [LteCellIdentity](#) id, [LteSignalStrengthInfo](#) ssInfo)
- [LteCellIdentity](#) getCellIdentity ()
- [LteSignalStrengthInfo](#) getSignalStrengthInfo ()

#### Additional Inherited Members

#### 5.3.1.9.1 Constructors and Destructors

**5.3.1.9.1.1** [telux::tel::LteCellInfo::LteCellInfo](#) ( int *registered*, [LteCellIdentity](#) *id*, [LteSignalStrengthInfo](#) *ssInfo* )

[LteCellInfo](#) constructor.

#### Parameters

in	<i>registered</i>	- Registration status of the cell.
in	<i>id</i>	- LTE cell identity class.
in	<i>ssInfo</i>	- LTE cell signal strength.

#### 5.3.1.9.2 Member Function Documentation

**5.3.1.9.2.1** [LteCellIdentity](#) [telux::tel::LteCellInfo::getCellIdentity](#) ( )

Get LTE cell identity information.

#### Returns

[LteCellIdentity](#).

**5.3.1.9.2.2** [LteSignalStrengthInfo](#) [telux::tel::LteCellInfo::getSignalStrengthInfo](#) ( )

Get LTE cell signal strength information.

#### Returns

[LteSignalStrengthInfo](#).

### 5.3.1.10 class telux::tel::WcdmaCellInfo

[WcdmaCellInfo](#) class provides methods to get cell type, cell registration status, cell identity and signal strength information.

#### Public member functions

- [WcdmaCellInfo](#) (int registered, [WcdmaCellIdentity](#) id, [WcdmaSignalStrengthInfo](#) ssInfo)
- [WcdmaCellIdentity](#) getCellIdentity ()
- [WcdmaSignalStrengthInfo](#) getSignalStrengthInfo ()

#### Additional Inherited Members

#### 5.3.1.10.1 Constructors and Destructors

**5.3.1.10.1.1** [telux::tel::WcdmaCellInfo::WcdmaCellInfo](#) ( int *registered*, [WcdmaCellIdentity](#) *id*, [WcdmaSignalStrengthInfo](#) *ssInfo* )

[WcdmaCellInfo](#) constructor.

#### Parameters

in	<i>registered</i>	- Registration status of the cell.
in	<i>id</i>	- WCDMA cell identity.
in	<i>ssInfo</i>	- WCDMA cell signal strength.

#### 5.3.1.10.2 Member Function Documentation

**5.3.1.10.2.1** [WcdmaCellIdentity](#) [telux::tel::WcdmaCellInfo::getCellIdentity](#) ( )

Get WCDMA cell identity information.

#### Returns

[WcdmaCellIdentity](#).

**5.3.1.10.2.2** [WcdmaSignalStrengthInfo](#) [telux::tel::WcdmaCellInfo::getSignalStrengthInfo](#) ( )

Get WCDMA cell signal strength information.

#### Returns

[WcdmaSignalStrengthInfo](#).

### 5.3.1.11 class telux::tel::TdscdmaCellInfo

[TdscdmaCellInfo](#) class provides methods to get cell type, cell registration status, cell identity and signal strength information.

#### Public member functions

- [TdscdmaCellInfo](#) (int registered, [TdscdmaCellIdentity](#) id, [TdscdmaSignalStrengthInfo](#) ssInfo)
- [TdscdmaCellIdentity](#) [getCellIdentity](#) ()
- [TdscdmaSignalStrengthInfo](#) [getSignalStrengthInfo](#) ()

#### Additional Inherited Members

#### 5.3.1.11.1 Constructors and Destructors

**5.3.1.11.1.1** [telux::tel::TdscdmaCellInfo::TdscdmaCellInfo](#) ( int *registered*, [TdscdmaCellIdentity](#) *id*, [TdscdmaSignalStrengthInfo](#) *ssInfo* )

[TdscdmaCellInfo](#) constructor.

#### Parameters

in	<i>registered</i>	- Registration status of the cell
in	<i>id</i>	- TDSCDMA cell identity.
in	<i>ssInfo</i>	- TDSCDMA cell signal strength.

#### 5.3.1.11.2 Member Function Documentation

**5.3.1.11.2.1** [TdscdmaCellIdentity](#) [telux::tel::TdscdmaCellInfo::getCellIdentity](#) ( )

Get TDSCDMA cell identity information.

#### Returns

[TdscdmaCellIdentity](#).

**5.3.1.11.2.2** [TdscdmaSignalStrengthInfo](#) [telux::tel::TdscdmaCellInfo::getSignalStrengthInfo](#) ( )

Get TDSCDMA cell signal strength information.

#### Returns

[TdscdmaSignalStrengthInfo](#).

### 5.3.1.12 struct telux::tel::ECallMsdOptionals

Represents MsdOptionals class as per European eCall MSD standard. i.e. EN 15722.

#### Data fields

Type	Field	Description
<a href="#">ECallOptionalData Type</a>	optionalData↔ Type	Type of optional data
bool	optionalData↔ Present	Availability of Optional data: true - Present or false - Absent
bool	recentVehicle↔ LocationN1↔ Present	Availability of Recent Vehicle Location N1 data: true - Present or false - Absent
bool	recentVehicle↔ LocationN2↔ Present	Availability of Recent Vehicle Location N2 data: true - Present or false - Absent
bool	numberOf↔ Passengers↔ Present	Availability of number of seat belts fastened data: true - Present or false - Absent

### 5.3.1.13 struct telux::tel::ECallMsdControlBits

Represents [ECallMsdControlBits](#) structure as per European eCall MSD standard. i.e. EN 15722.

#### Data fields

Type	Field	Description
bool	automatic↔ Activation	auto / manual activation
bool	testCall	test / emergency call
bool	positionCan↔ BeTrusted	false if coincidence < 95% of reported pos within +/- 150m
<a href="#">ECallVehicle Type</a>	vehicleType: 5	Represents a vehicle class as per EN 15722

### 5.3.1.14 struct telux::tel::ECallVehicleIdentificationNumber

Represents VehicleIdentificationNumber structure as per European eCall MSD standard. i.e. EN 15722. Vehicle Identification Number confirming ISO3779.

#### Data fields

Type	Field	Description
string	isowmi	World Manufacturer Index (WMI)
string	isovds	Vehicle Type Descriptor (VDS)
string	isovis↔ Modelyear	Model year from Vehicle Identifier Section (VIS)
string	isovisSeqPlant	Plant code + sequential number from VIS

### 5.3.1.15 struct telux::tel::ECallVehiclePropulsionStorageType

Represents VehiclePropulsionStorageType structure as per European eCall MSD standard. i.e. EN 15722. Vehicle Propulsion type (energy storage): True- Present, False - Absent

#### Data fields

Type	Field	Description
bool	gasolineTank↔ Present	Represents the presence of Gasoline Tank in the vehicle.
bool	dieselTank↔ Present	Represents the presence of Diesel Tank in the vehicle
bool	compressed↔ NaturalGas	Represents the presence of CNG in the vehicle
bool	liquid↔ PropaneGas	Represents the presence of Liquid Propane Gas in the vehicle
bool	electric↔ EnergyStorage	Represents the presence of Electronic Storage in the vehicle
bool	hydrogen↔ Storage	Represents the presence of Hydrogen Storage in the vehicle
bool	otherStorage	Represents the presence of Other types of storage in the vehicle

### 5.3.1.16 struct telux::tel::ECallVehicleLocation

Represents VehicleLocation structure as per European eCall MSD standard. i.e. EN 15722.

#### Data fields

Type	Field	Description
int32_t	position↔ Latitude	latitude in value range (-2147483648 to 2147483647)
int32_t	position↔ Longitude	

### 5.3.1.17 struct telux::tel::ECallVehicleLocationDelta

Represents VehicleLocationDelta structure as per European eCall MSD standard. i.e. EN 15722. Delta with respect to Current Vehicle location.

#### Data fields

Type	Field	Description
int16_t	latitudeDelta	( 1 Unit = 100 milliarcseconds, range: -512 to 511)
int16_t	longitudeDelta	( 1 Unit = 100 milliarcseconds, range: -512 to 511)

### 5.3.1.18 struct telux::tel::ECallOptionalPdu

Optional information for the emergency rescue service.

#### Data fields

Type	Field	Description
ECallDefault↔ Options	eCallDefault↔ Options	Optional information

### 5.3.1.19 struct telux::tel::ECallMsdata

Data structure to hold all details required to construct an MSD

#### Data fields

Type	Field	Description
ECallMsdata↔ Optionals	optionals	Indicates presence of optionals in ECall MSD
uint8_t	message↔ Identifier	Starts with 1 for each new , increment on retransmission
ECallMsdata↔ ControlBits	control	ECallMsdataControlBits structure as per European standard i.e. EN 15722
ECallVehicle↔ Identification↔ Number	vehicle↔ Identification↔ Number	VIN (vehicle identification number) according to ISO3779
ECallVehicle↔ Propulsion↔ StorageType	vehicle↔ Propulsion↔ Storage	VehiclePropulsionStorageType structure as per European standard i.e. EN 15722
uint32_t	timestamp	Seconds elapsed since midnight 01.01.1970 UTC
ECallVehicle↔ Location	vehicleLocation	VehicleLocation structure as per European standard. i.e. EN 15722
uint8_t	vehicle↔ Direction	Direction of travel in 2 degrees steps from magnetic north
ECallVehicle↔ LocationDelta	recentVehicle↔ LocationN1	Change in latitude and longitude compared to the last MSD transmission
ECallVehicle↔ LocationDelta	recentVehicle↔ LocationN2	Change in latitude and longitude compared to the last but one MSD transmission
uint8_t	numberOf↔ Passengers	Number of occupants in the vehicle
ECall↔ OptionalPdu	optionalPdu	Optional information for the emergency rescue service (103 bytes, ASN.1 encoded); may also point to an address, where this information is locatedOptional information for the emergency rescue service

### 5.3.1.20 struct telux::tel::ECallModelInfo

Represents eCall operating mode information



**Data fields**

Type	Field	Description
<a href="#">ECallMode</a>	mode	Represents eCall operating mode
<a href="#">ECallModeReason</a>	reason	Represents eCall operating mode change reason

**5.3.1.21 struct telux::tel::ECallHlapTimerStatus**

Represents status of various eCall High Level Application Protocol(HLAP) timers that are maintained by UE state machine. This does not retrieve status of timers maintained by the PSAP. The timers are represented according to EN 16062:2015 standard.

**Data fields**

Type	Field	Description
<a href="#">HlapTimerStatus</a>	t2	T2 Timer status
<a href="#">HlapTimerStatus</a>	t5	T5 Timer status
<a href="#">HlapTimerStatus</a>	t6	T6 Timer status
<a href="#">HlapTimerStatus</a>	t7	T7 Timer status
<a href="#">HlapTimerStatus</a>	t9	T9 Timer status

**5.3.1.22 struct telux::tel::ECallHlapTimerEvents**

Represents events that changes the status of various eCall High Level Application Protocol(HLAP) timers that are maintained by UE state machine. This does not retrieve events of timers maintained by the PSAP. The timers are represented according to EN 16062:2015 standard.

**Data fields**

Type	Field	Description
<a href="#">HlapTimerEvent</a>	t2	T2 Timer event
<a href="#">HlapTimerEvent</a>	t5	T5 Timer event
<a href="#">HlapTimerEvent</a>	t6	T6 Timer event
<a href="#">HlapTimerEvent</a>	t7	T7 Timer event
<a href="#">HlapTimerEvent</a>	t9	T9 Timer event

### 5.3.1.23 struct telux::tel::EcallConfig

Represents various configuration parameters related to automotive emergency call

#### Data fields

Type	Field	Description
<a href="#">EcallConfig</a> ↔ <a href="#">Validity</a>	config↔ ValidityMask	Indicates the valid configuration parameters in the structure. A bit set to 1 denotes that the corresponding configuration parameter is valid
bool	muteRxAudio	
<a href="#">ECallNumType</a>	numType	
string	overriddenNum	
bool	useCannedMsd	
uint32_t	gnssUpdate↔ Interval	
uint32_t	t2Timer	
uint32_t	t7Timer	
uint32_t	t9Timer	
uint8_t	msdVersion	

### 5.3.1.24 class telux::tel::IPhone

This class allows getting system information and registering for system events. Each Phone instance is associated with a single SIM. So on a dual SIM device you would have 2 Phone instances.

#### Public member functions

- virtual [telux::common::Status](#) [getPhoneId](#) (int &phId)=0
- virtual [RadioState](#) [getRadioState](#) ()=0
- virtual [telux::common::Status](#) [requestVoiceRadioTechnology](#) ([VoiceRadioTechResponseCb](#) callback)=0
- virtual [ServiceState](#) [getServiceState](#) ()=0
- virtual [telux::common::Status](#) [requestVoiceServiceState](#) (std::weak\_ptr< [IVoiceServiceStateCallback](#) > callback)=0
- virtual [telux::common::Status](#) [setRadioPower](#) (bool enable, std::shared\_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status](#) [requestCellInfo](#) ([CellInfoCallback](#) callback)=0
- virtual [telux::common::Status](#) [setCellInfoListRate](#) (uint32\_t timeInterval, [common::ResponseCallback](#) callback)=0
- virtual [telux::common::Status](#) [requestSignalStrength](#) (std::shared\_ptr< [ISignalStrengthCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status](#) [setECallOperatingMode](#) ([ECallMode](#) eCallMode, [telux::common::ResponseCallback](#) callback)=0

- virtual `telux::common::Status requestECallOperatingMode (ECallGetOperatingModeCallback callback)=0`
- virtual `~IPhone ()`

### 5.3.1.24.1 Constructors and Destructors

5.3.1.24.1.1 virtual `telux::tel::IPhone::~~IPhone ( ) [virtual]`

### 5.3.1.24.2 Member Function Documentation

5.3.1.24.2.1 virtual `telux::common::Status telux::tel::IPhone::getPhoneId ( int & phId ) [pure virtual]`

Get the Phone ID corresponding to phone.

#### Parameters

out	<i>phoneId</i>	Unique identifier for the phone
-----	----------------	---------------------------------

#### Returns

Status of `getPhoneId` i.e. success or suitable error code.

5.3.1.24.2.2 virtual `RadioState telux::tel::IPhone::getRadioState ( ) [pure virtual]`

Get Radio state of device.

#### Returns

[RadioState](#)

**Deprecated** Use `IPhoneManager::requestOperatingMode()` API instead

5.3.1.24.2.3 virtual `telux::common::Status telux::tel::IPhone::requestVoiceRadioTechnology ( VoiceRadioTechResponseCb callback ) [pure virtual]`

Request for Radio technology type (3GPP/3GPP2) used for voice.

#### Parameters

in	<i>callback</i>	callback pointer to get the response of radio power request <a href="#">telux::tel::VoiceRadioTechResponseCb</a>
----	-----------------	---

#### Returns

Status of `requestVoiceRadioTechnology` i.e. success or suitable error code [telux::common::Status](#).

**5.3.1.24.2.4 virtual ServiceState telux::tel::IPhone::getServiceState ( ) [pure virtual]**

Get service state of the phone.

**Returns**

[ServiceState](#)

**Deprecated** Use [requestVoiceServiceState\(\)](#) API

**5.3.1.24.2.5 virtual telux::common::Status telux::tel::IPhone::requestVoiceServiceState ( std::weak\_ptr< IVoiceServiceStateCallback > callback ) [pure virtual]**

Request for voice service state to get the information of phone serving states

**Parameters**

in	<i>callback</i>	callback pointer to get the response of voice service state <a href="#">telux::tel::IVoiceServiceStateCallback</a> .
----	-----------------	---

**Returns**

Status of requestVoiceServiceState i.e. success or suitable error code [telux::common::Status](#).

**5.3.1.24.2.6 virtual telux::common::Status telux::tel::IPhone::setRadioPower ( bool enable, std::shared\_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]**

Set the radio power on or off.

**Parameters**

in	<i>enable</i>	Flag that determines whether to turn radio on or off
in	<i>callback</i>	Optional callback pointer to get the response of set radio power request

**Returns**

Status of setRadioPower i.e. success or suitable error code.

**Deprecated** Use [IPhoneManager::setOperatingMode\(\)](#) API instead

**5.3.1.24.2.7 virtual telux::common::Status telux::tel::IPhone::requestCellInfo ( CellInfoCallback callback ) [pure virtual]**

Get the cell information about current serving cell and neighboring cells.

in	<i>callback</i>	Callback to get the response of cell info request <a href="#">tel::CellInfoCallback</a>
----	-----------------	--

**Returns**

Status of requestCellInfo i.e. success or suitable error

#### 5.3.1.24.2.8 virtual telux::common::Status telux::tel::IPhone::setCellInfoListRate ( uint32\_t *timeInterval*, common::ResponseCallback *callback* ) [pure virtual]

Set the minimum time in milliseconds between when the cell info list should be received.

**Parameters**

in	<i>timeInterval</i>	Value of 0 means receive cell info list when any info changes. Value of INT_MAX means never receive cell info list even on change. Default value is 0
in	<i>callback</i>	Callback to get the response for set cell info list rate.

**Returns**

Status of setCellInfoListRate i.e. success or suitable error

#### 5.3.1.24.2.9 virtual telux::common::Status telux::tel::IPhone::requestSignalStrength ( std::shared\_ptr<ISignalStrengthCallback > *callback = nullptr* ) [pure virtual]

Get current signal strength of the associated network.

**Parameters**

in	<i>callback</i>	Optional callback pointer to get the response of signal strength request
----	-----------------	--

**Returns**

Status of requestSignalStrength i.e. success or suitable error code.

#### 5.3.1.24.2.10 virtual telux::common::Status telux::tel::IPhone::setECallOperatingMode ( ECallMode *eCallMode*, telux::common::ResponseCallback *callback* ) [pure virtual]

Sets the eCall operating mode

**Parameters**

in	<i>eCallMode</i>	- <a href="#">ECallMode</a>
in	<i>callback</i>	- Callback function to get the response for set eCall operating mode request.

**Returns**

Status of setECallOperatingMode i.e. success or suitable error

### 5.3.1.24.2.11 virtual telux::common::Status telux::tel::IPhone::requestECallOperatingMode ( ECallGetOperatingModeCallback *callback* ) [pure virtual]

Get the eCall operating mode

**Parameters**

<i>in</i>	<i>callback</i>	- Callback function to get the response of eCall operating mode request
-----------	-----------------	---

**Returns**

Status of requestECallOperatingMode i.e. success or suitable error

### 5.3.1.25 class telux::tel::ISignalStrengthCallback

Interface for Signal strength callback object. Client needs to implement this interface to get single shot responses for commands like get signal strength.

The methods in callback can be invoked from multiple different threads. The implementation should be thread safe.

**Public member functions**

- virtual void [signalStrengthResponse](#) (std::shared\_ptr< [SignalStrength](#) > signalStrength, [telux::common::ErrorCode](#) error)
- virtual [~ISignalStrengthCallback](#) ()

#### 5.3.1.25.1 Constructors and Destructors

##### 5.3.1.25.1.1 virtual telux::tel::ISignalStrengthCallback::~ISignalStrengthCallback ( ) [virtual]

#### 5.3.1.25.2 Member Function Documentation

##### 5.3.1.25.2.1 virtual void telux::tel::ISignalStrengthCallback::signalStrengthResponse ( std::shared\_ptr< [SignalStrength](#) > *signalStrength*, [telux::common::ErrorCode](#) *error* ) [virtual]

This function is called with the response to requestSignalStrength API.

**Parameters**

<i>in</i>	<i>signalStrength</i>	Pointer to signal strength object
<i>in</i>	<i>error</i>	Return code for whether the operation succeeded or failed <a href="#">SUCCESS</a> <a href="#">RADIO_NOT_AVAILABLE</a>

### 5.3.1.26 class telux::tel::IVoiceServiceStateCallback

Interface for voice service state callback object. Client needs to implement this interface to get single shot responses for commands like request voice radio technology.

The methods in callback can be invoked from multiple different threads. The implementation should be thread safe.

#### Public member functions

- virtual void [voiceServiceStateResponse](#) (const std::shared\_ptr< [VoiceServiceInfo](#) > &serviceInfo, [telux::common::ErrorCode](#) error)
- virtual [~IVoiceServiceStateCallback](#) ()

#### 5.3.1.26.1 Constructors and Destructors

5.3.1.26.1.1 virtual [telux::tel::IVoiceServiceStateCallback::~~IVoiceServiceStateCallback](#) ( )  
[virtual]

#### 5.3.1.26.2 Member Function Documentation

5.3.1.26.2.1 virtual void [telux::tel::IVoiceServiceStateCallback::voiceServiceStateResponse](#) ( const std::shared\_ptr< [VoiceServiceInfo](#) > & *serviceInfo*, [telux::common::ErrorCode](#) *error* )  
[virtual]

This function is called with the response to requestVoiceServiceState API.

#### Parameters

in	<i>serviceInfo</i>	Pointer to voice service info object <a href="#">telux::tel::VoiceServiceInfo</a>
in	<i>error</i>	Return code for whether the operation succeeded or failed <ul style="list-style-type: none"> <li>• <a href="#">telux::common::ErrorCode::SUCCESS</a></li> <li>• <a href="#">telux::common::ErrorCode::RADIO_NOT_AVAILABLE</a></li> <li>• <a href="#">telux::common::ErrorCode::GENERIC_FAILURE</a></li> </ul>

### 5.3.1.27 struct telux::tel::SimRatCapability

Structure contains slotID and RAT capabilities corresponding to slot.

#### Data fields

Type	Field	Description
int	slotId	
<a href="#">RAT↔</a> <a href="#">Capabilities↔</a> <a href="#">Mask</a>	capabilities	

### 5.3.1.28 struct telux::tel::CellularCapabilityInfo

Structure contains information about device capability.

#### Data fields

Type	Field	Description
VoiceService↔ Technologies↔ Mask	voiceService↔ Techs	Indicates voice support capabilities
int	simCount	The maximum number of SIMs that can be supported simultaneously
int	maxActiveSims	The maximum number of SIMs that can be simultaneously active. If this number is less than numberOfSims, it implies that any combination of the SIMs can be active and the remaining can be in standby.
vector< Sim↔ RatCapability >	simRat↔ Capabilities	A Sim inserted in a slot allows for certain rat capabilities. And the UE's HW allows for certain rat capabilities. This field lists the intersection of capabilities allowed by the Sim and the HW. The capabilities are indexed based on slotId.

### 5.3.1.29 class telux::tel::IPhoneListener

A listener class for monitoring changes in specific telephony states on the device, including service state and signal strength. Override the methods for the state that you wish to receive updates for.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

#### Public member functions

- virtual void [onServiceStateChanged](#) (int phoneId, [ServiceState](#) state)
- virtual void [onSignalStrengthChanged](#) (int phoneId, std::shared\_ptr< [SignalStrength](#) > signalStrength)
- virtual void [onCellInfoListChanged](#) (int phoneId, std::vector< std::shared\_ptr< [CellInfo](#) >> cellInfoList)
- virtual void [onRadioStateChanged](#) (int phoneId, [RadioState](#) radioState)
- virtual void [onVoiceRadioTechnologyChanged](#) (int phoneId, [RadioTechnology](#) radioTech)
- virtual void [onVoiceServiceStateChanged](#) (int phoneId, const std::shared\_ptr< [VoiceServiceInfo](#) > &serviceInfo)
- virtual void [onOperatingModeChanged](#) ([OperatingMode](#) mode)
- virtual void [onECallOperatingModeChange](#) (int phoneId, [telux::tel::ECallModeInfo](#) info)
- virtual [~IPhoneListener](#) ()



### 5.3.1.29.1 Constructors and Destructors

5.3.1.29.1.1 `virtual telux::tel::IPhoneListener::~~IPhoneListener ( ) [virtual]`

### 5.3.1.29.2 Member Function Documentation

5.3.1.29.2.1 `virtual void telux::tel::IPhoneListener::onServiceStateChanged ( int phoneId, ServiceState state ) [virtual]`

This function is called when device service state changes.

#### Parameters

in	<i>phoneId</i>	Unique id of the phone on which service state changed.
in	<i>state</i>	Service state of the phone <a href="#">ServiceState</a>

**Deprecated** Use [onVoiceServiceStateChanged\(\)](#) listener

5.3.1.29.2.2 `virtual void telux::tel::IPhoneListener::onSignalStrengthChanged ( int phoneId, std::shared_ptr< SignalStrength > signalStrength ) [virtual]`

This function is called when network signal strength changes.

#### Parameters

in	<i>phoneId</i>	Unique id of the phone on which signal strength state changed.
in	<i>signalStrength</i>	Pointer to signal strength object

5.3.1.29.2.3 `virtual void telux::tel::IPhoneListener::onCellInfoListChanged ( int phoneId, std::vector< std::shared_ptr< CellInfo >> cellInfoList ) [virtual]`

This function is called when info pertaining to current or neighboring cells change.

#### Parameters

in	<i>phoneId</i>	Unique id of the phone on which cell info changed.
in	<i>cellInfoList</i>	vector of shared pointers to cell info object

5.3.1.29.2.4 `virtual void telux::tel::IPhoneListener::onRadioStateChanged ( int phoneId, RadioState radioState ) [virtual]`

This function is called when radio state changes on phone

#### Parameters

in	<i>phone</i>	Unique id of the phone on which radio state changed
in	<i>radioState</i>	Radio state of the phone <a href="#">RadioState</a>

**Deprecated** Use `onOperatingModeChanged()` API instead

**5.3.1.29.2.5** `virtual void telux::tel::IPhoneListener::onVoiceRadioTechnologyChanged ( int phoneId, RadioTechnology radioTech ) [virtual]`

This function is called when the radio technology for voice service changes

#### Parameters

in	<i>phone</i>	Unique id of the phone on which radio technology changed
in	<i>radioTech</i>	Radio state of the phone <a href="#">telux::tel::RadioTechnology</a>

**5.3.1.29.2.6** `virtual void telux::tel::IPhoneListener::onVoiceServiceStateChanged ( int phoneId, const std::shared_ptr< VoiceServiceInfo > & serviceInfo ) [virtual]`

This function is called when the service state for voice service changes

#### Parameters

in	<i>phone</i>	Unique id of the phone on which radio technology changed
in	<i>serviceInfo</i>	pointer of voice service state info object <a href="#">telux::tel::VoiceServiceInfo</a>

**5.3.1.29.2.7** `virtual void telux::tel::IPhoneListener::onOperatingModeChanged ( OperatingMode mode ) [virtual]`

This function is called when the operating mode changes

#### Parameters

in	<i>mode</i>	Operating mode <a href="#">OperatingMode</a> .
----	-------------	--

**5.3.1.29.2.8** `virtual void telux::tel::IPhoneListener::onECallOperatingModeChange ( int phoneId, telux::tel::ECallModeInfo info ) [virtual]`

This function is called when eCall operating mode changes.

#### Parameters

in	<i>phoneId</i>	- Unique Id of phone for which eCall operating mode changed
in	<i>info</i>	- Indicates eCall operating mode change reason <a href="#">ECallModeInfo</a>

### 5.3.1.30 class telux::tel::IPhoneManager

Phone Manager creates one or more phones based on SIM slot count, it allows clients to register for notification of system events. Clients should check if the subsystem is ready before invoking any of the APIs.

#### Public member functions

- virtual bool `isSubsystemReady ()=0`
- virtual `std::future< bool > onSubsystemReady ()=0`
- virtual `telux::common::Status getPhoneIds (std::vector< int > &phoneIds)=0`
- virtual `int getPhoneIdFromSlotId (int slotId)=0`
- virtual `int getSlotIdFromPhoneId (int phoneId)=0`
- virtual `std::shared_ptr< IPhone > getPhone (int phoneId=DEFAULT_PHONE_ID)=0`
- virtual `telux::common::Status requestCellularCapabilityInfo (std::shared_ptr< ICellularCapabilityCallback > callback=nullptr)=0`
- virtual `telux::common::Status requestOperatingMode (std::shared_ptr< IOperatingModeCallback > callback=nullptr)=0`
- virtual `telux::common::Status setOperatingMode (OperatingMode operatingMode, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status registerListener (std::weak_ptr< IPhoneListener > listener)=0`
- virtual `telux::common::Status removeListener (std::weak_ptr< IPhoneListener > listener)=0`
- virtual `~IPhoneManager ()`

#### 5.3.1.30.1 Constructors and Destructors

5.3.1.30.1.1 virtual `telux::tel::IPhoneManager::~~IPhoneManager ( ) [virtual]`

#### 5.3.1.30.2 Member Function Documentation

5.3.1.30.2.1 virtual bool `telux::tel::IPhoneManager::isSubsystemReady ( ) [pure virtual]`

Checks the status of telephony subsystems and returns the result.

#### Returns

If true PhoneManager is ready for service (i.e Phone, Sms and Card).

**5.3.1.30.2.2** `virtual std::future<bool> telux::tel::IPhoneManager::onSubsystemReady ( ) [pure virtual]`

Wait for telephony subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when telephony subsystem is ready.

**5.3.1.30.2.3** `virtual telux::common::Status telux::tel::IPhoneManager::getPhoneIds ( std::vector< int > & phoneIds ) [pure virtual]`

Retrieves a list of Phone Ids. Each id is unique per phone. For example: on a dual SIM device, there would be 2 Phones.

#### Parameters

out	<i>phoneIds</i>	List of phone ids
-----	-----------------	-------------------

#### Returns

Status of getPhoneIds i.e. success or suitable error code.

**5.3.1.30.2.4** `virtual int telux::tel::IPhoneManager::getPhoneIdFromSlotId ( int slotId ) [pure virtual]`

Get the Phone Id for a given Slot Id.

#### Parameters

in	<i>slotId</i>	SIM Card Slot Id
----	---------------	------------------

#### Returns

Phone Id corresponding to the Slot Id.

**5.3.1.30.2.5** `virtual int telux::tel::IPhoneManager::getSlotIdFromPhoneId ( int phoneId ) [pure virtual]`

Get the SIM Slot Id for a given Phone Id.

#### Parameters

in	<i>phoneId</i>	Phone Id of the phone
----	----------------	-----------------------

#### Returns

Slot Id corresponding to the Phone Id.

**5.3.1.30.2.6** `virtual std::shared_ptr<IPhone> telux::tel::IPhoneManager::getPhone ( int phoneId = DEFAULT_PHONE_ID ) [pure virtual]`

Get the phone instance for a given phone identifier.

#### Parameters

in	<i>phoneId</i>	Identifier for phone instance, retrieved from getPhoneIds API
----	----------------	---

#### Returns

Pointer to Phone object corresponding to phoneId.

**5.3.1.30.2.7** `virtual telux::common::Status telux::tel::IPhoneManager::requestCellularCapabilityInfo ( std::shared_ptr< ICellularCapabilityCallback > callback = nullptr ) [pure virtual]`

Get the information about cellular capability.

#### Parameters

in	<i>callback</i>	Optional callback pointer to get the response of cellular capability.
----	-----------------	---

#### Returns

Status of requestCellularCapabilityInfo i.e. success or suitable error code.

**5.3.1.30.2.8** `virtual telux::common::Status telux::tel::IPhoneManager::requestOperatingMode ( std::shared_ptr< IOperatingModeCallback > callback = nullptr ) [pure virtual]`

Get current operating mode of the device.

#### Parameters

in	<i>callback</i>	Optional callback pointer to get the response of operating mode request
----	-----------------	---

#### Returns

Status of requestOperatingMode i.e. success or suitable error code.

**5.3.1.30.2.9** `virtual telux::common::Status telux::tel::IPhoneManager::setOperatingMode ( Operating↔ Mode operatingMode, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Set the operating mode of the device. Only valid transitions allowed from one mode to another.

in	<i>operatingMode</i>	Operating Mode to be set
in	<i>callback</i>	Optional callback pointer to get the response of set operatingmode request. In callback following error is returned. <ul style="list-style-type: none"> <li>• <a href="#">telux::common::ErrorCode::INVALID_TRANSITION</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_ARGUMENTS</a></li> <li>• <a href="#">telux::common::ErrorCode::DEVICE_IN_USE</a></li> <li>• <a href="#">telux::common::ErrorCode::NO_MEMORY</a></li> </ul>

**Returns**

Status of setOperatingMode i.e. success or suitable error code.

**5.3.130.2.10** `virtual telux::common::Status telux::tel::IPhoneManager::registerListener ( std::weak_ptr< IPhoneListener > listener ) [pure virtual]`

Register a listener for specific events in the telephony subsystem.

**Parameters**

in	<i>listener</i>	Pointer to Phone Listener object that processes the notification
----	-----------------	--

**Returns**

Status of registerListener i.e. success or suitable error code.

**5.3.130.2.11** `virtual telux::common::Status telux::tel::IPhoneManager::removeListener ( std::weak_ptr< IPhoneListener > listener ) [pure virtual]`

Remove a previously added listener.

**Parameters**

in	<i>listener</i>	Pointer to Phone Listener object that needs to be removed
----	-----------------	---

**Returns**

Status of removeListener i.e. success or suitable error code.

**5.3.1.31 class telux::tel::ICellularCapabilityCallback**

Interface for callback corresponding to cellular capability request. Client needs to implement this interface to get single shot responses for commands like get cellular capability.

The methods in callback can be invoked from multiple different threads. The implementation should be thread safe.

**Public member functions**

- virtual void `cellularCapabilityResponse` (`CellularCapabilityInfo` capabilityInfo, `telux::common::ErrorCode` error)
- virtual `~ICellularCapabilityCallback` ()

**5.3.1.31.1 Constructors and Destructors**

**5.3.1.31.1.1** virtual `telux::tel::ICellularCapabilityCallback::~~ICellularCapabilityCallback` ( ) [`virtual`]

**5.3.1.31.2 Member Function Documentation**

**5.3.1.31.2.1** virtual void `telux::tel::ICellularCapabilityCallback::cellularCapabilityResponse` ( `CellularCapabilityInfo` *capabilityInfo*, `telux::common::ErrorCode` *error* ) [`virtual`]

This function is called with the response to requestCellularCapabilityInfo API.

**Parameters**

in	<i>capabilityInfo</i>	Cellular capability information.
in	<i>error</i>	Return code for whether the operation succeeded or failed <ul style="list-style-type: none"> <li>• <code>telux::common::ErrorCode::SUCCESS</code></li> <li>• <code>telux::common::ErrorCode::INTERNAL</code></li> <li>• <code>telux::common::ErrorCode::NO_MEMORY</code></li> </ul>

**5.3.1.32 class telux::tel::IOperatingModeCallback**

Interface for operating mode callback object. Client needs to implement this interface to get single shot responses for commands like request current operating mode.

The methods in callback can be invoked from multiple different threads. The implementation should be thread safe.

**Public member functions**

- virtual void `operatingModeResponse` (`OperatingMode` operatingMode, `telux::common::ErrorCode` error)
- virtual `~IOperatingModeCallback` ()

**5.3.1.32.1 Constructors and Destructors**

**5.3.1.32.1.1** virtual `telux::tel::IOperatingModeCallback::~~IOperatingModeCallback` ( ) [`virtual`]

### 5.3.1.32.2 Member Function Documentation

#### 5.3.1.32.2.1 virtual void telux::tel::IOperatingModeCallback::operatingModeResponse ( OperatingMode *operatingMode*, telux::common::ErrorCode *error* ) [virtual]

This function is called with the response to requestOperatingMode API.

#### Parameters

in	<i>operatingMode</i>	OperatingMode
in	<i>error</i>	Return code for whether the operation succeeded or failed <ul style="list-style-type: none"> <li>telux::common::ErrorCode::SUCCESS</li> <li>telux::common::ErrorCode::INTERNAL_ERR</li> <li>telux::common::ErrorCode::NO_MEMORY</li> </ul>

### 5.3.1.33 class telux::tel::SignalStrength

[SignalStrength](#) class provides access to LTE, GSM, CDMA, WCDMA, TDSCDMA signal strengths.

#### Public member functions

- [SignalStrength](#) (std::shared\_ptr< [LteSignalStrengthInfo](#) > lteSignalStrengthInfo, std::shared\_ptr< [GsmSignalStrengthInfo](#) > gsmSignalStrengthInfo, std::shared\_ptr< [CdmaSignalStrengthInfo](#) > cdmaSignalStrengthInfo, std::shared\_ptr< [WcdmaSignalStrengthInfo](#) > wcdmaSignalStrengthInfo, std::shared\_ptr< [TdsdmaSignalStrengthInfo](#) > tdsdmaSignalStrengthInfo, std::shared\_ptr< [Nr5gSignalStrengthInfo](#) > nr5gSignalStrengthInfo)
- std::shared\_ptr< [LteSignalStrengthInfo](#) > [getLteSignalStrength](#) ()
- std::shared\_ptr< [GsmSignalStrengthInfo](#) > [getGsmSignalStrength](#) ()
- std::shared\_ptr< [CdmaSignalStrengthInfo](#) > [getCdmaSignalStrength](#) ()
- std::shared\_ptr< [WcdmaSignalStrengthInfo](#) > [getWcdmaSignalStrength](#) ()
- std::shared\_ptr< [TdsdmaSignalStrengthInfo](#) > [getTdsdmaSignalStrength](#) ()
- std::shared\_ptr< [Nr5gSignalStrengthInfo](#) > [getNr5gSignalStrength](#) ()

#### 5.3.1.33.1 Constructors and Destructors

##### 5.3.1.33.1.1 telux::tel::SignalStrength::SignalStrength ( std::shared\_ptr< [LteSignalStrengthInfo](#) > *lteSignalStrengthInfo*, std::shared\_ptr< [GsmSignalStrengthInfo](#) > *gsmSignalStrengthInfo*, std::shared\_ptr< [CdmaSignalStrengthInfo](#) > *cdmaSignalStrengthInfo*, std::shared\_ptr< [WcdmaSignalStrengthInfo](#) > *wcdmaSignalStrengthInfo*, std::shared\_ptr< [TdsdmaSignalStrengthInfo](#) > *tdsdmaSignalStrengthInfo*, std::shared\_ptr< [Nr5gSignalStrengthInfo](#) > *nr5gSignalStrengthInfo* )



### 5.3.1.33.2 Member Function Documentation

**5.3.1.33.2.1** `std::shared_ptr<LteSignalStrengthInfo> telux::tel::SignalStrength::getLteSignalStrength ( )`

Gives LTE signal strength instance.

#### Returns

Pointer to LTE signal strength instance that can be used to get lte dbm, signal level values.

**5.3.1.33.2.2** `std::shared_ptr<GsmSignalStrengthInfo> telux::tel::SignalStrength::getGsmSignalStrength ( )`

Gives GSM signal strength instance.

#### Returns

Pointer to GSM signal strength instance that can be used to get GSM dbm, signal level values.

**5.3.1.33.2.3** `std::shared_ptr<CdmaSignalStrengthInfo> telux::tel::SignalStrength::getCdmaSignalStrength ( )`

Gives CDMA signal strength instance.

#### Returns

Pointer to CDMA signal strength instance that can be used to get cdma/evdo dbm, signal level values.

**5.3.1.33.2.4** `std::shared_ptr<WcdmaSignalStrengthInfo> telux::tel::SignalStrength::getWcdmaSignalStrength ( )`

Gives WCDMA signal strength instance.

#### Returns

Pointer to WCDMA signal strength instance that can be used to get WCDMA dbm, signal level values.

**5.3.1.33.2.5** `std::shared_ptr<TdscdmaSignalStrengthInfo> telux::tel::SignalStrength::getTdscdmaSignalStrength ( )`

Gives TDSWCDMA signal strength instance.

#### Returns

Pointer to TDSWCDMA signal strength instance that can be used to get TDSCDMA RSCP value.

### 5.3.1.33.2.6 `std::shared_ptr<Nr5gSignalStrengthInfo> telux::tel::SignalStrength::getNr5gSignalStrength ( )`

Gives 5G NR signal strength instance.

#### Returns

Pointer to 5G NR signal strength instance that can be used to get 5G NR dbm and snr values.

### 5.3.1.34 `class telux::tel::LteSignalStrengthInfo`

LTE signal strength class provides methods to get details of lte signals like dbm, signal level, reference signal-to-noise ratio, channel quality indicator and signal strength.

#### Public member functions

- [LteSignalStrengthInfo](#) (int lteSignalStrength, int lteRsrp, int lteRsrq, int lteRssnr, int lteCqi, int timingAdvance)
- const [SignalStrengthLevel](#) getLevel ( ) const
- const int getDbm ( ) const
- const int getLteSignalStrength ( ) const
- const int getLteReferenceSignalReceiveQuality ( ) const
- const int getLteReferenceSignalSnr ( ) const
- const int getLteChannelQualityIndicator ( ) const
- const int getTimingAdvance ( ) const

#### 5.3.1.34.1 Constructors and Destructors

5.3.1.34.1.1 `telux::tel::LteSignalStrengthInfo::LteSignalStrengthInfo ( int lteSignalStrength, int lteRsrp, int lteRsrq, int lteRssnr, int lteCqi, int timingAdvance )`

#### 5.3.1.34.2 Member Function Documentation

5.3.1.34.2.1 `const SignalStrengthLevel telux::tel::LteSignalStrengthInfo::getLevel ( ) const`

Get signal level in the range.

#### Returns

Signal levels indicates the quality of signal being received by the device.

5.3.1.34.2.2 `const int telux::tel::LteSignalStrengthInfo::getDbm ( ) const`

Get the signal strength in dBm. (Valid value range [-140, -44] and INVALID\_SIGNAL\_STRENGTH\_VALUE i.e. unavailable).

**Returns**

LTE dBm value.

**5.3.1.34.2.3 const int telux::tel::LteSignalStrengthInfo::getLteSignalStrength ( ) const**

Get the LTE signal strength. (Valid value range [0, 31] and INVALID\_SIGNAL\_STRENGTH\_VALUE i.e. unavailable).

**Returns**

LTE signal strength.

**5.3.1.34.2.4 const int telux::tel::LteSignalStrengthInfo::getLteReferenceSignalReceiveQuality ( ) const**

Get LTE reference signal receive quality in dB. (Valid value range [-20, -3] and INVALID\_SIGNAL\_STRENGTH\_VALUE i.e. unavailable).

**Returns**

LteRsrq.

**5.3.1.34.2.5 const int telux::tel::LteSignalStrengthInfo::getLteReferenceSignalSnr ( ) const**

Get LTE reference signal signal-to-noise ratio, multiply by 0.1 to get SNR in dB. (Valid value range [-200, +300] and INVALID\_SIGNAL\_STRENGTH\_VALUE i.e. unavailable). (-200 = -20.0 dB, +300 = 30dB).

**Returns**

LteSnr.

**5.3.1.34.2.6 const int telux::tel::LteSignalStrengthInfo::getLteChannelQualityIndicator ( ) const**

Get LTE channel quality indicator. (Valid value range [0, 15] and INVALID\_SIGNAL\_STRENGTH\_VALUE i.e. unavailable).

**Deprecated** This API not being supported

**Returns**

LteCqI.

**Deprecated** As of version 1.54.0 this API is no longer supported.

**5.3.1.34.2.7 const int telux::tel::LteSignalStrengthInfo::getTimingAdvance ( ) const**

Get the timing advance in micro seconds. (Valid value range [0, 0x7FFFFFFE] and INVALID\_SIGNAL\_STRENGTH\_VALUE i.e. unavailable).

**Deprecated** This API not being supported

### Returns

Timing advance value.

## 5.3.1.35 class telux::tel::GsmSignalStrengthInfo

GSM signal strength provides methods to get GSM signal strength in dBm and GSM signal level.

### Public member functions

- [GsmSignalStrengthInfo](#) (int gsmSignalStrength, int gsmBitErrorRate, int timingAdvance)
- const [SignalStrengthLevel](#) [getLevel](#) () const
- const int [getDbm](#) () const
- const int [getGsmSignalStrength](#) () const
- const int [getGsmBitErrorRate](#) () const
- const int [getTimingAdvance](#) ()

### 5.3.1.35.1 Constructors and Destructors

**5.3.1.35.1.1** `telux::tel::GsmSignalStrengthInfo::GsmSignalStrengthInfo ( int gsmSignalStrength, int gsmBitErrorRate, int timingAdvance )`

### 5.3.1.35.2 Member Function Documentation

**5.3.1.35.2.1** `const SignalStrengthLevel telux::tel::GsmSignalStrengthInfo::getLevel ( ) const`

Get signal level in the range.

### Returns

Signal levels indicates the quality of signal being received by the device.

**5.3.1.35.2.2** `const int telux::tel::GsmSignalStrengthInfo::getDbm ( ) const`

Get the signal strength in dBm. (Valid value range [-113, -51] and INVALID\_SIGNAL\_STRENGTH\_VALUE i.e. unavailable).

### Returns

GSM signal strength in dBm.

**5.3.1.35.2.3 const int telux::tel::GsmSignalStrengthInfo::getGsmSignalStrength ( ) const**

Get the GSM signal strength. (Valid value range [0, 31] and INVALID\_SIGNAL\_STRENGTH\_VALUE i.e. unavailable).

**Returns**

GSM signal strength.

**5.3.1.35.2.4 const int telux::tel::GsmSignalStrengthInfo::getGsmBitErrorRate ( ) const**

Get the GSM bit error rate. (Valid value range [0, 7] and INVALID\_SIGNAL\_STRENGTH\_VALUE i.e. unavailable).

**Deprecated** This API not being supported

**Returns**

GSM bit error rate.

**5.3.1.35.2.5 const int telux::tel::GsmSignalStrengthInfo::getTimingAdvance ( )**

Get the timing advance in bit periods . 1 bit period = 48/13 us (Valid value range [0, 219] and INVALID\_SIGNAL\_STRENGTH\_VALUE i.e. unavailable).

**Deprecated** This API not being supported

**Returns**

timing advance.

**5.3.1.36 class telux::tel::CdmaSignalStrengthInfo**

CDMA signal strength provides methods to get details of CDMA and EVDO like signal strength in dBm and signal level.

**Public member functions**

- [CdmaSignalStrengthInfo](#) (int cdmaDbm, int cdmaEcio, int evdoDbm, int evdoEcio, int evdoSignalNoiseRatio)
- const [SignalStrengthLevel](#) [getLevel](#) () const
- const int [getDbm](#) () const
- const int [getCdmaEcio](#) () const
- const int [getEvdoEcio](#) () const
- const int [getEvdoSignalNoiseRatio](#) () const

### 5.3.1.36.1 Constructors and Destructors

**5.3.1.36.1.1** `telux::tel::CdmaSignalStrengthInfo::CdmaSignalStrengthInfo ( int cdmaDbm, int cdmaEcio, int evdoDbm, int evdoEcio, int evdoSignalNoiseRatio )`

### 5.3.1.36.2 Member Function Documentation

**5.3.1.36.2.1** `const SignalStrengthLevel telux::tel::CdmaSignalStrengthInfo::getLevel ( ) const`

Get signal level in the range.

#### Returns

Signal levels indicates the quality of signal being received by the device.

**5.3.1.36.2.2** `const int telux::tel::CdmaSignalStrengthInfo::getDbm ( ) const`

Get the signal strength in dBm.

#### Returns

Minimum value of Evdo dBm and Cdma dBm.

**5.3.1.36.2.3** `const int telux::tel::CdmaSignalStrengthInfo::getCdmaEcio ( ) const`

Get the CDMA Ec/Io in dB.

#### Returns

CDMA Ec/Io.

**5.3.1.36.2.4** `const int telux::tel::CdmaSignalStrengthInfo::getEvdoEcio ( ) const`

Get the EVDO Ec/Io in dB.

#### Returns

EVDO Ec/Io.

**5.3.1.36.2.5** `const int telux::tel::CdmaSignalStrengthInfo::getEvdoSignalNoiseRatio ( ) const`

Get the EVDO signal noise ratio. (Valid value range [0, 8] and 8 is the highest signal to noise ratio.)

#### Returns

EVDO SNR.

### 5.3.1.37 class telux::tel::WcdmaSignalStrengthInfo

WCDMA signal strength provides methods to get WCDMA signal strength in dBm and WCDMA signal level.

#### Public member functions

- [WcdmaSignalStrengthInfo](#) (int signalStrength, int bitErrorRate)
- const [SignalStrengthLevel](#) getLevel () const
- const int getDbm () const
- const int getSignalStrength () const
- const int getBitErrorRate () const

#### 5.3.1.37.1 Constructors and Destructors

**5.3.1.37.1.1** `telux::tel::WcdmaSignalStrengthInfo::WcdmaSignalStrengthInfo ( int signalStrength, int bitErrorRate )`

#### 5.3.1.37.2 Member Function Documentation

**5.3.1.37.2.1** `const SignalStrengthLevel telux::tel::WcdmaSignalStrengthInfo::getLevel ( ) const`

Get signal level in the range.

#### Returns

Signal levels indicates the quality of signal being received by the device.

**5.3.1.37.2.2** `const int telux::tel::WcdmaSignalStrengthInfo::getDbm ( ) const`

Get the signal strength in dBm. (Valid value range [-113, -51] and INVALID\_SIGNAL\_STRENGTH\_VALUE i.e. unavailable).

#### Returns

WCDMA signal strength in dBm.

**5.3.1.37.2.3** `const int telux::tel::WcdmaSignalStrengthInfo::getSignalStrength ( ) const`

Get the WCDMA signal strength. (Valid value range [0, 31] and INVALID\_SIGNAL\_STRENGTH\_VALUE i.e. unavailable).

#### Returns

WCDMA signal strength.

#### 5.3.1.37.2.4 `const int telux::tel::WcdmaSignalStrengthInfo::getBitErrorRate ( ) const`

Get the WCDMA bit error rate. (Valid value range [0, 7] and INVALID\_SIGNAL\_STRENGTH\_VALUE i.e. unavailable).

**Deprecated** This API not being supported

#### Returns

WCDMA bit error rate.

### 5.3.1.38 `class telux::tel::TdscdmaSignalStrengthInfo`

Tdscdma signal strength provides methods to get received signal code power.

#### Public member functions

- [TdscdmaSignalStrengthInfo](#) (int rscp)
- `const int getRscp () const`

#### 5.3.1.38.1 Constructors and Destructors

##### 5.3.1.38.1.1 `telux::tel::TdscdmaSignalStrengthInfo::TdscdmaSignalStrengthInfo ( int rscp )`

#### 5.3.1.38.2 Member Function Documentation

##### 5.3.1.38.2.1 `const int telux::tel::TdscdmaSignalStrengthInfo::getRscp ( ) const`

Get TdScdma received signal code power in dBm. (Valid Range [-120,-25], and INVALID\_SIGNAL\_STRENGTH\_VALUE i.e. unavailable).

#### Returns

TdScdma signal code power.

### 5.3.1.39 `class telux::tel::Nr5gSignalStrengthInfo`

5G NR signal strength provides methods to get signal strength and signal-to-noise ratio.

#### Public member functions

- [Nr5gSignalStrengthInfo](#) (int rsrp, int rsrq, int rssnr)
- `const SignalStrengthLevel getLevel () const`
- `const int getDbm () const`
- `const int getReferenceSignalReceiveQuality () const`
- `const int getReferenceSignalSnr () const`



### 5.3.1.39.1 Constructors and Destructors

**5.3.1.39.1.1** `telux::tel::Nr5gSignalStrengthInfo::Nr5gSignalStrengthInfo ( int rsrp, int rsrq, int rssnr )`

### 5.3.1.39.2 Member Function Documentation

**5.3.1.39.2.1** `const SignalStrengthLevel telux::tel::Nr5gSignalStrengthInfo::getLevel ( ) const`

Get signal level in the range.

#### Returns

Signal levels indicates the quality of signal being received by the device.

**5.3.1.39.2.2** `const int telux::tel::Nr5gSignalStrengthInfo::getDbm ( ) const`

Get the signal strength in dBm. (Valid value range [-140, -44] and INVALID\_SIGNAL\_STRENGTH\_VALUE i.e. unavailable). INVALID\_SIGNAL\_STRENGTH\_VALUE indicates that modem is not in ENDC connected mode.

#### Returns

5G NR dBm value.

**5.3.1.39.2.3** `const int telux::tel::Nr5gSignalStrengthInfo::getReferenceSignalReceiveQuality ( ) const`

Get 5G NR reference signal receive quality in dB. (Valid value range [-20, -3] and INVALID\_SIGNAL\_STRENGTH\_VALUE i.e. unavailable). INVALID\_SIGNAL\_STRENGTH\_VALUE indicates that modem is not in ENDC connected mode.

#### Returns

5G NR rsrq.

**5.3.1.39.2.4** `const int telux::tel::Nr5gSignalStrengthInfo::getReferenceSignalSnr ( ) const`

Get 5G NR reference signal signal-to-noise ratio, multiply by 0.1 to get SNR in dB. (Valid value range [-200, +300] and INVALID\_SIGNAL\_STRENGTH\_VALUE i.e. unavailable). (-200 = -20.0 dB, +300 = 30dB). INVALID\_SIGNAL\_STRENGTH\_VALUE indicates that modem is not in ENDC connected mode.

#### Returns

5G NR signal-to-noise.

### 5.3.1.40 class telux::tel::VoiceServiceInfo

[VoiceServiceInfo](#) is a container class for obtaining serving state details like phone is registered to home network, roaming, in service, out of service or only emergency calls allowed.

## Public member functions

- [VoiceServiceInfo](#) ([VoiceServiceState](#) voiceServiceState, [VoiceServiceDenialCause](#) denialCause)
- [VoiceServiceState](#) getVoiceServiceState ()
- [VoiceServiceDenialCause](#) getVoiceServiceDenialCause ()
- bool isEmergency ()
- bool isInService ()
- bool isOutOfService ()

### 5.3.1.40.1 Constructors and Destructors

**5.3.1.40.1.1** `telux::tel::VoiceServiceInfo::VoiceServiceInfo ( VoiceServiceState voiceServiceState, VoiceServiceDenialCause denialCause )`

### 5.3.1.40.2 Member Function Documentation

**5.3.1.40.2.1** `VoiceServiceState telux::tel::VoiceServiceInfo::getVoiceServiceState ( )`

Get voice service state.

#### Returns

[VoiceServiceState](#)

**5.3.1.40.2.2** `VoiceServiceDenialCause telux::tel::VoiceServiceInfo::getVoiceServiceDenialCause ( )`

Get Voice service denial cause

#### Returns

[VoiceServiceDenialCause](#)

**5.3.1.40.2.3** `bool telux::tel::VoiceServiceInfo::isEmergency ( )`

Check if phone service is in emergency mode (i.e Only emergency numbers are allowed)

**5.3.1.40.2.4** `bool telux::tel::VoiceServiceInfo::isInService ( )`

Check if phone is registered to home network or roaming network, phone is in service mode

**5.3.1.40.2.5** `bool telux::tel::VoiceServiceInfo::isOutOfService ( )`

check if phone not registered, phone is in out of service mode

## 5.3.2 Enumeration Type Documentation

### 5.3.2.1 enum telux::tel::CellType [strong]

Defines all the cell info types.

#### Enumerator

**GSM**  
**CDMA**  
**LTE**  
**WCDMA**  
**TDSCDMA**

### 5.3.2.2 enum telux::tel::ECallVariant [strong]

ECall Variant

#### Enumerator

**ECALL\_TEST** Initiate a test voice eCall with a configured telephone number stored in the USIM.  
**ECALL\_EMERGENCY** Initiate an emergency eCall. The trigger can be a manually initiated eCall or automatically initiated eCall.  
**ECALL\_VOICE** Initiate a regular voice call with capability to transfer an MSD.

### 5.3.2.3 enum telux::tel::EmergencyCallType [strong]

Emergency Call Type

#### Enumerator

**CALL\_TYPE\_ECALL** eCall (0x0C)

### 5.3.2.4 enum telux::tel::ECallMsdTransmissionStatus [strong]

MSD Transmission Status

#### Enumerator

**SUCCESS** In-band MSD transmission is successful  
**FAILURE** In-band MSD transmission failed  
**MSD\_TRANSMISSION\_STARTED** In-band MSD transmission started  
**NACK\_OUT\_OF\_ORDER** Out of order NACK message detected during in-band MSD transmission  
**ACK\_OUT\_OF\_ORDER** Out of order ACK message detected during in-band MSD transmission  
**START\_RECEIVED** SEND-MSD(START) is received and SYNC is locked during in-band MSD transmission  
**LL\_ACK\_RECEIVED** Link-Layer Acknowledgement(LL-ACK) is received during in-band MSD transmission  
**OUTBAND\_MSD\_TRANSMISSION\_STARTED** Outband MSD transmission started in NG eCall  
**OUTBAND\_MSD\_TRANSMISSION\_SUCCESS** Outband MSD transmission succeeded in NG eCall  
**OUTBAND\_MSD\_TRANSMISSION\_FAILURE** Outband MSD transmission failed in NG eCall

### 5.3.2.5 enum telux::tel::ECallCategory [strong]

ECall category

#### Enumerator

**VOICE\_EMER\_CAT\_AUTO\_ECALL** Automatic emergency call  
**VOICE\_EMER\_CAT\_MANUAL** Manual emergency call

### 5.3.2.6 enum telux::tel::ECallVehicleType

Represents a vehicle class as per European eCall MSD standard. i.e. EN 15722.

#### Enumerator

**PASSENGER\_VEHICLE\_CLASS\_M1**  
**BUSES\_AND\_COACHES\_CLASS\_M2**  
**BUSES\_AND\_COACHES\_CLASS\_M3**  
**LIGHT\_COMMERCIAL\_VEHICLES\_CLASS\_N1**  
**HEAVY\_DUTY\_VEHICLES\_CLASS\_N2**  
**HEAVY\_DUTY\_VEHICLES\_CLASS\_N3**  
**MOTOR\_CYCLES\_CLASS\_L1E**  
**MOTOR\_CYCLES\_CLASS\_L2E**  
**MOTOR\_CYCLES\_CLASS\_L3E**  
**MOTOR\_CYCLES\_CLASS\_L4E**  
**MOTOR\_CYCLES\_CLASS\_L5E**  
**MOTOR\_CYCLES\_CLASS\_L6E**  
**MOTOR\_CYCLES\_CLASS\_L7E**

### 5.3.2.7 enum telux::tel::ECallOptionalDataType [strong]

Represents OptionalDataType class as per European eCall MSD standard. i.e. EN 15722.

#### Enumerator

**ECALL\_DEFAULT**

### 5.3.2.8 enum telux::tel::ECallMode [strong]

Represents eCall operating mode

#### Enumerator

**NORMAL** eCall and normal voice calls are allowed  
**ECALL\_ONLY** Only eCall is allowed  
**NONE** Invalid mode

### 5.3.2.9 enum telux::tel::ECallModeReason [strong]

Represents eCall operating mode change reason

**Enumerator**

**NORMAL** eCall operating mode changed due to normal operation like setting of eCall mode  
**ERA\_GLONASS** eCall operating mode changed due to ERA-GLONASS operation

**5.3.2.10 enum telux::tel::HlapTimerStatus [strong]**

Represents the status of an eCall High Level Application Protocol(HLAP) timer that is maintained by the UE state machine.

**Enumerator**

**UNKNOWN** Unknown  
**INACTIVE** eCall Timer is Inactive i.e it has not started or it has stopped/expired  
**ACTIVE** eCall Timer is Active i.e it has started but not yet stopped/expired

**5.3.2.11 enum telux::tel::HlapTimerEvent [strong]**

Represents an event causing a change in the the status of eCall High Level Application Protocol (HLAP) timer that is maintained by the UE state machine.

Timer STARTED notification is provided when the timer moves from INACTIVE to ACTIVE state. Timer STOPPED notification is provided when the timer moves from ACTIVE to INACTIVE state, after its underlying condition is satisfied. Timer EXPIRED notification is provided when the timer moves from ACTIVE to INACTIVE state, after its underlying condition not satisfied until its timeout.

**Enumerator**

**UNKNOWN** Unknown  
**UNCHANGED** No change in timer status  
**STARTED** eCall Timer is Started  
**STOPPED** eCall Timer is Stopped  
**EXPIRED** eCall Timer is expired

**5.3.2.12 enum telux::tel::EcallNumType [strong]**

Configuration that represents the type of the number to be dialed when an automotive emergency call is initiated.

**Enumerator**

**DEFAULT**  
**OVERRIDDEN**

**5.3.2.13 enum telux::tel::EcallConfigType**

Defines the supported ECall configuration parameters

**Enumerator**

**ECALL\_CONFIG\_MUTE\_RX\_AUDIO** Mute the local audio device during MSD transmission  
**ECALL\_CONFIG\_NUM\_TYPE** Decides which number needs to be dialed when an eCall is initiated  
**ECALL\_CONFIG\_OVERRIDDEN\_NUM** User configured/overridden number that will be dialed for

eCall

**ECALL\_CONFIG\_USE\_CANNED\_MSD** Use the pre-defined MSD in modem for eCall  
**ECALL\_CONFIG\_GNSS\_UPDATE\_INTERVAL** Time interval in milliseconds, at which modem updates the GNSS information in its internally generated MSD  
**ECALL\_CONFIG\_T2\_TIMER** T2 timer value  
**ECALL\_CONFIG\_T7\_TIMER** T7 timer value  
**ECALL\_CONFIG\_T9\_TIMER** T9 timer value  
**ECALL\_CONFIG\_MSD\_VERSION** MSD version to be used by modem when it internally generates MSD i.e when MSD is not sent by application and also canned MSD is not used  
**ECALL\_CONFIG\_COUNT**

### 5.3.2.14 enum telux::tel::RadioState [strong]

Defines the radio state

#### Enumerator

**RADIO\_STATE\_OFF** Radio is explicitly powered off  
**RADIO\_STATE\_UNAVAILABLE** Radio unavailable (eg, resetting or not booted)  
**RADIO\_STATE\_ON** Radio is on

### 5.3.2.15 enum telux::tel::ServiceState [strong]

Defines the service states

**Deprecated** Use requestVoiceServiceState() API or to know the status of phone

#### Enumerator

**EMERGENCY\_ONLY** Only emergency calls allowed  
**IN\_SERVICE** Normal operation, device is registered with a carrier and online  
**OUT\_OF\_SERVICE** Device is not registered with any carrier  
**RADIO\_OFF** Device radio is off - Airplane mode for example

### 5.3.2.16 enum telux::tel::RadioTechnology [strong]

Defines all available radio access technologies

#### Enumerator

**RADIO\_TECH\_UNKNOWN** Network type is unknown  
**RADIO\_TECH\_GPRS** Network type is GPRS  
**RADIO\_TECH\_EDGE** Network type is EDGE  
**RADIO\_TECH\_UMTS** Network type is UMTS  
**RADIO\_TECH\_IS95A** Network type is IS95A  
**RADIO\_TECH\_IS95B** Network type is IS95B  
**RADIO\_TECH\_1xRTT** Network type is 1xRTT  
**RADIO\_TECH\_EVDO\_0** Network type is EVDO revision 0  
**RADIO\_TECH\_EVDO\_A** Network type is EVDO revision A  
**RADIO\_TECH\_HSDPA** Network type is HSDPA  
**RADIO\_TECH\_HSUPA** Network type is HSUPA

**RADIO\_TECH\_HSPA** Network type is HSPA  
**RADIO\_TECH\_EVDO\_B** Network type is EVDO revision B  
**RADIO\_TECH\_EHRPD** Network type is eHRPD  
**RADIO\_TECH\_LTE** Network type is LTE  
**RADIO\_TECH\_HSPAP** Network type is HSPA+  
**RADIO\_TECH\_GSM** Network type is GSM, Only supports voice  
**RADIO\_TECH\_TD\_SCDMA** Network type is TD SCDMA  
**RADIO\_TECH\_IWLAN** Network type is TD IWLAN  
**RADIO\_TECH\_LTE\_CA** Network type is LTE CA

### 5.3.2.17 enum telux::tel::RATCapability [strong]

Defines all available RAT capabilities for each subscription

#### Enumerator

**AMPS**  
**CDMA**  
**HDR**  
**GSM**  
**WCDMA**  
**LTE**  
**TDS**

### 5.3.2.18 enum telux::tel::VoiceServiceTechnology [strong]

Defines all voice support available on device

#### Enumerator

**VOICE\_TECH\_GW\_CSFB**  
**VOICE\_TECH\_1x\_CSFB**  
**VOICE\_TECH\_VOLTE**

### 5.3.2.19 enum telux::tel::OperatingMode [strong]

Defines operating modes of the device.

#### Enumerator

**ONLINE** Online mode  
**AIRPLANE** Low Power mode i.e temporarily disabled RF  
**FACTORY\_TEST** Special mode for manufacturer use  
**OFFLINE** Device has deactivated RF and partially shutdown  
**RESETTING** Device is in process of power cycling  
**SHUTTING\_DOWN** Device is in process of shutting down  
**PERSISTENT\_LOW\_POWER** Persists low power mode even on reset

### 5.3.2.20 enum telx::tel::SignalStrengthLevel [strong]

Defines all the signal levels that [SignalStrength](#) class can return where level 1 is low and level 5 is high.

#### Enumerator

**LEVEL\_1**  
**LEVEL\_2**  
**LEVEL\_3**  
**LEVEL\_4**  
**LEVEL\_5**  
**LEVEL\_UNKNOWN**

### 5.3.2.21 enum telx::tel::VoiceServiceState [strong]

Defines the voice service states

#### Enumerator

**NOT\_REG\_AND\_NOT\_SEARCHING** Not registered, MT is not currently searching a new operator to register  
**REG\_HOME** Registered, home network  
**NOT\_REG\_AND\_SEARCHING** Not registered, but MT is currently searching a new operator to register  
**REG\_DENIED** Registration denied  
**UNKNOWN** Unknown  
**REG\_ROAMING** Registered, roaming  
**NOT\_REG\_AND\_EMERGENCY\_AVAILABLE\_AND\_NOT\_SEARCHING** Same as NOT\_REG\_AND\_NOT\_SEARCHING but indicates that emergency calls are enabled  
**NOT\_REG\_AND\_EMERGENCY\_AVAILABLE\_AND\_SEARCHING** Same as NOT\_REG\_AND\_SEARCHING but indicates that emergency calls are enabled  
**REG\_DENIED\_AND\_EMERGENCY\_AVAILABLE** Same as REG\_DENIED but indicates that emergency calls are enabled  
**UNKNOWN\_AND\_EMERGENCY\_AVAILABLE** Same as UNKNOWN but indicates that emergency calls are enabled

### 5.3.2.22 enum telx::tel::VoiceServiceDenialCause [strong]

Defines the voice service denial cause why voice service state registration was denied See 3GPP TS 24.008, 10.5.3.6 and Annex G.

#### Enumerator

**UNDEFINED** Undefined  
**GENERAL** General  
**AUTH\_FAILURE** Authentication Failure  
**IMSI\_UNKNOWN** IMSI unknown in HLR  
**ILLEGAL\_MS** Illegal Mobile Station (MS), network refuses service to the MS either because an identity of the MS is not acceptable to the network or because the MS does not pass the authentication check  
**IMSI\_UNKNOWN\_VLR** IMSI unknown in Visitors Location Register (VLR)



**IMEI\_NOT\_ACCEPTED** Network does not accept emergency call establishment using an IMEI or not accept attach procedure for emergency services using an IMEI

**ILLEGAL\_ME** ME used is not acceptable to the network

**GPRS\_SERVICES\_NOT\_ALLOWED** Not allowed to operate GPRS services.

**GPRS\_NON\_GPRS\_NOT\_ALLOWED** Not allowed to operate either GPRS or non-GPRS services

**MS\_IDENTITY\_FAILED** the network cannot derive the MS's identity from the P-TMSI/GUTI.

**IMPLICITLY\_DETACHED** network has implicitly detached the MS

**GPRS\_NOT\_ALLOWED\_IN\_PLMN** GPRS services not allowed in this PLMN

**MSC\_TEMPORARILY\_NOT\_REACHABLE** MSC temporarily not reachable

**SMS\_PROVIDED\_VIA\_GPRS** SMS provided via GPRS in this routing area

**NO\_PDP\_CONTEXT\_ACTIVATED** No PDP context activated

**PLMN\_NOT\_ALLOWED** if the network initiates a detach request or UE requests a services, in a PLMN where the MS, by subscription or due to operator determined barring is not allowed to operate.

**LOCATION\_AREA\_NOT\_ALLOWED** network initiates a detach request, in a location area where the HPLMN determines that the MS, by subscription, is not allowed to operate or roaming subscriber the subscriber is denied service even if other PLMNs are available on which registration was possible

**ROAMING\_NOT\_ALLOWED** Roaming not allowed in this Location Area

**NO\_SUITABLE\_CELLS** No Suitable Cells in this Location Area

**NOT\_AUTHORIZED** Not Authorized for this CSG

**NETWORK\_FAILURE** Network Failure

**MAC\_FAILURE** MAC failure

**SYNC\_FAILURE** USIM detects that the SQN in the AUTHENTICATION REQUEST or AUTHENTICATION\_AND\_CIPHERING REQUEST message is out of range

**CONGESTION** network cannot serve a request from the MS because of congestion

**GSM\_AUTHENTICATION\_UNACCEPTABLE** GSM Authentication unacceptable

**SERVICE\_OPTION\_NOT\_SUPPORTED** Service option not supported

**SERVICE\_OPTION\_NOT\_SUBSCRIBED** Requested service option not subscribed

**SERVICE\_OPTION\_OUT\_OF\_ORDER** Service option temporarily out of order

**CALL\_NOT\_IDENTIFIED** Call cannot be identified

**RETRY\_FOR\_NEW\_CELL** Retry upon entry into a new cell

**INCORRECT\_MESSAGE** Semantically incorrect message

**INVALID\_INFO** Invalid mandatory information

**MSG\_TYPE\_NOT\_IMPLEMENTED** Message type non-existent or not implemented

**MSG\_NOT\_COMPATIBLE** Message not compatible with protocol state

**INFO\_NOT\_IMPLEMENTED** Information element non-existent or not implemented

**CONDITIONAL\_IE\_ERROR** Conditional IE error

**PROTOCOL\_ERROR\_UNSPECIFIED** Protocol error, unspecified

## 5.4 Call

This section contains APIs related to Call.

### 5.4.1 Data Structure Documentation

#### 5.4.1.1 class telux::tel::ICall

**ICall** represents a call in progress. An **ICall** cannot be directly created by the client, rather it is returned as a result of instantiating a call or from the PhoneListener when receiving an incoming call.

##### Public member functions

- virtual [telux::common::Status answer](#) (std::shared\_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status hold](#) (std::shared\_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status resume](#) (std::shared\_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status reject](#) (std::shared\_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status reject](#) (const std::string &rejectSMS, std::shared\_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status hangup](#) (std::shared\_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status playDtmfTone](#) (char tone, std::shared\_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status startDtmfTone](#) (char tone, std::shared\_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status stopDtmfTone](#) (std::shared\_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [CallState getCallState](#) ()=0
- virtual int [getCallIndex](#) ()=0
- virtual [CallDirection getCallDirection](#) ()=0
- virtual std::string [getRemotePartyNumber](#) ()=0
- virtual [CallEndCause getCallEndCause](#) ()=0
- virtual int [getPhoneId](#) ()=0
- virtual [~ICall](#) ()

### 5.4.1.1.1 Constructors and Destructors

5.4.1.1.1.1 virtual telux::tel::ICall::~ICall ( ) [virtual]

### 5.4.1.1.2 Member Function Documentation

5.4.1.1.2.1 virtual telux::common::Status telux::tel::ICall::answer ( std::shared\_ptr< telux::common::ICommandResponseCallback > *callback = nullptr* ) [pure virtual]

Allows the client to answer the call. This is only applicable for [CallState::INCOMING](#) and [CallState::WAITING](#) calls. If a Waiting call is being answered and the existing call is Active, then existing call will move to Hold state. If the existing call is on Hold already, then it will remain on Hold. The waiting call state transition from Waiting to Active.

#### Parameters

in	<i>callback</i>	- optional callback pointer to get the response of answer request below are possible error codes for callback response <ul style="list-style-type: none"> <li>• <a href="#">SUCCESS</a></li> <li>• RADIO_NOT_AVAILABLE</li> <li>• NO_MEMORY</li> <li>• MODEM_ERR</li> <li>• INTERNAL_ERR</li> <li>• INVALID_STATE</li> <li>• INVALID_CALL_ID</li> <li>• INVALID_ARGUMENTS</li> <li>• OPERATION_NOT_ALLOWED</li> <li>• GENERIC_FAILURE</li> </ul>
----	-----------------	--

#### Returns

Status of hold function i.e. success or suitable error code.

5.4.1.1.2.2 virtual telux::common::Status telux::tel::ICall::hold ( std::shared\_ptr< telux::common::ICommandResponseCallback > *callback = nullptr* ) [pure virtual]

Puts the ongoing call on hold.

**Parameters**

in	<i>callback</i>	<p>- optional callback pointer to get the response of hold request below are possible error codes for callback response</p> <ul style="list-style-type: none"> <li>• <a href="#">SUCCESS</a></li> <li>• RADIO_NOT_AVAILABLE</li> <li>• NO_MEMORY</li> <li>• MODEM_ERR</li> <li>• INTERNAL_ERR</li> <li>• INVALID_STATE</li> <li>• INVALID_CALL_ID</li> <li>• INVALID_ARGUMENTS</li> <li>• OPERATION_NOT_ALLOWED</li> <li>• GENERIC_FAILURE</li> </ul>
----	-----------------	---

**Returns**

Status of hold function i.e. success or suitable error code.

**5.4.1.1.2.3** `virtual telux::common::Status telux::tel::lCall::resume ( std::shared_ptr< telux::common::lCommandResponseCallback > callback = nullptr ) [pure virtual]`

Resumes this call from on-hold state to active state

**Parameters**

in	<i>callback</i>	<p>- optional callback pointer to get the response of resume request below are possible error codes for callback response</p> <ul style="list-style-type: none"> <li>• <a href="#">SUCCESS</a></li> <li>• RADIO_NOT_AVAILABLE</li> <li>• NO_MEMORY</li> <li>• MODEM_ERR</li> <li>• INTERNAL_ERR</li> <li>• INVALID_STATE</li> <li>• INVALID_CALL_ID</li> <li>• INVALID_ARGUMENTS</li> <li>• OPERATION_NOT_ALLOWED</li> <li>• GENERIC_FAILURE</li> </ul>
----	-----------------	---

**Returns**

Status of resume function i.e. success or suitable error code.

**5.4.1.1.2.4** `virtual telux::common::Status telux::tel::lCall::reject ( std::shared_ptr< telux::common::lCommandResponseCallback > callback = nullptr ) [pure virtual]`

Reject the incoming/waiting call. Only applicable for [CallState::INCOMING](#) and [CallState::WAITING](#) calls.

in	<i>callback</i>	- optional callback pointer to get the response of reject request below are possible error codes for callback response <ul style="list-style-type: none"> <li>• <a href="#">SUCCESS</a></li> <li>• RADIO_NOT_AVAILABLE</li> <li>• NO_MEMORY</li> <li>• MODEM_ERR</li> <li>• INTERNAL_ERR</li> <li>• INVALID_STATE</li> <li>• INVALID_CALL_ID</li> <li>• INVALID_ARGUMENTS</li> <li>• OPERATION_NOT_ALLOWED</li> <li>• GENERIC_FAILURE</li> </ul>
----	-----------------	---

**Returns**

Status of reject function i.e. success or suitable error code.

**5.4.1.1.2.5 virtual telux::common::Status telux::tel::lCall::reject ( const std::string & rejectSMS, std::shared\_ptr< telux::common::lCommandResponseCallback > callback = nullptr ) [pure virtual]**

Reject the call and send an SMS to caller. Only applicable for [CallState::INCOMING](#) and [CallState::WAITING](#) calls.

**Parameters**

in	<i>rejectSMS</i>	SMS string used to send in response to a call rejection.
in	<i>callback</i>	- optional callback pointer to get the response of rejectwithSMS request below are possible error codes for callback response <ul style="list-style-type: none"> <li>• <a href="#">SUCCESS</a></li> <li>• RADIO_NOT_AVAILABLE</li> <li>• NO_MEMORY</li> <li>• MODEM_ERR</li> <li>• INTERNAL_ERR</li> <li>• INVALID_STATE</li> <li>• INVALID_CALL_ID</li> <li>• INVALID_ARGUMENTS</li> <li>• OPERATION_NOT_ALLOWED</li> <li>• GENERIC_FAILURE</li> </ul>

**Returns**

Status of success for call [reject\(\)](#) or suitable error code.

**5.4.1.1.2.6 virtual telx::common::Status telx::tel::lCall::hangup ( std::shared\_ptr< telx::common::lCommandResponseCallback > callback = nullptr ) [pure virtual]**

Hangup the call if the call state is either active, hold, dialing, waiting or alerting.

**Parameters**

in	<i>callback</i>	- optional callback pointer to get the response of hangup request below are possible error codes for callback response <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• RADIO_NOT_AVAILABLE</li> <li>• NO_MEMORY</li> <li>• MODEM_ERR</li> <li>• INTERNAL_ERR</li> <li>• INVALID_STATE</li> <li>• INVALID_CALL_ID</li> <li>• INVALID_ARGUMENTS</li> <li>• OPERATION_NOT_ALLOWED</li> <li>• GENERIC_FAILURE</li> </ul>
----	-----------------	--

**Returns**

Status of hangup i.e. success or suitable error code.

**5.4.1.1.2.7 virtual telx::common::Status telx::tel::lCall::playDtmfTone ( char tone, std::shared\_ptr< telx::common::lCommandResponseCallback > callback = nullptr ) [pure virtual]**

Play a DTMF tone and stop it. The interval for which the tone is played is dependent on the system implementation. If continuous DTMF tone is playing, it will be stopped.

**Parameters**

in	<i>tone</i>	- a single character with one of 12 values: 0-9, *, #.
in	<i>callback</i>	- Optional callback pointer to get the result of playDtmfTones function

**Returns**

Status of playDtmfTones i.e. success or suitable error code.

**5.4.1.1.2.8 virtual telx::common::Status telx::tel::lCall::startDtmfTone ( char tone, std::shared\_ptr< telx::common::lCommandResponseCallback > callback = nullptr ) [pure virtual]**

Starts a continuous DTMF tone. To terminate the continuous DTMF tone, stopDtmfTone API needs to be invoked explicitly.

in	<i>tone</i>	- a single character with one of 12 values: 0-9, *, #.
in	<i>callback</i>	- Optional callback pointer to get the result of startDtmfTone function.

**Returns**

Status of startDtmfTone i.e. success or suitable error code.

**5.4.1.1.2.9 virtual telux::common::Status telux::tel::ICall::stopDtmfTone ( std::shared\_ptr< telux::common::ICommandResponseCallback > *callback* = nullptr ) [pure virtual]**

Stop the currently playing continuous DTMF tone.

**Parameters**

in	<i>callback</i>	- Optional callback pointer to get the result of stopDtmfTone function.
----	-----------------	---

**Returns**

Status of stopDtmfTone i.e. success or suitable error code.

**5.4.1.1.2.10 virtual CallState telux::tel::ICall::getCallState ( ) [pure virtual]**

Get the current state of the call, such as ringing, in progress etc.

**Returns**

CallState - enumeration representing call State

**5.4.1.1.2.11 virtual int telux::tel::ICall::getCallIndex ( ) [pure virtual]**

Get the unique index of the call assigned by Telephony subsystem

**Returns**

Call Index

**5.4.1.1.2.12 virtual CallDirection telux::tel::ICall::getCallDirection ( ) [pure virtual]**

Get the direction of the call

**Returns**

CallDirection - enumeration representing call direction i.e. INCOMING/ OUTGOING

**5.4.1.1.2.13 virtual std::string telux::tel::ICall::getRemotePartyNumber ( ) [pure virtual]**

Get the dailing number

**Returns**

Phone Number to which the call was dialed out Empty string in case of INCOMING call direction

**5.4.1.1.2.14 virtual CallEndCause telux::tel::ICall::getCallEndCause ( ) [pure virtual]**

Get the cause of the termination of the call.

**Returns**

Enum representing call end cause.

**5.4.1.1.2.15 virtual int telux::tel::ICall::getPhoneId ( ) [pure virtual]**

Get id of the phone object which represents the network/SIM on which the call is in progress.

**Returns**

Phone Id.

**5.4.1.2 class telux::tel::ICallListener**

A listener class for monitoring changes in call, including call state change and ECall state change. Override the methods for the state that you wish to receive updates for.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

**Public member functions**

- virtual void [onIncomingCall](#) (std::shared\_ptr< ICall > call)
- virtual void [onCallInfoChange](#) (std::shared\_ptr< ICall > call)
- virtual void [onECallMsdTransmissionStatus](#) (int phoneId, [telux::common::ErrorCode](#) errorCode)
- virtual void [onECallMsdTransmissionStatus](#) (int phoneId, [telux::tel::ECallMsdTransmissionStatus](#) msdTransmissionStatus)
- virtual void [onECallHlapTimerEvent](#) (int phoneId, [ECallHlapTimerEvents](#) timersStatus)
- virtual [~ICallListener](#) ()

**5.4.1.2.1 Constructors and Destructors**



5.4.1.2.1.1 virtual telux::tel::ICallListener::~ICallListener ( ) [virtual]

#### 5.4.1.2.2 Member Function Documentation

5.4.1.2.2.1 virtual void telux::tel::ICallListener::onIncomingCall ( std::shared\_ptr< ICall > *call* ) [virtual]

This function is called when device receives an incoming/waiting call.

##### Parameters

in	<i>call</i>	- Pointer to <a href="#">ICall</a> instance
----	-------------	---

5.4.1.2.2.2 virtual void telux::tel::ICallListener::onCallInfoChange ( std::shared\_ptr< ICall > *call* ) [virtual]

This function is called when there is a change in call attributes

##### Parameters

in	<i>call</i>	- Pointer to <a href="#">ICall</a> instance
----	-------------	---

5.4.1.2.2.3 virtual void telux::tel::ICallListener::onECallMsdTransmissionStatus ( int *phoneId*, telux::common::ErrorCode *errorCode* ) [virtual]

This function is called when device completes MSD Transmission.

##### Parameters

in	<i>phoneId</i>	- Unique Id of phone on which MSD Transmission Status is being reported
in	<i>status</i>	- Indicates MSD Transmission status i.e. success or failure

**Deprecated** Use another [onECallMsdTransmissionStatus\(\)](#) API with argument `ECallMsdTransmissionStatus`

5.4.1.2.2.4 virtual void telux::tel::ICallListener::onECallMsdTransmissionStatus ( int *phoneId*, telux::tel::ECallMsdTransmissionStatus *msdTransmissionStatus* ) [virtual]

This function is called when MSD Transmission status is changed.

##### Parameters

in	<i>phoneId</i>	- Unique Id of phone on which MSD Transmission Status is being reported
in	<i>msdTransmissionStatus</i>	- Indicates MSD Transmission status <code>ECallMsdTransmissionStatus</code>

#### 5.4.1.2.2.5 virtual void telux::tel::ICallListener::onECallHlapTimerEvent ( int *phoneId*, ECallHlap↔ TimerEvents *timersStatus* ) [virtual]

This function is called when the eCall High Level Application Protocol(HLAP) timers status is changed.

#### Parameters

in	<i>phoneId</i>	- Unique Id of phone on which HLAP timer status is being reported
in	<i>timersStatus</i>	- Indicates the HLAP timer event <a href="#">ECallHlapTimerEvents</a>

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.4.1.3 class telux::tel::ICallManager

Call Manager is the primary interface for call related operations Allows to conference calls, swap calls, make normal voice call and emergency call, send and update MSD pdu.

#### Public member functions

- virtual [telux::common::ServiceStatus](#) [getServiceStatus](#) ()=0
- virtual [telux::common::Status](#) [makeCall](#) (int phoneId, const std::string &dialNumber, std::shared\_ptr< [IMakeCallCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status](#) [makeECall](#) (int phoneId, const [ECallMsData](#) &eCallMsData, int category, int variant, std::shared\_ptr< [IMakeCallCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status](#) [makeECall](#) (int phoneId, const std::string dialNumber, const [ECallMsData](#) &eCallMsData, int category, std::shared\_ptr< [IMakeCallCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status](#) [makeECall](#) (int phoneId, const std::vector< uint8\_t > &msdPdu, int category, int variant, [MakeCallCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [makeECall](#) (int phoneId, const std::string dialNumber, const std::vector< uint8\_t > &msdPdu, int category, [MakeCallCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [makeECall](#) (int phoneId, int category, int variant, [MakeCallCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [makeECall](#) (int phoneId, const std::string dialNumber, int category, [MakeCallCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [updateECallMsData](#) (int phoneId, const [ECallMsData](#) &eCallMsData, std::shared\_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status](#) [updateECallMsData](#) (int phoneId, const std::vector< uint8\_t > &msdPdu, [telux::common::ResponseCallback](#) callback)=0
- virtual [telux::common::Status](#) [requestECallHlapTimerStatus](#) (int phoneId, [ECallHlapTimerStatusCallback](#) callback)=0

- virtual `std::vector< std::shared_ptr< ICall > > getInProgressCalls ()=0`
- virtual `telux::common::Status conference (std::shared_ptr< ICall > call1, std::shared_ptr< ICall > call2, std::shared_ptr< telux::common:: ICommandResponseCallback > callback=nullptr)=0`
- virtual `telux::common::Status swap (std::shared_ptr< ICall > callToHold, std::shared_ptr< ICall > callToActivate, std::shared_ptr< telux::common:: ICommandResponseCallback > callback=nullptr)=0`
- virtual `telux::common::Status registerListener (std::shared_ptr< telux::tel:: ICallListener > listener)=0`
- virtual `telux::common::Status removeListener (std::shared_ptr< telux::tel:: ICallListener > listener)=0`
- virtual `~ICallManager ()`

**5.4.1.3.1 Constructors and Destructors**

**5.4.1.3.1.1** virtual `telux::tel:: ICallManager::~ICallManager ( ) [virtual]`

**5.4.1.3.2 Member Function Documentation**

**5.4.1.3.2.1** virtual `telux::common::ServiceStatus telux::tel:: ICallManager::getServiceStatus ( ) [pure virtual]`

This status indicates whether the `ICallManager` object is in a usable state.

**Returns**

SERVICE\_AVAILABLE - If CallManager is ready for service. SERVICE\_UNAVAILABLE - If CallManager is temporarily unavailable. SERVICE\_FAILED - If CallManager encountered an irrecoverable failure.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.4.1.3.2.2** virtual `telux::common::Status telux::tel:: ICallManager::makeCall ( int phoneId, const std::string & dialNumber, std::shared_ptr< IMakeCallCallback > callback = nullptr ) [pure virtual]`

Initiate a voice call. This API can also be used for e911/e112 type of regular emergency call. This is not meant for an automotive eCall.

**Parameters**

in	<i>phoneId</i>	Represents phone corresponding to which on make call operation is performed
in	<i>dialNumber</i>	String representing the dialing number

in	<i>callback</i>	<p>Optional callback pointer to get the response of makeCall request. Possible(not exhaustive) error codes for callback response</p> <ul style="list-style-type: none"> <li>• <code>telux::common::ErrorCode::SUCCESS</code></li> <li>• <code>telux::common::ErrorCode::RADIO_NOT_AVAILABLE</code></li> <li>• <code>telux::common::ErrorCode::DIAL_MODIFIED_TO_USSD</code></li> <li>• <code>telux::common::ErrorCode::DIAL_MODIFIED_TO_SS</code></li> <li>• <code>telux::common::ErrorCode::DIAL_MODIFIED_TO_DIAL</code></li> <li>• <code>telux::common::ErrorCode::INVALID_ARGUMENTS</code></li> <li>• <code>telux::common::ErrorCode::NO_MEMORY</code></li> <li>• <code>telux::common::ErrorCode::INVALID_STATE</code></li> <li>• <code>telux::common::ErrorCode::NO_RESOURCES</code></li> <li>• <code>telux::common::ErrorCode::INTERNAL_ERR</code></li> <li>• <code>telux::common::ErrorCode::FDN_CHECK_FAILURE</code></li> <li>• <code>telux::common::ErrorCode::MODEM_ERR</code></li> <li>• <code>telux::common::ErrorCode::NO_SUBSCRIPTION</code></li> <li>• <code>telux::common::ErrorCode::NO_NETWORK_FOUND</code></li> <li>• <code>telux::common::ErrorCode::INVALID_CALL_ID</code></li> <li>• <code>telux::common::ErrorCode::DEVICE_IN_USE</code></li> <li>• <code>telux::common::ErrorCode::MODE_NOT_SUPPORTED</code></li> <li>• <code>telux::common::ErrorCode::ABORTED</code></li> <li>• <code>telux::common::ErrorCode::GENERIC_FAILURE</code></li> </ul>
----	-----------------	---

### Returns

Status of makeCall i.e. success or suitable status code.

**5.4.1.3.2.3** `virtual telux::common::Status telux::tel::ICallManager::makeECall ( int phoneld, const E↔ CallMsdData & eCallMsdData, int category, int variant, std::shared_ptr< IMakeCallCallback > callback = nullptr ) [pure virtual]`

Initiate an automotive eCall.

**Parameters**

in	<i>phoneId</i>	Represents phone corresponding to which make eCall operation is performed
in	<i>eCallMsddata</i>	The structure containing required fields to create eCall Minimum Set of Data (MSD)
in	<i>category</i>	<a href="#">ECallCategory</a>
in	<i>variant</i>	<a href="#">ECallVariant</a>
in	<i>callback</i>	Optional callback pointer to get the response of makeECall request. Possible(not exhaustive) error codes for callback response <ul style="list-style-type: none"> <li>• <a href="#">telux::common::ErrorCode::SUCCESS</a></li> <li>• <a href="#">telux::common::ErrorCode::RADIO_NOT_AVAILABLE</a></li> <li>• <a href="#">telux::common::ErrorCode::NO_MEMORY</a></li> <li>• <a href="#">telux::common::ErrorCode::MODEM_ERR</a></li> <li>• <a href="#">telux::common::ErrorCode::INTERNAL_ERR</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_STATE</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_CALL_ID</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_ARGUMENTS</a></li> <li>• <a href="#">telux::common::ErrorCode::OPERATION_NOT_ALLOWED</a></li> <li>• <a href="#">telux::common::ErrorCode::GENERIC_FAILURE</a></li> </ul>

**Returns**

Status of makeECall i.e. success or suitable status code.

```
5.4.1.3.2.4 virtual telux::common::Status telux::tel::ICallManager::makeECall ( int phoneId, const std::string dialNumber, const ECallMsddata & eCallMsddata, int category, std::shared_ptr<IMakeCallCallback > callback = nullptr ) [pure virtual]
```

Initiate an automotive eCall to the specified phone number for TPS eCall. It will be treated like a regular voice call by the UE and the network.

**Parameters**

in	<i>phoneId</i>	Represents phone corresponding to which make eCall operation is performed
in	<i>dialNumber</i>	String representing the dialing number
in	<i>eCallMsdata</i>	The structure containing required fields to create eCall Minimum Set of Data (MSD)
in	<i>category</i>	<a href="#">ECallCategory</a>
in	<i>callback</i>	Optional callback pointer to get the response of makeECall request. Possible(not exhaustive) error codes for callback response <ul style="list-style-type: none"> <li>• <a href="#">telux::common::ErrorCode::SUCCESS</a></li> <li>• <a href="#">telux::common::ErrorCode::RADIO_NOT_AVAILABLE</a></li> <li>• <a href="#">telux::common::ErrorCode::NO_MEMORY</a></li> <li>• <a href="#">telux::common::ErrorCode::MODEM_ERR</a></li> <li>• <a href="#">telux::common::ErrorCode::INTERNAL_ERR</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_STATE</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_CALL_ID</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_ARGUMENTS</a></li> <li>• <a href="#">telux::common::ErrorCode::OPERATION_NOT_ALLOWED</a></li> <li>• <a href="#">telux::common::ErrorCode::GENERIC_FAILURE</a></li> </ul>

**Returns**

Status of makeECall i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

```
5.4.1.3.2.5 virtual telux::common::Status telux::tel::ICallManager::makeECall ( int phoneId, const
std::vector< uint8_t > & msdPdu, int category, int variant, MakeCallCallback callback =
nullptr ) [pure virtual]
```

Initiate an automotive eCall with raw MSD pdu.

**Parameters**

in	<i>phoneId</i>	Represents phone corresponding to which on make eCall operation is performed
in	<i>msdPdu</i>	Encoded MSD(Minimum Set of Data) PDU as per spec EN 15722 2015 or GOST R 54620-2011/33464-2015
in	<i>category</i>	<a href="#">ECallCategory</a>
in	<i>variant</i>	<a href="#">ECallVariant</a>

in	<i>callback</i>	<p>Callback function to get the response of makeECall request. Possible(not exhaustive) error codes for callback response</p> <ul style="list-style-type: none"> <li>• <a href="#">telux::common::ErrorCode::SUCCESS</a></li> <li>• <a href="#">telux::common::ErrorCode::RADIO_NOT_AVAILABLE</a></li> <li>• <a href="#">telux::common::ErrorCode::NO_MEMORY</a></li> <li>• <a href="#">telux::common::ErrorCode::MODEM_ERR</a></li> <li>• <a href="#">telux::common::ErrorCode::INTERNAL_ERR</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_STATE</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_CALL_ID</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_ARGUMENTS</a></li> <li>• <a href="#">telux::common::ErrorCode::OPERATION_NOT_ALLOWED</a></li> <li>• <a href="#">telux::common::ErrorCode::GENERIC_FAILURE</a></li> </ul>
----	-----------------	---

### Returns

Status of makeECall i.e. success or suitable status code.

**5.4.1.3.2.6 virtual telux::common::Status telux::tel::lCallManager::makeECall ( int *phoneId*, const std::string *dialNumber*, const std::vector< uint8\_t > & *msdPdu*, int *category*, MakeCallCallback *callback* = nullptr ) [pure virtual]**

Initiate an automotive eCall with raw MSD pdu, to the specified phone number for TPS eCall. It will be treated like a regular voice call by the UE and the network.

### Parameters

in	<i>phoneId</i>	Represents phone corresponding to which on make eCall operation is performed
in	<i>dialNumber</i>	String representing the dialing number
in	<i>msdPdu</i>	Encoded MSD(Minimum Set of Data) PDU as per spec EN 15722 2015 or GOST R 54620-2011/33464-2015
in	<i>category</i>	<a href="#">ECallCategory</a>
in	<i>callback</i>	<p>Callback function to get the response of makeECall request. Possible(not exhaustive) error codes for callback response</p> <ul style="list-style-type: none"> <li>• <a href="#">telux::common::ErrorCode::SUCCESS</a></li> <li>• <a href="#">telux::common::ErrorCode::RADIO_NOT_AVAILABLE</a></li> <li>• <a href="#">telux::common::ErrorCode::NO_MEMORY</a></li> <li>• <a href="#">telux::common::ErrorCode::MODEM_ERR</a></li> <li>• <a href="#">telux::common::ErrorCode::INTERNAL_ERR</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_STATE</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_CALL_ID</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_ARGUMENTS</a></li> <li>• <a href="#">telux::common::ErrorCode::OPERATION_NOT_ALLOWED</a></li> <li>• <a href="#">telux::common::ErrorCode::GENERIC_FAILURE</a></li> </ul>

**Returns**

Status of makeECall i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.4.1.3.2.7 `virtual telux::common::Status telux::tel::ICallManager::makeECall ( int phoneId, int category, int variant, MakeCallCallback callback = nullptr ) [pure virtual]`

Initiate an automotive eCall without transmitting Minimum Set of Data (MSD) at call connect.

**Parameters**

in	<i>phoneId</i>	Represents phone corresponding to which make eCall operation is performed
in	<i>category</i>	<a href="#">ECallCategory</a>
in	<i>variant</i>	<a href="#">ECallVariant</a>
in	<i>callback</i>	Optional callback function to get the response of makeECall request. Possible(not exhaustive) error codes for callback response <ul style="list-style-type: none"> <li>• <a href="#">telux::common::ErrorCode::SUCCESS</a></li> <li>• <a href="#">telux::common::ErrorCode::RADIO_NOT_AVAILABLE</a></li> <li>• <a href="#">telux::common::ErrorCode::NO_MEMORY</a></li> <li>• <a href="#">telux::common::ErrorCode::MODEM_ERR</a></li> <li>• <a href="#">telux::common::ErrorCode::INTERNAL_ERR</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_STATE</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_CALL_ID</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_ARGUMENTS</a></li> <li>• <a href="#">telux::common::ErrorCode::OPERATION_NOT_ALLOWED</a></li> <li>• <a href="#">telux::common::ErrorCode::GENERIC_FAILURE</a></li> </ul>

**Returns**

Status of makeECall i.e. success or suitable status code.

#### 5.4.1.3.2.8 `virtual telux::common::Status telux::tel::ICallManager::makeECall ( int phoneId, const std::string dialNumber, int category, MakeCallCallback callback = nullptr ) [pure virtual]`

Initiate an automotive eCall to the specified phone number for TPS eCall, without transmitting Minimum Set of Data(MSD) at call connect. It will be treated like a regular voice call by the UE and the network.



in	<i>phoneId</i>	Represents phone corresponding to which make eCall operation is performed
in	<i>dialNumber</i>	String representing the dialing number
in	<i>category</i>	<a href="#">ECallCategory</a>
in	<i>callback</i>	Optional callback function to get the response of makeECall request. Possible(not exhaustive) error codes for callback response <ul style="list-style-type: none"> <li>• <a href="#">telux::common::ErrorCode::SUCCESS</a></li> <li>• <a href="#">telux::common::ErrorCode::RADIO_NOT_AVAILABLE</a></li> <li>• <a href="#">telux::common::ErrorCode::NO_MEMORY</a></li> <li>• <a href="#">telux::common::ErrorCode::MODEM_ERR</a></li> <li>• <a href="#">telux::common::ErrorCode::INTERNAL_ERR</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_STATE</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_CALL_ID</a></li> <li>• <a href="#">telux::common::ErrorCode::INVALID_ARGUMENTS</a></li> <li>• <a href="#">telux::common::ErrorCode::OPERATION_NOT_ALLOWED</a></li> <li>• <a href="#">telux::common::ErrorCode::GENERIC_FAILURE</a></li> </ul>

### Returns

Status of makeECall i.e. success or suitable status code.

**5.4.1.3.2.9** `virtual telux::common::Status telux::tel::ICallManager::updateECallMsd ( int phoneId, const ECallMsdData & eCallMsd, std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]`

Update the eCall MSD in modem to be sent to Public Safety Answering [Point](#) (PSAP) when requested.

### Parameters

in	<i>phoneId</i>	Represents phone corresponding to which updateECallMsd operation is performed
in	<i>eCallMsd</i>	The data structure represents the Minimum Set of Data (MSD)
in	<i>callback</i>	Optional callback pointer to get the response of updateECallMsd.

### Returns

Status of updateECallMsd i.e. success or suitable error code.

**5.4.1.3.2.10** `virtual telux::common::Status telux::tel::ICallManager::updateECallMsd ( int phoneId, const std::vector< uint8_t > & msdPdu, telux::common::ResponseCallback callback ) [pure virtual]`

Update the eCall MSD in modem to be sent to Public Safety Answering [Point](#) (PSAP) when requested.

in	<i>phoneId</i>	Represents phone corresponding to which updateECallMsd operation is performed
in	<i>msdPdu</i>	Encoded MSD(Minimum Set of Data) PDU as per spec EN 15722 2015 or GOST R 54620-2011/33464-2015
in	<i>callback</i>	Callback function to get the response of updateECallMsd.

### Returns

Status of updateECallMsd i.e. success or suitable error code.

#### 5.4.1.3.2.11 `virtual telux::common::Status telux::tel::ICallManager::requestECallHlapTimerStatus ( int phoneId, ECallHlapTimerStatusCallback callback ) [pure virtual]`

Request for status of eCall High Level Application Protocol(HLAP) timers that are maintained by the UE state machine. This does not retrieve status of timers maintained by the PSAP. The provided timers are as per EN 16062:2015 standard.

### Parameters

in	<i>phoneId</i>	Represents phone corresponding on which requestECallHlapTimerStatus operation is performed
in	<i>callback</i>	Callback function to get the response of requestECallHlapTimerStatus

### Returns

Status of requestECallHlapTimerStatus i.e. success or suitable error code.

### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.4.1.3.2.12 `virtual std::vector<std::shared_ptr<ICall> > telux::tel::ICallManager::getInProgressCalls ( ) [pure virtual]`

Get in-progress calls.

### Returns

List of active calls.

#### 5.4.1.3.2.13 `virtual telux::common::Status telux::tel::ICallManager::conference ( std::shared_ptr< ICall > call1, std::shared_ptr< ICall > call2, std::shared_ptr< telux::common::ICommand↔ ResponseCallback > callback = nullptr ) [pure virtual]`

Merge two calls in a conference.

in	<i>call1</i>	Call object to conference.
in	<i>call2</i>	Call object to conference.
in	<i>callback</i>	Optional callback pointer to get the result of conference function

**Returns**

Status of conference i.e. success or suitable error code.

**5.4.1.3.2.14** `virtual telux::common::Status telux::tel::ICallManager::swap ( std::shared_ptr< ICall > callToHold, std::shared_ptr< ICall > callToActivate, std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]`

Swap calls to make one active and put the another on hold.

**Parameters**

in	<i>callToHold</i>	Active call object to swap to hold state.
in	<i>callToActivate</i>	Hold call object to swap to active state.
in	<i>callback</i>	Optional callback pointer to get the result of swap function

**Returns**

Status of swap i.e. success or suitable error code.

**5.4.1.3.2.15** `virtual telux::common::Status telux::tel::ICallManager::registerListener ( std::shared_ptr< telux::tel::ICallListener > listener ) [pure virtual]`

Add a listener to listen for incoming call, call info change and eCall MSD transmission status change.

**Parameters**

in	<i>listener</i>	Pointer to <a href="#">ICallListener</a> object which receives event corresponding to phone
----	-----------------	---

**Returns**

Status of registerListener i.e. success or suitable error code.

**5.4.1.3.2.16** `virtual telux::common::Status telux::tel::ICallManager::removeListener ( std::shared_ptr< telux::tel::ICallListener > listener ) [pure virtual]`

Remove a previously added listener.

**Parameters**

in	<i>listener</i>	Listener to be removed.
----	-----------------	-------------------------

**Returns**

Status of removeListener i.e. success or suitable error code.

**5.4.1.4 class telux::tel::IMakeCallCallback**

Interface for Make Call callback object. Client needs to implement this interface to get single shot responses for commands like make call.

The methods in callback can be invoked from multiple different threads. The implementation should be thread safe.

**Public member functions**

- virtual void `makeCallResponse` (`telux::common::ErrorCode` error, `std::shared_ptr< ICall >` call=nullptr)
- virtual `~IMakeCallCallback` ()

**5.4.1.4.1 Constructors and Destructors**

**5.4.1.4.1.1** virtual `telux::tel::IMakeCallCallback::~~IMakeCallCallback` ( ) [`virtual`]

**5.4.1.4.2 Member Function Documentation**

**5.4.1.4.2.1** virtual void `telux::tel::IMakeCallCallback::makeCallResponse` ( `telux::common::ErrorCode` error, `std::shared_ptr< ICall >` call = nullptr ) [`virtual`]

This function is called with the response to makeCall API.

**Parameters**

out	<i>error</i>	ErrorCode
out	<i>call</i>	Pointer to Call object or nullptr in case of failure

**5.4.2 Enumeration Type Documentation****5.4.2.1 enum telux::tel::CallDirection** [`strong`]

Defines type of call like incoming, outgoing and none.

**Enumerator**

***INCOMING***  
***OUTGOING***  
***NONE***

### 5.4.2.2 enum telux::tel::CallState [strong]

Defines the states a call can be in

#### Enumerator

**CALL\_IDLE** idle call, default state of a newly created call object  
**CALL\_ACTIVE** active call  
**CALL\_ON\_HOLD** on hold call  
**CALL\_DIALING** out going call, in dialing state and not yet connected, MO Call only  
**CALL\_INCOMING** incoming call, not yet answered  
**CALL\_WAITING** waiting call  
**CALL\_ALERTING** alerting call, MO Call only  
**CALL\_ENDED** call ended / disconnected

### 5.4.2.3 enum telux::tel::CallEndCause [strong]

Reason for the recently terminated call (either normally ended or failed)

#### Enumerator

**UNOBTAINABLE\_NUMBER**  
**NO\_ROUTE\_TO\_DESTINATION**  
**CHANNEL\_UNACCEPTABLE**  
**OPERATOR\_DETERMINED\_BARRING**  
**NORMAL**  
**BUSY**  
**NO\_USER\_RESPONDING**  
**NO\_ANSWER\_FROM\_USER**  
**CALL\_REJECTED**  
**NUMBER\_CHANGED**  
**PREEMPTION**  
**DESTINATION\_OUT\_OF\_ORDER**  
**INVALID\_NUMBER\_FORMAT**  
**FACILITY\_REJECTED**  
**RESP\_TO\_STATUS\_ENQUIRY**  
**NORMAL\_UNSPECIFIED**  
**CONGESTION**  
**NETWORK\_OUT\_OF\_ORDER**  
**TEMPORARY\_FAILURE**  
**SWITCHING\_EQUIPMENT\_CONGESTION**  
**ACCESS\_INFORMATION\_DISCARDED**  
**REQUESTED\_CIRCUIT\_OR\_CHANNEL\_NOT\_AVAILABLE**  
**RESOURCES\_UNAVAILABLE\_OR\_UNSPECIFIED**  
**QOS\_UNAVAILABLE**  
**REQUESTED\_FACILITY\_NOT\_SUBSCRIBED**  
**INCOMING\_CALLS\_BARRED\_WITHIN\_CUG**  
**BEARER\_CAPABILITY\_NOT\_AUTHORIZED**  
**BEARER\_CAPABILITY\_UNAVAILABLE**  
**SERVICE\_OPTION\_NOT\_AVAILABLE**  
**BEARER\_SERVICE\_NOT\_IMPLEMENTED**

**ACM\_LIMIT\_EXCEEDED**  
**REQUESTED\_FACILITY\_NOT\_IMPLEMENTED**  
**ONLY\_DIGITAL\_INFORMATION\_BEARER\_AVAILABLE**  
**SERVICE\_OR\_OPTION\_NOT\_IMPLEMENTED**  
**INVALID\_TRANSACTION\_IDENTIFIER**  
**USER\_NOT\_MEMBER\_OF\_CUG**  
**INCOMPATIBLE\_DESTINATION**  
**INVALID\_TRANSIT\_NW\_SELECTION**  
**SEMANTICALLY\_INCORRECT\_MESSAGE**  
**INVALID\_MANDATORY\_INFORMATION**  
**MESSAGE\_TYPE\_NON\_IMPLEMENTED**  
**MESSAGE\_TYPE\_NOT\_COMPATIBLE\_WITH\_PROTOCOL\_STATE**  
**INFORMATION\_ELEMENT\_NON\_EXISTENT**  
**CONDITIONAL\_IE\_ERROR**  
**MESSAGE\_NOT\_COMPATIBLE\_WITH\_PROTOCOL\_STATE**  
**RECOVERY\_ON\_TIMER\_EXPIRED**  
**PROTOCOL\_ERROR\_UNSPECIFIED**  
**INTERWORKING\_UNSPECIFIED**  
**CALL\_BARRED**  
**FDN\_BLOCKED**  
**IMSI\_UNKNOWN\_IN\_VLR**  
**IMEI\_NOT\_ACCEPTED**  
**DIAL\_MODIFIED\_TO\_USSD**  
**DIAL\_MODIFIED\_TO\_SS**  
**DIAL\_MODIFIED\_TO\_DIAL**  
**CDMA\_LOCKED\_UNTIL\_POWER\_CYCLE**  
**CDMA\_DROP**  
**CDMA\_INTERCEPT**  
**CDMA\_REORDER**  
**CDMA\_SO\_REJECT**  
**CDMA\_RETRY\_ORDER**  
**CDMA\_ACCESS\_FAILURE**  
**CDMA\_PREEMPTED**  
**CDMA\_NOT\_EMERGENCY**  
**CDMA\_ACCESS\_BLOCKED**  
**ERROR\_UNSPECIFIED**

## 5.5 SMS

This section contains APIs related to Sending and Receiving SMS.

### 5.5.1 Data Structure Documentation

#### 5.5.1.1 struct telux::tel::MessageAttributes

Contains structure of message attributes like encoding type, number of segments, characters left in last segment.

##### Data fields

Type	Field	Description
<a href="#">SmsEncoding</a>	encoding	Data encoding type
int	numberOf↔ Segments	Number of segments
int	segmentSize	Max size of each segment
int	numberOf↔ CharsLeftIn↔ LastSegment	characters left in last segment

#### 5.5.1.2 struct telux::tel::MessagePartInfo

Structure containing information about the part of multi-part SMS such as concatenated message reference number, number of segments and segment number. During concatenation this information along with originating address helps in associating each part of the multi-part message to the corresponding multi-part message.

##### Data fields

Type	Field	Description
uint16_t	refNumber	Concatenated message reference number as per spec 3GPP TS 23.040 9.2.3.24.1. For each part of multipart message this message reference will be the same
uint8_t	numberOf↔ Segments	Number of segments
uint8_t	segment↔ Number	Segment Number

#### 5.5.1.3 class telux::tel::SmsMessage

Data structure represents an incoming SMS. This is applicable for single part message or part of the multipart message.

##### Public member functions

- [SmsMessage](#) (std::string text, std::string sender, std::string receiver, [SmsEncoding](#) encoding, std::string pdu, [PduBuffer](#) pduBuffer, std::shared\_ptr< [MessagePartInfo](#) > info)

- `const std::string & getText () const`
- `const std::string & getSender () const`
- `const std::string & getReceiver () const`
- `SmsEncoding getEncoding () const`
- `const std::string & getPdu () const`
- `PduBuffer getRawPdu () const`
- `std::shared_ptr< MessagePartInfo > getMessagePartInfo ()`
- `const std::string toString () const`

### 5.5.1.3.1 Constructors and Destructors

**5.5.1.3.1.1** `telux::tel::SmsMessage::SmsMessage ( std::string text, std::string sender, std::string receiver, SmsEncoding encoding, std::string pdu, PduBuffer pduBuffer, std::shared_ptr< MessagePartInfo > info )`

### 5.5.1.3.2 Member Function Documentation

**5.5.1.3.2.1** `const std::string& telux::tel::SmsMessage::getText ( ) const`

Get the message text for the single part message or part of the multipart message.

#### Returns

String containing SMS message.

**5.5.1.3.2.2** `const std::string& telux::tel::SmsMessage::getSender ( ) const`

Get the originating address (sender address).

#### Returns

String containing sender address.

**5.5.1.3.2.3** `const std::string& telux::tel::SmsMessage::getReceiver ( ) const`

Get the destination address (receiver address).

#### Returns

String containing receiver address



**5.5.1.3.2.4 SmsEncoding telux::tel::SmsMessage::getEncoding ( ) const**

Get encoding format used for the single part message or part of the multipart message.

**Returns**

SMS message encoding used.

**5.5.1.3.2.5 const std::string& telux::tel::SmsMessage::getPdu ( ) const**

Get the raw PDU for the single part message or part of the multipart message.

**Returns**

String containing raw PDU content.

**Deprecated** Use API [SmsMessage::getRawPdu](#)

**5.5.1.3.2.6 PduBuffer telux::tel::SmsMessage::getRawPdu ( ) const**

Get the raw PDU buffer for the single part message or part of the multipart message.

**Returns**

Buffer containing raw PDU content.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backward compatibility

**5.5.1.3.2.7 std::shared\_ptr<MessagePartInfo> telux::tel::SmsMessage::getMessagePartInfo ( )**

Applicable for multi-part SMS only. Get the information such as segment number, number of segments and concatenated reference number corresponding to the part of multi-part SMS.

**Returns**

If a message is single part SMS the method returns null otherwise returns message part information.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.5.1.3.2.8 const std::string telux::tel::SmsMessage::toString ( ) const

Get the text related informative representation of this object.

#### Returns

String containing informative string.

#### 5.5.1.4 class telux::tel::ISmsManager

SmsManager class is the primary interface to manage SMS operations such as send and receive an SMS text and raw encoded PDU(s). This class handles single part and multi-part messages.

#### Public member functions

- virtual [telux::common::ServiceStatus](#) getServiceStatus ()=0
- virtual [telux::common::Status](#) sendSms (const std::string &message, const std::string &receiver←Address, std::shared\_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr, std::shared\_ptr< [telux::common::ICommandResponseCallback](#) > deliveryCallback=nullptr)=0
- virtual [telux::common::Status](#) sendSms (std::string message, std::string receiverAddress, bool deliveryReportNeeded=true, [SmsResponseCb](#) sentCallback=nullptr, std::string smscAddr="")=0
- virtual [telux::common::Status](#) sendRawSms (const std::vector< [PduBuffer](#) > rawPdus, [SmsResponseCb](#) sentCallback=nullptr)=0
- virtual [telux::common::Status](#) requestSmscAddress (std::shared\_ptr< [ISmscAddressCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status](#) setSmscAddress (const std::string &smscAddress, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [MessageAttributes](#) calculateMessageAttributes (const std::string &message)=0
- virtual int getPhoneId ()=0
- virtual [telux::common::Status](#) registerListener (std::weak\_ptr< [ISmsListener](#) > listener)=0
- virtual [telux::common::Status](#) removeListener (std::weak\_ptr< [ISmsListener](#) > listener)=0
- virtual [~ISmsManager](#) ()

#### 5.5.1.4.1 Constructors and Destructors

5.5.1.4.1.1 virtual telux::tel::ISmsManager::~ISmsManager ( ) [virtual]

#### 5.5.1.4.2 Member Function Documentation

#### 5.5.1.4.2.1 virtual telux::common::ServiceStatus telux::tel::ISmsManager::getServiceStatus ( ) [pure virtual]

This status indicates whether the [ISmsManager](#) object is in a usable state.

#### Returns

[telux::common::ServiceStatus](#)

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.5.1.4.2.2 virtual telux::common::Status telux::tel::ISmsManager::sendSms ( const std::string & message, const std::string & receiverAddress, std::shared\_ptr< telux::common::ICommandResponseCallback > callback = nullptr, std::shared\_ptr< telux::common::ICommandResponseCallback > deliveryCallback = nullptr ) [pure virtual]

Send SMS to the destination address. When registered on IMS the SMS will be attempted over IMS. If sending SMS over IMS fails, an automatic retry would be attempted to send the message over CS. Only support UCS2 format, GSM 7 bit default alphabet and does not support National language shift tables.

#### Parameters

in	<i>message</i>	Message text to be sent
in	<i>receiverAddress</i>	Receiver or destination address
in	<i>sentCallback</i>	Optional callback pointer to get the response of send SMS request.
in	<i>deliveryCallback</i>	Optional callback pointer to get message delivery status

**Deprecated** Use API [ISmsManager::sendSms\(const std::string &message, const std::string &receiverAddress, bool deliveryReportNeeded = true, SmsResponseCb callback = nullptr, std::string smscAddr = ""\)](#)

#### Returns

Status of sendSms i.e. success or suitable error code.

#### 5.5.1.4.2.3 virtual telux::common::Status telux::tel::ISmsManager::sendSms ( std::string message, std::string receiverAddress, bool deliveryReportNeeded = true, SmsResponseCb sentCallback = nullptr, std::string smscAddr = "" ) [pure virtual]

Send single or multipart SMS to the destination address. When registered on IMS the SMS will be attempted over IMS. If sending SMS over IMS fails, an automatic retry would be attempted to send the message over CS. Only support UCS2 format, GSM 7 bit default alphabet and does not support National language shift tables.

in	<i>message</i>	Message text to be send.
in	<i>receiverAddress</i>	Receiver or destination address
in	<i>deliveryReportNeeded</i>	Delivery status received in the listener API <a href="#">telux::tel::ISmsListener</a> if <i>deliveryReportNeeded</i> is true. Provided recipient responds to SMSC before the validity period expires. If <i>deliveryReportNeeded</i> is false delivery report will not be received.
in	<i>sentCallback</i>	Optional callback pointer to get the sent response for single part or multi-part SMS.
in	<i>smcAddr</i>	SMS is sent to SMSC address. If SMSC address is empty then pre-configured SMSC address is used.

### Returns

Status of `sendSms` i.e. success or suitable error code.

### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.5.1.4.2.4 virtual `telux::common::Status telux::tel::ISmsManager::sendRawSms ( const std::vector< PduBuffer > rawPdus, SmsResponseCb sentCallback = nullptr ) [pure virtual]`

Send an SMS that is provided as a raw encoded PDU(s). When registered on IMS the SMS will be attempted over IMS. If sending SMS over IMS fails, an automatic retry would be attempted to send the message over CS. If the SMS is a multi-part message, the API expects multiple PDU to be passed to it.

### Parameters

in	<i>rawPdus</i>	Each element in the vector represents a part of a multipart message. For single part message the vector will have single element.
in	<i>sentCallback</i>	Optional callback to get the sent response for single part or multi-part SMS.

### Returns

Status of `sendRawSms` i.e. success or suitable error code.

### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.5.1.4.2.5 virtual telux::common::Status telux::tel::ISmsManager::requestSmscAddress ( std::shared\_ptr< ISmscAddressCallback > *callback* = nullptr ) [pure virtual]**

Request for Short Messaging Service Center (SMSC) Address. Purpose of SMSC is to store, forward, convert and deliver Short Message Service (SMS) messages.

#### Parameters

in	<i>callback</i>	Optional callback pointer to get the response of Smsc address request
----	-----------------	---

#### Returns

Status of getSmscAddress i.e. success or suitable error code.

**5.5.1.4.2.6 virtual telux::common::Status telux::tel::ISmsManager::setSmscAddress ( const std::string & *smcAddress*, telux::common::ResponseCallback *callback* = nullptr ) [pure virtual]**

Sets the Short Message Service Center(SMSC) address on the device.

This will change the SMSC address for all the SMS messages sent from any app.

#### Parameters

in	<i>smcAddress</i>	SMSC address
in	<i>callback</i>	Optional callback pointer to get the response of set SMSC address

#### Returns

Status of setSmscAddress i.e. success or suitable error code.

**5.5.1.4.2.7 virtual MessageAttributes telux::tel::ISmsManager::calculateMessageAttributes ( const std::string & *message* ) [pure virtual]**

Calculate message attributes for the given message.

#### Parameters

in	<i>message</i>	Message to send
----	----------------	-----------------

#### Returns

[MessageAttributes](#) structure containing encoding type, number of segments, max size of segment and characters left in last segment.

**5.5.1.4.2.8 virtual int telux::tel::ISmsManager::getPhoneId ( ) [pure virtual]**

Get associated phone id for this SMSManager.

**Returns**

PhoneId.

**5.5.1.4.2.9 virtual telux::common::Status telux::tel::ISmsManager::registerListener ( std::weak\_ptr< ISmsListener > *listener* ) [pure virtual]**

Register a listener for Sms events

**Parameters**

in	<i>listener</i>	Pointer to <a href="#">ISmsListener</a> object which receives event corresponding to SMS
----	-----------------	--

**Returns**

Status of registerListener i.e. success or suitable error code.

**5.5.1.4.2.10 virtual telux::common::Status telux::tel::ISmsManager::removeListener ( std::weak\_ptr< ISmsListener > *listener* ) [pure virtual]**

Remove a previously added listener.

**Parameters**

in	<i>listener</i>	Pointer to <a href="#">ISmsListener</a> object
----	-----------------	--

**Returns**

Status of removeListener i.e. success or suitable error code.

**5.5.1.5 class telux::tel::ISmsListener**

A listener class receives notification for the incoming message(s) and delivery report for sent message(s).

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

**Public member functions**

- virtual void [onIncomingSms](#) (int phoneId, std::shared\_ptr< [SmsMessage](#) > message)
- virtual void [onIncomingSms](#) (int phoneId, std::shared\_ptr< std::vector< [SmsMessage](#) >> messages)
- virtual void [onDeliveryReport](#) (int phoneId, int msgRef, std::string receiverAddress, [telux::common::ErrorCode](#) error)

- virtual [~ISmsListener](#) ()

### 5.5.1.5.1 Constructors and Destructors

5.5.1.5.1.1 virtual `telux::tel::ISmsListener::~ISmsListener ( )` [virtual]

### 5.5.1.5.2 Member Function Documentation

5.5.1.5.2.1 virtual void `telux::tel::ISmsListener::onIncomingSms ( int phoneId, std::shared_ptr< SmsMessage > message )` [virtual]

This function will be invoked when a single part message is received or when a part of a multi-part message is received.

#### Parameters

in	<i>phoneId</i>	Unique identifier per SIM slot. Phone on which the message is received.
in	<a href="#">SmsMessage</a>	Pointer to <a href="#">SmsMessage</a> object

5.5.1.5.2.2 virtual void `telux::tel::ISmsListener::onIncomingSms ( int phoneId, std::shared_ptr< std::vector< SmsMessage >> messages )` [virtual]

This function will be invoked when either a single part message is received, or when all the parts of a multipart message have been received.

#### Parameters

in	<i>phoneId</i>	Unique identifier per SIM slot. Phone on which the message is received.
in	<i>messages</i>	Pointer to list of <a href="#">SmsMessage</a> received corresponding to single part or all parts of multipart message.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.5.1.5.2.3 virtual void `telux::tel::ISmsListener::onDeliveryReport ( int phoneId, int msgRef, std::string receiverAddress, telux::common::ErrorCode error )` [virtual]

This function will be invoked when either a delivery report for a single part message is received or when the delivery report for part of a multi-part message is received. In order to determine delivery of all parts of the multi-part message, the client application shall compare message reference received in the delivery indications with message references received in `telux::tel::SmsResponseCb`.

in	<i>phoneId</i>	Unique identifier per SIM slot. Phone on which the message is received.
in	<i>msgRef</i>	Message reference number (as per spec 3GPP TS 23.040 9.2.2.3) for a single part message or part of multipart message.
in	<i>receiverAddress</i>	Receiver or destination address
in	<i>error</i>	<a href="#">telux::common::ErrorCode</a>

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.5.1.6 class telux::tel::ISmscAddressCallback**

Interface for SMS callback object. Client needs to implement this interface to get single shot responses for send SMS.

The methods in callback can be invoked from multiple different threads. The implementation should be thread safe.

**Public member functions**

- virtual void [smscAddressResponse](#) (const std::string &address, [telux::common::ErrorCode](#) error)=0

**5.5.1.6.1 Member Function Documentation****5.5.1.6.1.1 virtual void telux::tel::ISmscAddressCallback::smscAddressResponse ( const std::string & address, telux::common::ErrorCode error ) [pure virtual]**

This function is called with the response to the Smsc address request.

**Parameters**

in	<i>address</i>	Smsc address
in	<i>error</i>	ErrorCode

**5.5.2 Enumeration Type Documentation****5.5.2.1 enum telux::tel::SmsEncoding [strong]**

Specifies the encoding of the SMS message.

**Enumerator**

- GSM7** GSM 7-bit default alphabet encoding
- GSM8** GSM 8-bit data encoding
- UCS2** UCS-2 encoding
- UNKNOWN** Unknown encoding



## 5.6 SIM Card Services

This section contains APIs related to Card Services.

### 5.6.1 Data Structure Documentation

#### 5.6.1.1 class telux::tel::ICardApp

Represents a single card application.

##### Public member functions

- virtual [AppType](#) `getAppType ()=0`
- virtual [AppState](#) `getAppState ()=0`
- virtual `std::string` `getAppId ()=0`
- virtual [telux::common::Status](#) `changeCardPassword (CardLockType lockType, std::string oldPwd, std::string newPwd, PinOperationResponseCb callback)=0`
- virtual [telux::common::Status](#) `unlockCardByPuk (CardLockType lockType, std::string puk, std::string newPin, PinOperationResponseCb callback)=0`
- virtual [telux::common::Status](#) `unlockCardByPin (CardLockType lockType, std::string pin, PinOperationResponseCb callback)=0`
- virtual [telux::common::Status](#) `queryPin1LockState (QueryPin1LockResponseCb callback)=0`
- virtual [telux::common::Status](#) `queryFdnLockState (QueryFdnLockResponseCb callback)=0`
- virtual [telux::common::Status](#) `setCardLock (CardLockType lockType, std::string password, bool isEnabled, PinOperationResponseCb callback)=0`
- virtual `~ICardApp ()`

#### 5.6.1.1.1 Constructors and Destructors

5.6.1.1.1.1 virtual `telux::tel::ICardApp::~~ICardApp ( ) [virtual]`

#### 5.6.1.1.2 Member Function Documentation

5.6.1.1.2.1 virtual `AppType` `telux::tel::ICardApp::getAppType ( ) [pure virtual]`

Get Application type like SIM, USIM, RUIM, CSIM or ISIM.

##### Returns

[AppType](#).

**5.6.1.1.2.2 virtual AppState telux::tel::ICardApp::getAppState ( ) [pure virtual]**

Get Application state like PIN1, PUK required and others.

**Returns**

[AppState](#).

**5.6.1.1.2.3 virtual std::string telux::tel::ICardApp::getAppld ( ) [pure virtual]**

Get application identifier.

**Returns**

Application Id.

**5.6.1.1.2.4 virtual telux::common::Status telux::tel::ICardApp::changeCardPassword ( CardLockType lockType, std::string oldPwd, std::string newPwd, PinOperationResponseCb callback ) [pure virtual]**

Change the password used in PIN1/PIN2 lock.

**Parameters**

in	<i>lockType</i>	CardLockType. Applicable lock types are PIN1 and PIN2.
in	<i>oldPwd</i>	Old password
in	<i>newPwd</i>	New password
in	<i>callback</i>	Callback function to get the response of change pin password.

**5.6.1.1.2.5 virtual telux::common::Status telux::tel::ICardApp::unlockCardByPuk ( CardLockType lockType, std::string puk, std::string newPin, PinOperationResponseCb callback ) [pure virtual]**

Unlock the Sim card for an app by entering PUK and new pin.

**Parameters**

in	<i>lockType</i>	CardLockType. Applicable lock types are PUK1 and PUK2
in	<i>puk</i>	PUK1/PUK2
in	<i>pin</i>	New PIN1/PIN2
in	<i>callback</i>	Callback function to get the response of unlock card lock.

**5.6.1.1.2.6 virtual telux::common::Status telux::tel::ICardApp::unlockCardByPin ( CardLockType lockType, std::string pin, PinOperationResponseCb callback ) [pure virtual]**

Unlock the Sim card for an app by entering PIN.

in	<i>lockType</i>	CardLockType. Applicable lock types are PIN1 and PIN2.
in	<i>pin</i>	New PIN1/PIN2
in	<i>callback</i>	Callback function to get the response of unlock card lock.

#### 5.6.1.1.2.7 virtual telux::common::Status telux::tel::ICardApp::queryPin1LockState ( QueryPin1Lock↔ ResponseCb *callback* ) [pure virtual]

Query Pin1 lock state.

##### Parameters

in	<i>callback</i>	Callback function to get the response of query pin1 lock state.
----	-----------------	---

#### 5.6.1.1.2.8 virtual telux::common::Status telux::tel::ICardApp::queryFdnLockState ( QueryFdnLock↔ ResponseCb *callback* ) [pure virtual]

Query FDN lock state.

##### Parameters

in	<i>callback</i>	Callback function to get the response of query fdn lock state.
----	-----------------	--

#### 5.6.1.1.2.9 virtual telux::common::Status telux::tel::ICardApp::setCardLock ( CardLockType *lockType*, std::string *password*, bool *isEnabled*, PinOperationResponseCb *callback* ) [pure virtual]

Enable or disable FDN or Pin1 lock.

##### Parameters

in	<i>lockType</i>	CardLockType. Applicable lock type such as PIN1 and FDN
in	<i>password</i>	Password of PIN1 and FDN
in	<i>isEnabled</i>	If true then enable else disable.
in	<i>callback</i>	Callback function to get the response of set card lock.

### 5.6.1.2 struct telux::tel::lccResult

The APDU response with status for transmit APDU operation.

#### Public member functions

- const std::string [toString](#) () const

#### Data Fields

- int [sw1](#)
- int [sw2](#)

- `std::string` [payload](#)
- `std::vector< int >` [data](#)

### 5.6.1.2.1 Member Function Documentation

#### 5.6.1.2.1.1 `const std::string telux::tel::lccResult::toString ( ) const`

### 5.6.1.2.2 Field Documentation

#### 5.6.1.2.2.1 `int telux::tel::lccResult::sw1`

Status word 1 for command processing status

#### 5.6.1.2.2.2 `int telux::tel::lccResult::sw2`

Status word 2 for command processing qualifier

#### 5.6.1.2.2.3 `std::string telux::tel::lccResult::payload`

response as a hex string

#### 5.6.1.2.2.4 `std::vector<int> telux::tel::lccResult::data`

vector of raw data received as part of response to the card services request

### 5.6.1.3 `class telux::tel::ICardManager`

[ICardManager](#) provide APIs for slot count, retrieve slot ids, get card state and get card.

#### Public member functions

- virtual `bool` [isSubsystemReady](#) ()=0
- virtual `std::future< bool >` [onSubsystemReady](#) ()=0
- virtual `telux::common::Status` [getSlotCount](#) (int &count)=0
- virtual `telux::common::Status` [getSlotIds](#) (std::vector< int > &slotIds)=0
- virtual `std::shared_ptr< ICard >` [getCard](#) (int slotId=DEFAULT\_SLOT\_ID, `telux::common::Status *status=nullptr`)=0
- virtual `telux::common::Status` [registerListener](#) (std::shared\_ptr< [ICardListener](#) > listener)=0
- virtual `telux::common::Status` [removeListener](#) (std::shared\_ptr< [ICardListener](#) > listener)=0
- virtual `~ICardManager` ()

### 5.6.1.3.1 Constructors and Destructors

5.6.1.3.1.1 `virtual telux::tel::ICardManager::~ICardManager ( ) [virtual]`

### 5.6.1.3.2 Member Function Documentation

5.6.1.3.2.1 `virtual bool telux::tel::ICardManager::isSubsystemReady ( ) [pure virtual]`

Checks the status of telephony subsystems and returns the result.

#### Returns

If true then CardManager is ready for service.

5.6.1.3.2.2 `virtual std::future<bool> telux::tel::ICardManager::onSubsystemReady ( ) [pure virtual]`

Wait for telephony subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when card manager is ready.

5.6.1.3.2.3 `virtual telux::common::Status telux::tel::ICardManager::getSlotCount ( int & count ) [pure virtual]`

Get SIM slot count.

#### Parameters

<code>out</code>	<code>count</code>	SIM slot count.
------------------	--------------------	-----------------

#### Returns

Status of getSlotCount i.e. success or suitable status code.

5.6.1.3.2.4 `virtual telux::common::Status telux::tel::ICardManager::getSlotIds ( std::vector< int > & slotIds ) [pure virtual]`

Get list of SIM slots.

#### Parameters

<code>out</code>	<code>slotIds</code>	List of SIM slot ids.
------------------	----------------------	-----------------------

#### Returns

Status of getSlotIds i.e. success or suitable status code.

**5.6.1.3.2.5** `virtual std::shared_ptr<ICard> telux::tel::ICardManager::getCard ( int slotId = DEFAULT_SLOT_ID, telux::common::Status * status = nullptr ) [pure virtual]`

Get the Card corresponding to SIM slot.

#### Parameters

in	<i>slotId</i>	Slot id corresponding to the card.
out	<i>status</i>	Status of getCard i.e. success or suitable status code.

#### Returns

Pointer to [ICard](#) object.

**5.6.1.3.2.6** `virtual telux::common::Status telux::tel::ICardManager::registerListener ( std::shared_ptr<ICardListener > listener ) [pure virtual]`

Register a listener for card events.

#### Parameters

in	<i>listener</i>	Pointer to <a href="#">ICardListener</a> object that processes the notification.
----	-----------------	--

#### Returns

Status of registerListener i.e. success or suitable status code.

**5.6.1.3.2.7** `virtual telux::common::Status telux::tel::ICardManager::removeListener ( std::shared_ptr<ICardListener > listener ) [pure virtual]`

Remove a previously added listener.

#### Parameters

in	<i>listener</i>	Pointer to <a href="#">ICardListener</a> object that needs to be removed.
----	-----------------	---

#### Returns

Status of removeListener i.e. success or suitable status code.

### 5.6.1.4 class telux::tel::ICard

[ICard](#) represents currently inserted UICC or eUICC.

**Public member functions**

- virtual `telux::common::Status getState (CardState &cardState)=0`
- virtual `std::vector< std::shared_ptr< ICardApp > > getApplications (telux::common::Status *status=nullptr)=0`
- virtual `telux::common::Status openLogicalChannel (std::string applicationId, std::shared_ptr< ICardChannelCallback > callback=nullptr)=0`
- virtual `telux::common::Status closeLogicalChannel (int channelId, std::shared_ptr< telux::common::ICommandResponseCallback > callback=nullptr)=0`
- virtual `telux::common::Status transmitApuLogicalChannel (int channel, uint8_t cla, uint8_t instruction, uint8_t p1, uint8_t p2, uint8_t p3, std::vector< uint8_t > data, std::shared_ptr< ICardCommandCallback > callback=nullptr)=0`
- virtual `telux::common::Status transmitApuBasicChannel (uint8_t cla, uint8_t instruction, uint8_t p1, uint8_t p2, uint8_t p3, std::vector< uint8_t > data, std::shared_ptr< ICardCommandCallback > callback=nullptr)=0`
- virtual `telux::common::Status exchangeSimIO (uint16_t fileId, uint8_t command, uint8_t p1, uint8_t p2, uint8_t p3, std::string filePath, std::vector< uint8_t > data, std::string pin2, std::string aid, std::shared_ptr< ICardCommandCallback > callback=nullptr)=0`
- virtual `int getSlotId ()=0`
- virtual `telux::common::Status requestEid (EidResponseCallback callback)=0`

**5.6.1.4.1 Member Function Documentation****5.6.1.4.1.1 virtual telux::common::Status telux::tel::ICard::getState ( CardState & cardState ) [pure virtual]**

Get the card state for the slot id.

**Parameters**

out	<i>cardState</i>	<code>CardState</code> - state of the card.
-----	------------------	---

**Returns**

Status of `getCardState` i.e. success or suitable status code.

**5.6.1.4.1.2 virtual std::vector<std::shared\_ptr<ICardApp> > telux::tel::ICard::getApplications ( telux::common::Status \* status = nullptr ) [pure virtual]**

Get card applications.

**Parameters**

out	<i>status</i>	Status of <code>getApplications</code> i.e. success or suitable status code.
-----	---------------	--

**Returns**

List of card applications.

**5.6.1.4.1.3** `virtual telux::common::Status telux::tel::ICard::openLogicalChannel ( std::string applicationId, std::shared_ptr< ICardChannelCallback > callback = nullptr ) [pure virtual]`

Open a logical channel to the SIM.

**Parameters**

in	<i>applicationId</i>	Application Id.
in	<i>callback</i>	Optional callback pointer to get the response of open logical channel request.

**Returns**

Status of openLogicalChannel i.e. success or suitable status code.

**5.6.1.4.1.4** `virtual telux::common::Status telux::tel::ICard::closeLogicalChannel ( int channelId, std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]`

Close a previously opened logical channel to the SIM.

**Parameters**

in	<i>channelId</i>	The channel ID to be closed.
in	<i>callback</i>	Optional callback pointer to get the response of close logical channel request.

**Returns**

Status of closeLogicalChannel i.e. success or suitable status code.

**5.6.1.4.1.5** `virtual telux::common::Status telux::tel::ICard::transmitApduLogicalChannel ( int channel, uint8_t cla, uint8_t instruction, uint8_t p1, uint8_t p2, uint8_t p3, std::vector< uint8_t > data, std::shared_ptr< ICardCommandCallback > callback = nullptr ) [pure virtual]`

Transmit an APDU to the ICC card over a logical channel.

**Parameters**

in	<i>channel</i>	Channel Id of the channel to use for communication. Has to be greater than zero.
in	<i>cla</i>	Class of the APDU command.
in	<i>instruction</i>	Instruction of the APDU command.



in	<i>p1</i>	Instruction Parameter 1 value of the APDU command.
in	<i>p2</i>	Instruction Parameter 2 value of the APDU command.
in	<i>p3</i>	Number of bytes present in the data field of the APDU command. If <i>p3</i> is negative, a 4 byte APDU is sent to the SIM.
in	<i>data</i>	Data to be sent with the APDU.
in	<i>callback</i>	Optional callback pointer to get the response of transmit APDU request.

### Returns

Status of transmitApuLogicalChannel i.e. success or suitable status code.

**5.6.1.4.1.6** virtual telux::common::Status telux::tel::ICard::transmitApuBasicChannel ( uint8\_t *cla*, uint8\_t *instruction*, uint8\_t *p1*, uint8\_t *p2*, uint8\_t *p3*, std::vector< uint8\_t > *data*, std::shared\_ptr< ICardCommandCallback > *callback = nullptr* ) [pure virtual]

Exchange APDUs with the SIM on a basic channel.

### Parameters

in	<i>cla</i>	Class of the APDU command.
in	<i>instruction</i>	Instruction of the APDU command.
in	<i>p1</i>	Instruction Param1 value of the APDU command.
in	<i>p2</i>	Instruction Param1 value of the APDU command.
in	<i>p3</i>	Number of bytes present in the data field of the APDU command. If <i>p3</i> is negative, a 4 byte APDU is sent to the SIM.
in	<i>data</i>	Data to be sent with the APDU.
in	<i>callback</i>	Optional callback pointer to get the response of transmit APDU request.

### Returns

Status of transmitApuBasicChannel i.e. success or suitable status code.

**5.6.1.4.1.7** virtual telux::common::Status telux::tel::ICard::exchangeSimIO ( uint16\_t *fileId*, uint8\_t *command*, uint8\_t *p1*, uint8\_t *p2*, uint8\_t *p3*, std::string *filePath*, std::vector< uint8\_t > *data*, std::string *pin2*, std::string *aid*, std::shared\_ptr< ICardCommandCallback > *callback = nullptr* ) [pure virtual]

Performs SIM IO operation, This is similar to the TS 27.007 "restricted SIM" operation where it assumes all of the EF selection will be done by the callee

### Parameters

in	<i>fileId</i>	Elementary File Identifier
in	<i>command</i>	APDU Command for SIM IO operation
in	<i>p1</i>	Instruction Param1 value of the APDU command
in	<i>p2</i>	Instruction Param2 value of the APDU command

in	<i>p3</i>	Number of bytes present in the data field of APDU command. If p3 is negative, a 4 byte APDU is sent to the SIM.
in	<i>filePath</i>	Path of the file
in	<i>data</i>	Data to be sent with the APDU, send empty or null string in case no data
in	<i>pin2</i>	Pin value of the SIM. Invalid attempt of PIN2 value will lock the SIM. send empty or null string in case of no Pin2 value
in	<i>aid</i>	Application identifier, send empty or null string in case of no aid
in	<i>callback</i>	Optional callback pointer to get the response of SIM IO request

**Returns**

- Status of exchangeSimIO i.e. success or suitable status code

**5.6.1.4.1.8 virtual int telux::tel::ICard::getSlotId ( ) [pure virtual]**

Get associated slot id for [ICard](#)

**Returns**

SlotId

**5.6.1.4.1.9 virtual telux::common::Status telux::tel::ICard::requestEid ( EidResponseCallback *callback* ) [pure virtual]**

Request eUICC identifier (EID) of eUICC card.

**Parameters**

in	<i>callback</i>	Callback function to get the result of request EID.
----	-----------------	---

**Returns**

Status of request EID i.e. success or suitable error code.

**Dependencies Card should be eUICC capable****5.6.1.5 class telux::tel::ICardChannelCallback**

Interface for Card callback object. Client needs to implement this interface to get single shot responses for commands like open logical channel and close logical channel.

The methods in callback can be invoked from multiple different threads. The implementation should be thread safe.

**Public member functions**

- virtual void `onChannelResponse` (int channel, `IccResult` result, `telux::common::ErrorCode` error)=0

**5.6.1.5.1 Member Function Documentation**

**5.6.1.5.1.1** virtual void `telux::tel::IccChannelCallback::onChannelResponse` ( int *channel*, `IccResult` *result*, `telux::common::ErrorCode` *error* ) [pure virtual]

This function is called with the response to the open logical channel operation.

**Parameters**

in	<i>channel</i>	Channel Id for the logical channel.
in	<i>result</i>	<code>IccResult</code> of open logical channel.
in	<i>error</i>	<code>ErrorCode</code> of the request.

**5.6.1.6 class telux::tel::IccCommandCallback****Public member functions**

- virtual void `onResponse` (`IccResult` result, `telux::common::ErrorCode` error)=0

**5.6.1.6.1 Member Function Documentation**

**5.6.1.6.1.1** virtual void `telux::tel::IccCommandCallback::onResponse` ( `IccResult` *result*, `telux::common::ErrorCode` *error* ) [pure virtual]

This function is called when SIM Card transmit APDU over Logical, Basic Channel and Exchange Sim IO.

**Parameters**

in	<i>result</i>	<code>IccResult</code> of transmit APDU command
in	<i>error</i>	<code>ErrorCode</code> of the request, Possible error codes are <ul style="list-style-type: none"> <li>• <code>SUCCESS</code></li> <li>• <code>INTERNAL</code></li> <li>• <code>NO_MEMORY</code></li> <li>• <code>INVALID_ARG</code></li> <li>• <code>MISSING_ARG</code></li> </ul>

**5.6.1.7 class telux::tel::IccListener**

Interface for SIM Card Listener object. Client needs to implement this interface to get access to card services notifications on card state change.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

**Public member functions**

- virtual void [onCardInfoChanged](#) (int slotId)
- virtual [~ICardListener](#) ()

**5.6.1.7.1 Constructors and Destructors**

**5.6.1.7.1.1** virtual `telux::tel::ICardListener::~~ICardListener ( )` [virtual]

**5.6.1.7.2 Member Function Documentation**

**5.6.1.7.2.1** virtual void `telux::tel::ICardListener::onCardInfoChanged ( int slotId )` [virtual]

This function is called when info of card gets updated.

**Parameters**

in	<i>slotId</i>	Slot identifier.
----	---------------	------------------

**5.6.1.8 struct telux::tel::CardReaderStatus**

Structure contains identity of card reader status

**Data fields**

Type	Field	Description
int	id	Card Reader ID
bool	isRemovable	Card reader is removable
bool	isPresent	Card reader is present
bool	isID1size	Card reader present is ID-1 size
bool	isCardPresent	Card is present in reader
bool	isCard↔ PoweredOn	Card in reader is powered

**5.6.1.9 class telux::tel::ISapCardManager**

[ISapCardManager](#) provide APIs for SAP related operations.

**Public member functions**

- virtual bool [isReady](#) ()=0
- virtual std::future< bool > [onReady](#) ()=0
- virtual `telux::common::Status` [getState](#) (SapState &sapState)=0
- virtual `telux::common::Status` [requestSapState](#) (SapStateResponseCallback callback)=0
- virtual `telux::common::Status` [openConnection](#) (SapCondition sapCondition=SapCondition::SAP\_CONDITION\_BLOCK\_VOICE\_OR\_DATA, std::shared\_ptr<

- `telux::common::ICommandResponseCallback > callback=nullptr)=0`
- virtual `telux::common::Status closeConnection (std::shared_ptr< telux::common::ICommandResponseCallback > callback=nullptr)=0`
- virtual `telux::common::Status requestAtr (std::shared_ptr< IAtrResponseCallback > callback=nullptr)=0`
- virtual `telux::common::Status transmitApdu (uint8_t cla, uint8_t instruction, uint8_t p1, uint8_t p2, uint8_t lc, std::vector< uint8_t > data, uint8_t le=0, std::shared_ptr< ISapCardCommandCallback > callback=nullptr)=0`
- virtual `telux::common::Status requestSimPowerOff (std::shared_ptr< telux::common::ICommandResponseCallback > callback=nullptr)=0`
- virtual `telux::common::Status requestSimPowerOn (std::shared_ptr< telux::common::ICommandResponseCallback > callback=nullptr)=0`
- virtual `telux::common::Status requestSimReset (std::shared_ptr< telux::common::ICommandResponseCallback > callback=nullptr)=0`
- virtual `telux::common::Status requestCardReaderStatus (std::shared_ptr< ICardReaderCallback > callback=nullptr)=0`
- virtual `int getSlotId ()=0`
- virtual `telux::common::Status registerListener (std::shared_ptr< ISapCardListener > listener)=0`
- virtual `telux::common::Status removeListener (std::shared_ptr< ISapCardListener > listener)=0`
- virtual `~ISapCardManager ()`

### 5.6.1.9.1 Constructors and Destructors

5.6.1.9.1.1 virtual `telux::tel::ISapCardManager::~~ISapCardManager ( ) [virtual]`

### 5.6.1.9.2 Member Function Documentation

5.6.1.9.2.1 virtual `bool telux::tel::ISapCardManager::isReady ( ) [pure virtual]`

Checks the status of SIM access profile(SAP) subsystem and returns the result.

#### Returns

If true then SapCardManager is ready for service.

5.6.1.9.2.2 virtual `std::future<bool> telux::tel::ISapCardManager::onReady ( ) [pure virtual]`

Wait for IM access profile(SAP) subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when card manager is ready.

### 5.6.1.9.2.3 virtual telux::common::Status telux::tel::ISapCardManager::getState ( SapState & *sapState* ) [pure virtual]

Get SIM access profile (SAP) client connection state.

#### Parameters

out	<i>sapState</i>	SapState of the SIM Card
-----	-----------------	--------------------------

#### Returns

Status of getState i.e. success or suitable status code.

**Deprecated** Use [requestSapState\(\)](#) API below to get SAP state

### 5.6.1.9.2.4 virtual telux::common::Status telux::tel::ISapCardManager::requestSapState ( SapStateResponseCallback *callback* ) [pure virtual]

Get SIM access profile(SAP) client connection state.

#### Parameters

out	<i>callback</i>	Callback function pointer to get the response of requestSapState.
-----	-----------------	---

#### Returns

Status of requestSapState i.e. success or suitable status code.

### 5.6.1.9.2.5 virtual telux::common::Status telux::tel::ISapCardManager::openConnection ( SapCondition *sapCondition* = SapCondition::SAP\_CONDITION\_BLOCK\_VOICE\_OR\_DATA, std::shared\_ptr< telux::common::ICommandResponseCallback > *callback* = nullptr ) [pure virtual]

Establishes SIM access profile (SAP) client connection with SIM Card.

#### Parameters

in	<i>sapCondition</i>	Condition to enable sap connection.
in	<i>callback</i>	Optional callback to get the response of open sap connection request or possible error codes i.e. <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• INTERNAL</li> <li>• NO_MEMORY</li> <li>• INVALID_ARG</li> <li>• MISSING_ARG</li> </ul>

**Returns**

Status of openConnection i.e. success or suitable status code.

**5.6.1.9.2.6** `virtual telux::common::Status telux::tel::ISapCardManager::closeConnection ( std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]`

Releases a SAP connection to SIM Card.

**Parameters**

in	<i>callback</i>	Optional callback to get the response of close sap connection request or possible error codes i.e. <ul style="list-style-type: none"> <li>• <a href="#">SUCCESS</a></li> <li>• INTERNAL</li> <li>• NO_MEMORY</li> <li>• INVALID_ARG</li> <li>• MISSING_ARG</li> </ul>
----	-----------------	---

**Returns**

Status of closeConnection i.e. success or suitable status code

**5.6.1.9.2.7** `virtual telux::common::Status telux::tel::ISapCardManager::requestAtr ( std::shared_ptr< IAtrResponseCallback > callback = nullptr ) [pure virtual]`

Request for SAP Answer To Reset command.

**Parameters**

in	<i>callback</i>	Optional callback to get the response of requestAtr.
----	-----------------	--

**Returns**

Status of requestAtr i.e. success or suitable status code.

**5.6.1.9.2.8** `virtual telux::common::Status telux::tel::ISapCardManager::transmitApdu ( uint8_t cla, uint8_t instruction, uint8_t p1, uint8_t p2, uint8_t lc, std::vector< uint8_t > data, uint8_t le = 0, std::shared_ptr< ISapCardCommandCallback > callback = nullptr ) [pure virtual]`

Send the Apdu on SAP mode.

**Parameters**

in	<i>cla</i>	Class of the APDU command.
in	<i>instruction</i>	Instruction of the APDU command.
in	<i>p1</i>	Instruction Parameter 1 value of the APDU command.

in	<i>p2</i>	Instruction Parameter 1 value of the APDU command.
in	<i>lc</i>	Number of bytes present in the data field of the APDU command. If <i>lc</i> is negative, a 4 byte APDU is sent to the SIM.
in	<i>data</i>	List of data to be sent with the APDU.
in	<i>le</i>	Maximum number of bytes expected in the data field of the response to the command.
in	<i>callback</i>	Optional callback to send APDU in SAP mode.

**Returns**

Status of transmitApdu i.e. success or suitable status code.

**5.6.1.9.2.9** `virtual telux::common::Status telux::tel::ISapCardManager::requestSimPowerOff ( std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]`

Send the SAP SIM power off request.

**Parameters**

in	<i>callback</i>	Optional callback to get the response for SIM power off.
----	-----------------	--

**Returns**

Status of requestSimPowerOff i.e. success or suitable status code.

**5.6.1.9.2.10** `virtual telux::common::Status telux::tel::ISapCardManager::requestSimPowerOn ( std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]`

Send the SAP SIM power on request.

**Parameters**

in	<i>callback</i>	Optional callback to get the response for SIM power on.
----	-----------------	---

**Returns**

Status of requestSimPowerOn i.e. success or suitable status code.

**5.6.1.9.2.11** `virtual telux::common::Status telux::tel::ISapCardManager::requestSimReset ( std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]`

Send the SAP SIM reset request.



in	<i>callback</i>	Optional callback to get the response for SIM reset
----	-----------------	---

**Returns**

Status of requestSimReset i.e. success or suitable status code.

**5.6.1.9.2.12** `virtual telux::common::Status telux::tel::ISapCardManager::requestCardReaderStatus ( std::shared_ptr< ICardReaderCallback > callback = nullptr ) [pure virtual]`

Send the SAP Card Reader Status request command.

**Parameters**

in	<i>callback</i>	Optional callback to get the response for card reader status
----	-----------------	--

**Returns**

Status of requestCardReaderStatus i.e. success or suitable status code.

**5.6.1.9.2.13** `virtual int telux::tel::ISapCardManager::getSlotId ( ) [pure virtual]`

Get associated slot id for the SapCardManager.

**Returns**

SlotId

**5.6.1.9.2.14** `virtual telux::common::Status telux::tel::ISapCardManager::registerListener ( std::shared_ptr< ISapCardListener > listener ) [pure virtual]`

Register a listener for SAP events.

**Parameters**

in	<i>listener</i>	Pointer to <a href="#">ISapCardListener</a> object that processes the notification.
----	-----------------	---

**Returns**

Status of registerListener i.e. success or suitable status code.

**5.6.1.9.2.15** `virtual telux::common::Status telux::tel::ISapCardManager::removeListener ( std::shared_ptr< ISapCardListener > listener ) [pure virtual]`

Remove a previously added listener.

in	<i>listener</i>	Pointer to <a href="#">ISapCardListener</a> object that needs to be removed.
----	-----------------	--

## Returns

Status of removeListener i.e. success or suitable status code.

### 5.6.1.10 class telux::tel::IAtrResponseCallback

#### Public member functions

- virtual void [atrResponse](#) (std::vector< int > responseAtr, [telux::common::ErrorCode](#) error)=0

#### 5.6.1.10.1 Member Function Documentation

**5.6.1.10.1.1** virtual void [telux::tel::IAtrResponseCallback::atrResponse](#) ( std::vector< int > *responseAtr*, [telux::common::ErrorCode](#) *error* ) [pure virtual]

This function is called in response to requestAtr() request.

#### Parameters

in	<i>responseAtr</i>	response ATR values
in	<i>error</i>	ErrorCode of the request possible error codes are <ul style="list-style-type: none"> <li>• <a href="#">SUCCESS</a></li> <li>• INTERNAL</li> <li>• NO_MEMORY</li> <li>• INVALID_ARG</li> <li>• MISSING_ARG</li> </ul>

### 5.6.1.11 class telux::tel::ISapCardCommandCallback

#### Public member functions

- virtual void [onResponse](#) ([IccResult](#) result, [telux::common::ErrorCode](#) error)=0

#### 5.6.1.11.1 Member Function Documentation

**5.6.1.11.1.1** virtual void [telux::tel::ISapCardCommandCallback::onResponse](#) ( [IccResult](#) *result*, [telux::common::ErrorCode](#) *error* ) [pure virtual]

This function is called when SIM Card transmit APDU on SAP mode.

**Parameters**

in	<i>result</i>	<a href="#">IccResult</a> of transmit APDU command
in	<i>error</i>	ErrorCode of the request, possible error codes are <ul style="list-style-type: none"> <li>• <a href="#">SUCCESS</a></li> <li>• INTERNAL</li> <li>• NO_MEMORY</li> <li>• INVALID_ARG</li> <li>• MISSING_ARG</li> </ul>

**5.6.1.12 class telux::tel::ICardReaderCallback****Public member functions**

- virtual void [cardReaderResponse](#) ([CardReaderStatus](#) cardReaderStatus, [telux::common::ErrorCode](#) error)=0

**5.6.1.12.1 Member Function Documentation****5.6.1.12.1.1 virtual void telux::tel::ICardReaderCallback::cardReaderResponse ( [CardReaderStatus](#) *cardReaderStatus*, [telux::common::ErrorCode](#) *error* ) [pure virtual]**

This function is called in response to requestCardReaderStatus() method.

**Parameters**

in	<i>cardReaderStatus</i>	Structure contains the identity of the card reader
in	<i>error</i>	ErrorCode of the request

**5.6.1.13 class telux::tel::ISapCardListener**

Interface for SAP Listener object. Client needs to implement this interface to get access to SAP service notifications like service status change.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

**Public member functions**

- virtual [~ISapCardListener](#) ()

**5.6.1.13.1 Constructors and Destructors****5.6.1.13.1.1 virtual telux::tel::ISapCardListener::~~ISapCardListener ( ) [virtual]**

## 5.6.2 Enumeration Type Documentation

### 5.6.2.1 enum telux::tel::CardState [strong]

Defines all state of Card like absent, present etc

#### Enumerator

**CARDSTATE\_UNKNOWN** Unknown card state  
**CARDSTATE\_ABSENT** Card is absent  
**CARDSTATE\_PRESENT** Card is present  
**CARDSTATE\_ERROR** Card is having error, either card is removed and not readable  
**CARDSTATE\_RESTRICTED** Card is present but not usable due to carrier restrictions.

### 5.6.2.2 enum telux::tel::CardError [strong]

Defines the reasons for error in CardState

#### Enumerator

**UNKNOWN** Unknown error  
**POWER\_DOWN** Power down  
**POLL\_ERROR** Poll error  
**NO\_ATR\_RECEIVED** No ATR received  
**VOLT\_MISMATCH** Volt mismatch  
**PARITY\_ERROR** Parity error  
**POSSIBLY\_REMOVED** Unknown, possibly removed  
**TECHNICAL\_PROBLEMS** Card returned technical problems  
**NULL\_BYTES** Card returned NULL bytes  
**SAP\_CONNECTED** Terminal in SAP mode  
**CMD\_TIMEOUT** Command timeout error

### 5.6.2.3 enum telux::tel::CardLockType [strong]

Defines all types of card locks which uses in PIN management APIs

#### Enumerator

**PIN1** Lock type is PIN1  
**PIN2** Lock type is PIN2  
**PUK1** Lock type is Pin Unblocking Key1  
**PUK2** Lock type is Pin Unblocking Key2  
**FDN** Lock type is Fixed Dialing Number

### 5.6.2.4 enum telux::tel::AppType

Defines all type of UICC application such as SIM, RUIM, USIM, CSIM and ISIM.

#### Enumerator

**APPTYPE\_UNKNOWN** Unknown application type  
**APPTYPE\_SIM** UICC application type is SIM

**APPTYPE\_USIM** UICC application type is USIM  
**APPTYPE\_RUIM** UICC application type is RSIM  
**APPTYPE\_CSIM** UICC application type is CSIM  
**APPTYPE\_ISIM** UICC application type is ISIM

### 5.6.2.5 enum telux::tel::AppState

Defines all application states.

#### Enumerator

**APPSTATE\_UNKNOWN** Unknown application state  
**APPSTATE\_DETECTED** application state detected  
**APPSTATE\_PIN** If PIN1 or UPin is required  
**APPSTATE\_PUK** If PUK1 or Puk for UPin is required  
**APPSTATE\_SUBSCRIPTION\_PERSO** PersoSubstate should be look at when application state is assigned to this value  
**APPSTATE\_READY** application State is ready

### 5.6.2.6 enum telux::tel::SapState [strong]

Defines all SIM access profile (SAP) connection states.

#### Enumerator

**SAP\_STATE\_NOT\_ENABLED** SAP connection not enabled  
**SAP\_STATE\_CONNECTING** SAP State is connecting  
**SAP\_STATE\_CONNECTED\_SUCCESSFULLY** SAP connection is successful  
**SAP\_STATE\_CONNECTION\_ERROR** SAP connection error  
**SAP\_STATE\_DISCONNECTING** SAP state is disconnecting  
**SAP\_STATE\_DISCONNECTED\_SUCCESSFULLY** SAP state disconnection is successful

### 5.6.2.7 enum telux::tel::SapCondition [strong]

Indicates type of connection required, default behavior is to block a SAP connection when a voice or data call is active.

#### Enumerator

**SAP\_CONDITION\_BLOCK\_VOICE\_OR\_DATA** Block a SAP connection when a voice or data call is active (Default)  
**SAP\_CONDITION\_BLOCK\_DATA** Block a SAP connection when a data call is active  
**SAP\_CONDITION\_BLOCK\_VOICE** Block a SAP connection when a voice call is active  
**SAP\_CONDITION\_BLOCK\_NONE** Allow Sap connection in all cases

## 5.7 Location Services

This section contains APIs related to Location Services.

### 5.7.1 Data Structure Documentation

#### 5.7.1.1 class telux::loc::IDgnssStatusListener

Listener class for getting RTCM injection event notification information.

##### Public member functions

- virtual void [onDgnssStatusUpdate](#) ([DgnssStatus](#) status)
- virtual [~IDgnssStatusListener](#) ()

##### 5.7.1.1.1 Constructors and Destructors

###### 5.7.1.1.1.1 virtual telux::loc::IDgnssStatusListener::~~IDgnssStatusListener ( ) [virtual]

Destructor of IRTCMStatusListener

##### 5.7.1.1.2 Member Function Documentation

###### 5.7.1.1.2.1 virtual void telux::loc::IDgnssStatusListener::onDgnssStatusUpdate ( [DgnssStatus status](#) ) [virtual]

This function is called asynchronously to report RTCM injection status

##### Parameters

in	<i>status</i>	- the status enum.
----	---------------	--------------------

#### 5.7.1.2 class telux::loc::IDgnssManager

IRtcManager provides interface to inject RTCM data into modem, register event listener reported by cdfw(correction data framework).

##### Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status registerListener](#) (std::weak\_ptr< [IDgnssStatusListener](#) > listener)=0
- virtual [telux::common::Status deRegisterListener](#) (void)=0
- virtual [telux::common::Status createSource](#) ([DgnssDataFormat](#) dataFormat)=0
- virtual [telux::common::Status releaseSource](#) (void)=0

- virtual [telux::common::Status injectCorrectionData](#) (const uint8\_t \*buffer, uint32\_t bufferSize)=0
- virtual [~IDgnssManager](#) ()

### 5.7.1.2.1 Constructors and Destructors

#### 5.7.1.2.1.1 virtual [telux::loc::IDgnssManager::~~IDgnssManager](#) ( ) [virtual]

Destructor of IRtcmManager

### 5.7.1.2.2 Member Function Documentation

#### 5.7.1.2.2.1 virtual bool [telux::loc::IDgnssManager::isSubsystemReady](#) ( ) [pure virtual]

Checks the status of location subsystems and returns the result.

#### Returns

True if location subsystem is ready for service otherwise false.

#### 5.7.1.2.2.2 virtual [std::future<bool> telux::loc::IDgnssManager::onSubsystemReady](#) ( ) [pure virtual]

Wait for location subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when location subsystem is ready.

#### 5.7.1.2.2.3 virtual [telux::common::Status telux::loc::IDgnssManager::registerListener](#) ( [std::weak\\_ptr<IDgnssStatusListener>](#) *listener* ) [pure virtual]

Register a listener for Dgnss injection status update.

#### Parameters

in	<i>listener</i>	- Pointer of <a href="#">IDgnssStatusListener</a> object that processes the notification.
----	-----------------	---

#### Returns

Status of registerListener i.e success or suitable status code.

**5.7.1.2.2.4 virtual telux::common::Status telux::loc::IDgnssManager::deRegisterListener ( void )**  
**[pure virtual]**

deRegister a listener for Dgnss injection status update.

#### Returns

Status of registerListener i.e success or suitable status code.

**5.7.1.2.2.5 virtual telux::common::Status telux::loc::IDgnssManager::createSource ( DgnssDataFormat dataFormat )**  
**[pure virtual]**

Create a Dgnss injection source. Only one source is permitted at any given time. If a new source is to be used, user must call [releaseSource\(\)](#) to release previous source before calling this function.

#### Parameters

in	<i>format</i>	Dgnss injection data format.
----	---------------	------------------------------

#### Returns

Success of suitable status code

**5.7.1.2.2.6 virtual telux::common::Status telux::loc::IDgnssManager::releaseSource ( void )**  
**[pure virtual]**

Release current Dgnss injection source (previously created by [createSource\(\)](#) call) This function is to be called if it's determined that current injection data is not suitable anymore, and a new source will be created and used as injection source.

#### Parameters

<i>none</i>		
-------------	--	--

#### Returns

none

**5.7.1.2.2.7 virtual telux::common::Status telux::loc::IDgnssManager::injectCorrectionData ( const uint8\_t\* buffer, uint32\_t bufferSize )**  
**[pure virtual]**

Inject correction data This function is to be called when a source has been created, either through a explicit call to [createSource\(\)](#), or after DgnssManager object was instantiated through the factory method(The factory method create a default source for DgnssManager object).

#### Parameters

in	<i>buffer</i>	buffer contains the data to be injected.
in	<i>bufferSize</i>	size of the buffer.



## Returns

success or suitable status code.

### 5.7.1.3 class telux::loc::ILocationConfigurator

**ILocationConfigurator** allows general engine configurations (example: TUNC, PACE etc), configuration of specific engines like SPE (example: minSVElevation, minGPSWeek etc) or DRE, deletion of warm and cold aiding data, NMEA configuration and support for XTRA feature. **ILocationConfigurator** APIs strictly adheres to the principle of single client per process.

#### Public Types

- using **GetMinGpsWeekCallback** = std::function< void(uint16\_t minGpsWeek, telux::common::ErrorCode error)>
- using **GetMinSVElevationCallback** = std::function< void(uint8\_t minSVElevation, telux::common::ErrorCode error)>
- using **GetRobustLocationCallback** = std::function< void(const telux::loc::RobustLocationConfiguration rLConfig, telux::common::ErrorCode error)>
- using **GetXtraStatusCallback** = std::function< void(const telux::loc::XtraStatus xtraStatus, telux::common::ErrorCode error)>

#### Public member functions

- virtual bool **isSubsystemReady** ()=0
- virtual std::future< bool > **onSubsystemReady** ()=0
- virtual telux::common::Status **configureCTunc** (bool enable, telux::common::ResponseCallback callback=nullptr, float timeUncertainty=DEFAULT\_TUNC\_THRESHOLD, uint32\_t energyBudget=DEFAULT\_TUNC\_ENERGY\_THRESHOLD)=0
- virtual telux::common::Status **configurePACE** (bool enable, telux::common::ResponseCallback callback=nullptr)=0
- virtual telux::common::Status **deleteAllAidingData** (telux::common::ResponseCallback callback=nullptr)=0
- virtual telux::common::Status **configureLeverArm** (const LeverArmConfigInfo &info, telux::common::ResponseCallback callback=nullptr)=0
- virtual telux::common::Status **configureConstellations** (const SvBlackList &list, telux::common::ResponseCallback callback=nullptr, bool resetToDefault=false)=0
- virtual telux::common::Status **configureRobustLocation** (bool enable, bool enableForE911=false, telux::common::ResponseCallback callback=nullptr)=0
- virtual telux::common::Status **requestRobustLocation** (GetRobustLocationCallback cb)=0
- virtual telux::common::Status **configureMinGpsWeek** (uint16\_t minGpsWeek, telux::common::ResponseCallback callback=nullptr)=0
- virtual telux::common::Status **requestMinGpsWeek** (GetMinGpsWeekCallback cb)=0

- virtual `telux::common::Status configureMinSVElevation` (`uint8_t minSVElevation`, `telux::common::ResponseCallback callback=nullptr`)=0
- virtual `telux::common::Status requestMinSVElevation` (`GetMinSVElevationCallback cb`)=0
- virtual `telux::common::Status deleteAidingData` (`AidingData aidingDataMask`, `telux::common::ResponseCallback callback=nullptr`)=0
- virtual `telux::common::Status configureDR` (`const DREngineConfiguration &config`, `telux::common::ResponseCallback callback=nullptr`)=0
- virtual `telux::common::Status configureNmeaTypes` (`const NmeaSentenceConfig nmeaType`, `telux::common::ResponseCallback callback=nullptr`)=0
- virtual `telux::common::Status configureNmea` (`const NmeaConfig configParams`, `telux::common::ResponseCallback callback=nullptr`)=0
- virtual `telux::common::Status provideConsentForTerrestrialPositioning` (`bool userConsent`, `telux::common::ResponseCallback callback=nullptr`)=0
- virtual `telux::common::Status registerLocationInjector` (`std::weak_ptr< ILocationInjectionListener > listener`)=0
- virtual `telux::common::Status deregisterLocationInjector` (`std::weak_ptr< ILocationInjectionListener > listener`)=0
- virtual `telux::common::Status injectLocationData` (`const ExternalLocationInfo &location`)=0
- virtual `telux::common::Status configureEngineIntegrityRisk` (`const EngineType engineType`, `uint32_t integrityRisk`, `telux::common::ResponseCallback callback=nullptr`)=0
- virtual `telux::common::Status configureXtraParams` (`bool enable`, `const XtraConfig configParams`, `telux::common::ResponseCallback callback=nullptr`)=0
- virtual `telux::common::Status requestXtraStatus` (`GetXtraStatusCallback callback`)=0
- virtual `telux::common::Status registerListener` (`LocConfigIndications indicationList`, `std::weak_ptr< ILocationConfigListener > listener`)=0
- virtual `telux::common::Status deRegisterListener` (`LocConfigIndications indicationList`, `std::weak_ptr< ILocationConfigListener > listener`)=0
- virtual `~ILocationConfigurator` ()

### 5.7.1.3.1 Member Typedef Documentation

5.7.1.3.1.1 `using telux::loc::ILocationConfigurator::GetMinGpsWeekCallback = std::function<void(uint16_t minGpsWeek, telux::common::ErrorCode error)>`

This function is called with the response to requestMinGpsWeek API.

#### Parameters

in	<i>minGpsWeek</i>	- minimum gps week.
in	<i>error</i>	- Return code which indicates whether the operation succeeded or not.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.7.1.3.1.2** using `telux::loc::ILocationConfigurator::GetMinSVElevationCallback = std::function<void(uint8_t minSVElevation, telux::common::ErrorCode error)>`

This function is called with the response to requestMinSVElevation API.

**Parameters**

in	<i>minSVElevation</i>	- minimum SV Elevation angle in units of degree.
in	<i>error</i>	- Return code which indicates whether the operation succeeded or not.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.7.1.3.1.3** using `telux::loc::ILocationConfigurator::GetRobustLocationCallback = std::function<void(const telux::loc::RobustLocationConfiguration rLConfig, telux::common::ErrorCode error)>`

This function is called with the response to requestRobustLocation API.

**Parameters**

in	<i>rLConfig</i>	- robust location settings information.
in	<i>error</i>	- Return code which indicates whether the operation succeeded or not.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.7.1.3.1.4** using `telux::loc::ILocationConfigurator::GetXtraStatusCallback = std::function<void(const telux::loc::XtraStatus xtraStatus, telux::common::ErrorCode error)>`

This function is called with the response to requestXtraStatus API.

**Parameters**

in	<i>xtraStatus</i>	- Information pertaining to Xtra assistance data.
in	<i>error</i>	- Return code which indicates whether the operation succeeded or not.

### 5.7.1.3.2 Constructors and Destructors

#### 5.7.1.3.2.1 virtual telux::loc::ILocationConfigurator::~~ILocationConfigurator ( ) [virtual]

Destructor of [ILocationConfigurator](#)

### 5.7.1.3.3 Member Function Documentation

#### 5.7.1.3.3.1 virtual bool telux::loc::ILocationConfigurator::isSubsystemReady ( ) [pure virtual]

Checks the status of location configuration subsystems and returns the result.

#### Returns

True if location configuration subsystem is ready for service otherwise false.

#### 5.7.1.3.3.2 virtual std::future<bool> telux::loc::ILocationConfigurator::onSubsystemReady ( ) [pure virtual]

Wait for location configuration subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when location configuration subsystem is ready.

#### 5.7.1.3.3.3 virtual telux::common::Status telux::loc::ILocationConfigurator::configureC↔ Tunc ( bool *enable*, telux::common::ResponseCallback *callback* = *nullptr*, float *timeUncertainty* = DEFAULT\_TUNC\_THRESHOLD, uint32\_t *energyBudget* = DEFAULT\_TUNC\_ENERGY\_THRESHOLD ) [pure virtual]

This API enables or disables the constrained time uncertainty(C-TUNC) feature. When the vehicle is turned off this API helps to put constraint on the time uncertainty. For multiple invocations of this API, client should wait for the command to finish, e.g.: via ResponseCallback recieved before issuing a second configureCTunc command.

#### Parameters

in	<i>enable</i>	- true for enable C-TUNC feature and false for disable C-TUNC feature.
in	<i>callback</i>	- Optional callback to get the response of enablement/disablement of C-TUNC.
in	<i>timeUncertainty</i>	- specifies the time uncertainty threshold that gps engine needs to maintain, in unit of milli-seconds.
in	<i>energyBudget</i>	- specifies the power budget that the GPS engine is allowed to spend to maintain the time uncertainty, in the unit of 100 micro watt second. If the power exceeds the energyBudget then this API is disabled. This is a cumulative energy budget.

## Returns

Status of configureCTunc i.e. success or suitable status code.

### 5.7.1.3.3.4 virtual telux::common::Status telux::loc::ILocationConfigurator::configurePACE ( bool *enable*, telux::common::ResponseCallback *callback* = nullptr ) [pure virtual]

This API enables or disables position assisted clock estimator feature. For multiple invocations of this API, client should wait for the command to finish, e.g.: via ResponseCallback received before issuing a second configurePACE command.

#### Parameters

in	<i>enable</i>	- to enable/disable position assisted clock estimator feature.
in	<i>callback</i>	- Optional callback to get the response of enablement/disablement of PACE.

### 5.7.1.3.3.5 virtual telux::common::Status telux::loc::ILocationConfigurator::deleteAllAidingData ( telux::common::ResponseCallback *callback* = nullptr ) [pure virtual]

This API deletes all form of aiding data from all position engines. This API deletes all assistance data used by GPS engine and force engine to do a cold start for next session. Invoking this API will trigger cold start of all position engines on the device. This will cause significant delay for the position engines to produce next fix and may have other performance impact.

#### Parameters

in	<i>callback</i>	- Optional callback to get the response of delete aiding data.
----	-----------------	--

### 5.7.1.3.3.6 virtual telux::common::Status telux::loc::ILocationConfigurator::configureLeverArm ( const LeverArmConfigInfo & *info*, telux::common::ResponseCallback *callback* = nullptr ) [pure virtual]

This API sets the lever arm parameters for the vehicle. For multiple invocations of this API client should wait for the command to finish, e.g.: via ResponseCallback received before issuing a second configureLeverArm command.

#### Parameters

in	<i>info</i>	- lever arm configuration info regarding below three types of lever arm info: a: GNSS Antenna w.r.t the origin at the IMU (inertial measurement unit) for DR engine b: GNSS Antenna w.r.t the origin at the IMU (inertial measurement unit) for VEPP engine c: VRP (Vehicle Reference Point) w.r.t the origin (at the GNSS Antenna). Vehicle manufacturers prefer the position output to be tied to a specific point in the vehicle rather than where the antenna is placed (midpoint of the rear axle is typical).
in	<i>callback</i>	- Optional callback to get the response of configure lever arm.

**5.7.1.3.3.7 virtual telux::common::Status telux::loc::ILocationConfigurator::configureConstellations ( const SvBlackList & list, telux::common::ResponseCallback callback = nullptr, bool resetToDefault = false ) [pure virtual]**

This API blacklists some constellations or subset of SVs from the constellation from being used by the GNSS engine on modem. For multiple invocations of this API, client should wait for the command to finish, e.g.: via ResponseCallback received before issuing a second configureConstellations command. This API call is not incremental and the new settings will completely overwrite the previous call. Supported constellations for this API are GLONASS, QZSS, BEIDOU, GALILEO and SBAS. For other constellations NOTSUPPORTED status will be returned. When resetToDefault is false then the list is expected to contain the constellations or SVs that should be blacklisted. An empty list could be specified to allow all constellations/SVs (i.e. none will be blacklisted) in determining the fix. When resetToDefault is set to true, the device will revert to the default list of SV/ constellations to be blacklisted.

#### Parameters

in	<i>list</i>	- specify the set of constellations and SVs that should not be used by the GNSS engine on modem. Constellations and SVs not specified in blacklistedSvList could get used by the GNSS engine on modem.
in	<i>callback</i>	- Optional callback to get the response of configure constellations.
in	<i>resetToDefault</i>	- when set to true, the device will revert to the default list of SV/constellation to be blacklisted. When set to false, list will be inspected to determine what should be blacklisted.

**5.7.1.3.3.8 virtual telux::common::Status telux::loc::ILocationConfigurator::configureRobustLocation ( bool enable, bool enableForE911 = false, telux::common::ResponseCallback callback = nullptr ) [pure virtual]**

This API enables/disables robust location feature and enables/disables robust location while device is on E911. This API focuses on detection and reporting GNSS spoofing in position, time and navigation data. When this API is enabled it reports confidence of the GNSS spoofing by the getConformityIndex() API defined in the [ILocationInfoEx](#) class.

#### Parameters

in	<i>enable</i>	- true to enable robust location and false to disable robust location.
in	<i>enableForE911</i>	- true to enable robust location when the device is on E911 session and false to disable on E911 session. This parameter is only valid if robust location is enabled.
in	<i>callback</i>	- Optional callback to get the response of configure robust location.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.7.1.3.3.9 virtual telux::common::Status telux::loc::ILocationConfigurator::requestRobustLocation ( GetRobustLocationCallback *cb* ) [pure virtual]

This API retrieves the robust location settings used by the GNSS engine.

#### Parameters

in	<i>cb</i>	- callback to retrieve robust location information.
----	-----------	---

#### Returns

Status of requestRobustLocation i.e. success or suitable status code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.7.1.3.3.10 virtual telux::common::Status telux::loc::ILocationConfigurator::configureMinGpsWeek ( uint16\_t *minGpsWeek*, telux::common::ResponseCallback *callback = nullptr* ) [pure virtual]

This API configures the minimum GPS week used by the modem GNSS engine. Client should wait for the command to finish, e.g.: via ResponseCallback received before issuing a second configureMinGpsWeek command.

#### Parameters

in	<i>minGpsWeek</i>	- minimum GPS week to be used by modem GNSS engine.
in	<i>callback</i>	- Optional callback to get the response of configure minimum GPS week.

#### Returns

Status of configureMinGpsWeek i.e. success or suitable status code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.7.1.3.3.11 virtual telux::common::Status telux::loc::ILocationConfigurator::requestMinGpsWeek ( GetMinGpsWeekCallback *cb* ) [pure virtual]

This API retrieves the minimum GPS week configuration used by the modem GNSS engine.

**Parameters**

in	<i>cb</i>	- callback to retrieve the minimum gps week.
----	-----------	--

**Returns**

Status of requestMinGpsWeek i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.7.1.3.3.12** `virtual telux::common::Status telux::loc::ILocationConfigurator::configureMinSVElevation (uint8_t minSVElevation, telux::common::ResponseCallback callback = nullptr) [pure virtual]`

This API configures the minimum SV elevation angle setting used by the GNSS standard position engine. Configuring minimum SV elevation setting will not cause position engine to stop tracking low elevation SVs. This elevation mask is used to filter out the SVs that are used in determining the position. SVs with an elevation below this setting will be excluded from position determination. So configuring this to a large angle will filter out most of the SVs and will have adverse affects on the performance of the position engine.

This setting does not impact the SV information and SV measurement reports retrieved from APIs such as `IGnssSvInfo::getSVInfoList`, `ILocationListener::onGnssMeasurementsInfo`.

Client should wait for the command to finish, e.g.: via `ResponseCallback` received, before issuing a second `configureMinElevation` command. If this API is called while the GNSS Position Engine is in the middle of a session, `ResponseCallback` will still be invoked shortly after to indicate the setting has been received by the SPE engine. However the engine can take some time to apply this setting if it is in middle of a session (as long as 255 seconds in some implementations). It is advised to use this API with caution and only for very limited usage scenario, e.g.: for performance test, certification process and for one-time device configuration.

**Parameters**

in	<i>minSVElevation</i>	- minimum SV elevation to be used by GNSS standard position engine (SPE). Valid range is [0, 90] in unit of degree.
in	<i>callback</i>	- Optional callback to get the response of configure minimum SV Elevation angle.

**Returns**

Status of `configureMinSVElevation` i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.



### 5.7.1.3.3.13 `virtual telux::common::Status telux::loc::ILocationConfigurator::requestMinSVElevation ( GetMinSVElevationCallback cb ) [pure virtual]`

This API retrieves the minimum SV Elevation configuration used by the modem GNSS SPE engine. If this API is invoked right after the `configureMinSVElevation`, the returned setting may not match the one specified in `configureMinSVElevation`, as the setting received via `configureMinSVElevation` might not have been applied yet as it takes time to apply the setting if the GNSS SPE engine has an on-going session.

#### Parameters

in	<i>cb</i>	- callback to retrieve the minimum SV elevation.
----	-----------	--

#### Returns

Status of `requestMinSVElevation` i.e. success or suitable status code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.7.1.3.3.14 `virtual telux::common::Status telux::loc::ILocationConfigurator::deleteAidingData ( AidingData aidingDataMask, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

This API deletes specified aiding data from all position engines on the device. For example, removing ephemeris data may trigger GNSS engine to do a warm start.

#### Parameters

in	<i>aidingDataMask</i>	- specify the set of aiding data to be deleted from all position engines.
in	<i>callback</i>	- Optional callback to get the response of delete aiding data.

#### Returns

Status of `deleteAidingData` i.e. success or suitable status code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.7.1.3.3.15 `virtual telux::common::Status telux::loc::ILocationConfigurator::configureDR ( const DREngineConfiguration & config, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

This API configures various parameters for dead reckoning position engine. Clients should wait for the command to finish e.g.: via `ResponseCallback` to be received before issuing a second `configureDREngine`←

Params command. Behavior is not defined if client issues a second request of `configureDREngineParams` without waiting for the completion of the previous `configureDREngineParams` request.

### Parameters

in	<i>config</i>	- specify dead reckoning engine configuration.
in	<i>callback</i>	- Optional callback to get the response of <code>configureDREngineParams</code> .

### Returns

Status of `configureDREngineParams` i.e. success or suitable status code.

### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.7.1.3.3.16** `virtual telux::common::Status telux::loc::ILocationConfigurator::configureNmeaTypes ( const NmeaSentenceConfig nmeaType, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

This API is used to configure the NMEA sentence types that clients will receive via [ILocationManager::startDetailedReports](#) or [ILocationManager::startDetailedEngineReports](#). Without prior invocation to this API, all NMEA sentences supported in the system will get generated and delivered to all the clients that register to receive NMEA sentences. The NMEA sentence type configuration is common across all clients and updating it will affect all clients. This API call is not incremental and the new NMEA sentence types will completely overwrite the previous call to this API.

### Parameters

in	<i>nmeaType</i>	- specify the set of NMEA sentences
in	<i>callback</i>	- Optional callback to get the response of <code>configureNmeaTypes</code> .

### Returns

Status of `configureNmeaTypes` i.e. success or suitable status code.

### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.7.1.3.3.17** `virtual telux::common::Status telux::loc::ILocationConfigurator::configureNmea ( const NmeaConfig configParams, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

This API is used to configure the NMEA sentences that the clients will receive via [ILocationManager::startDetailedReports](#) or [ILocationManager::startDetailedEngineReports](#). Without prior

invocation to this API, all NMEA sentences supported in the system will get generated and delivered to all the clients that register to receive NMEA sentences. The NMEA sentence type configuration is common across all clients and updating it will affect all clients.

Please note that for the NMEA datum type request to be successful, the nmea provider configuration in the GPS configuration file should be set to application processor.

This API call is not incremental and the new NMEA configuration will completely overwrite the previous call to this API.

#### Parameters

in	<i>configParams</i>	- Configuration Parameters for Nmea on the device.
in	<i>callback</i>	- Optional callback to get the response of configureNmea.

#### Returns

Status of configureNmea i.e. success or suitable status code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.7.1.3.3.18 virtual telux::common::Status telux::loc::ILocationConfigurator::provideConsentForTerrestrialPositioning ( bool *userConsent*, telux::common::ResponseCallback *callback* = *nullptr* ) [pure virtual]**

Clients can request Terrestrial Positioning using [ILocationManager::getTerrestrialPosition](#). Terrestrial Positioning requires sending device data to the cloud to get the position. This functionality requires user consent. This API needs to be invoked to provide the user consent.

The consent will remain effective across power cycles, until this API is called with a different value.

#### Parameters

in	<i>userConsent</i>	- true indicates user consents to sending device data to cloud, false indicates user does not consent.
in	<i>callback</i>	- Optional callback to get the response of provideConsentForTerrestrialPositioning.

#### Returns

Status of provideConsentForTerrestrialPositioning i.e. success or suitable status code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.7.1.3.3.19 virtual telux::common::Status telux::loc::ILocationConfigurator::registerLocationInjector ( std::weak\_ptr< ILocationInjectionListener > *listener* ) [pure virtual]**

This API registers an [ILocationInjectionListener](#) listener and will receive notifications regarding when to start or stop the location report injection using the [injectLocationData](#) API.

**Parameters**

in	<i>listener</i>	- Pointer of <a href="#">ILocationInjectionListener</a> object.
----	-----------------	---

**Returns**

Status of registerLocationInjector i.e success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.7.1.3.3.20 virtual telux::common::Status telux::loc::ILocationConfigurator::deregisterLocationInjector ( std::weak\_ptr< ILocationInjectionListener > *listener* ) [pure virtual]**

This API removes a previously registered listener and will also stop receiving notifications related to location data injection for that particular listener.

**Parameters**

in	<i>listener</i>	- Pointer of <a href="#">ILocationInjectionListener</a> object.
----	-----------------	---

**Returns**

Status of deregisterLocationInjector i.e success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.7.1.3.3.21 virtual telux::common::Status telux::loc::ILocationConfigurator::injectLocationData ( const ExternalLocationInfo & *location* ) [pure virtual]**

Inject best location data

This API is used to inject the [ExternalLocationInfo](#) data into the GNSS engine.

This API is intended to be invoked after [ILocationInjectionListener::onStartInjection](#) API call and is not expected to be invoked after [ILocationInjectionListener::onStopInjection](#) API call. If this API is used without meeting the above mentioned pre-requisites, the GNSS engine might produce low quality GNSS position.

in	<i>location</i>	<a href="#">ExternalLocationInfo</a> data to be injected.
----	-----------------	---

**Returns**

, Status of injectLocationData i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.7.1.3.3.22** `virtual telux::common::Status telux::loc::LocationConfigurator::configureEngine( IntegrityRisk ( const EngineType engineType, uint32_t integrityRisk, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

This API is used to instruct the specified engine to use the provided integrity risk level for protection level calculation in position report. This API can be called when a position session is in progress. Prior to calling this API for a particular engine, the engine shall not calculate the protection levels and shall not include the protection levels in its position report. The implementation might not support protection levels across all engines. For engines that don't support it, ResponseCallback will get invoked with ErrorCode::NOT\_SUPPORTED.

**Parameters**

in	<i>engineType</i>	- the engine that is instructed to use the specified integrity risk level for protection level calculation.
in	<i>integrityRisk</i>	- the integrity risk level used for calculating protection level. The integrity risk is defined as a probability per epoch, in unit of 2.5e-10. The valid range for actual integrity is [2.5e-10, 1-2.5e-10]), this corresponds to range of [1,4e9-1] of this parameter.
in	<i>callback</i>	- Optional callback to get the response of configureEngineIntegrityRisk.

**Returns**

Status of configureEngineIntegrityRisk i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.7.1.3.3.23** `virtual telux::common::Status telux::loc::LocationConfigurator::configureXtraParams ( bool enable, const XtraConfig configParams, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

This API is used to enable/disable the XTRA (Predicted GNSS Satellite Orbit Data) feature on device. If XTRA feature is to be enabled, this API is also used to configure the various XTRA settings in device.

Clients need to note the below-

1. Wait for the ongoing request to finish prior to the next invocation else the behavior is undefined.
2. The API is non-incremental i.e, the second call will overwrite the first call. Also the configured XTRA params will be persistent.

#### Parameters

in	<i>enable</i>	- Enable XTRA Feature on the device. False would disable both the XTRA Assistance Data and NTP Time Download.
in	<i>configParams</i>	- Configuration Parameters for XTRA on the device.
in	<i>callback</i>	- Optional callback stating the response errorcode.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.7.1.3.3.24 virtual telux::common::Status telux::loc::ILocationConfigurator::requestXtraStatus ( GetXtraStatusCallback *callback* ) [pure virtual]

This API is used to query xtra feature setting and xtra assistance data status used by the GNSS standard position engine (SPE). If XTRA\_DATA\_STATUS\_UNKNOWN is returned but XTRA feature is enabled, the client shall wait a few seconds before calling this API again.

#### Parameters

in	<i>callback</i>	- Callback to get the Xtra data status information.
----	-----------------	---

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.7.1.3.3.25 virtual telux::common::Status telux::loc::ILocationConfigurator::registerListener ( LocConfigIndications *indicationList*, std::weak\_ptr< ILocationConfigListener > *listener* ) [pure virtual]

This API is used to register a configuration listener for getting specific indications/updates.

#### Parameters

in	<i>indicationList</i>	- List of indications client wants to register under <a href="#">telux::loc::LocConfigIndicationsType</a> .
in	<i>listener</i>	- Pointer of <a href="#">ILocationConfigListener</a> object.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.7.1.3.3.26 virtual telux::common::Status telux::loc::ILocationConfigurator::deRegisterListener ( LocConfigIndications *indicationList*, std::weak\_ptr< ILocationConfigListener > *listener* ) [pure virtual]

This API is used to deregister a configuration listener from specific indications/updates.

**Parameters**

in	<i>indicationList</i>	- List of indications client wants to deregister from under <a href="#">telux::loc::LocConfigIndicationsType</a> .
in	<i>listener</i>	- Pointer of <a href="#">ILocationConfigListener</a> object.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.7.1.4 struct telux::loc::GnssKinematicsData

Specifies kinematics related information.

**Data fields**

Type	Field	Description
<a href="#">Kinematic↔ DataValidity</a>	bodyFrame↔ DataMask	Contains Body frame LocPosDataMask bits.
float	longAccel	Forward Acceleration in body frame (m/s <sup>2</sup> )
float	latAccel	Sideward Acceleration in body frame (m/s <sup>2</sup> )
float	vertAccel	Vertical Acceleration in body frame (m/s <sup>2</sup> )
float	yawRate	Heading Rate (Radians/second)
float	pitch	Body pitch (Radians)
float	longAccelUnc	Uncertainty of Forward Acceleration in body frame
float	latAccelUnc	Uncertainty of Side-ward Acceleration in body frame
float	vertAccelUnc	Uncertainty of Vertical Acceleration in body frame
float	yawRateUnc	Uncertainty of Heading Rate
float	pitchUnc	Uncertainty of Body pitch

#### 5.7.1.5 struct telux::loc::LLAInfo

The location info is calculated according to the vehicle's GNSS antenna where as Vehicle Reference Point (VRP) refers to a point on the vehicle where the display of the car sits. The VRP based info is calculated by adding that extra difference between GNSS antenna and the VRP on the top where the location info is received. The VRP parameters can be configured through [ILocationConfigurator::configureLeverArm](#). [LLAInfo](#) specifies latitude, longitude and altitude info of location for VRP-based.

**Data fields**

Type	Field	Description
double	latitude	Latitude, in unit of degrees, range [-90.0, 90.0].
double	longitude	Longitude, in unit of degrees, range [-180.0, 180.0].
float	altitude	Altitude above the WGS 84 reference ellipsoid, in unit of meters.

**5.7.1.6 struct telux::loc::TimeInfo****Data fields**

Type	Field	Description
<a href="#">GnssTime</a> ↔ <a href="#">Validity</a>	validityMask	Validity mask for below fields
uint16_t	systemWeek	Extended week number at reference tick. Unit: Week. Set to 65535 if week number is unknown. For GPS: Calculated from midnight, Jan. 6, 1980. OTA decoded 10 bit GPS week is extended to map between: [NV6264 to (NV6264 + 1023)]. NV6264: Minimum GPS week number configuration. Default value of NV6264: 1738 For BDS: Calculated from 00:00:00 on January 1, 2006 of Coordinated Universal Time (UTC). For GAL: Calculated from 00:00 UT on Sunday August 22, 1999 (midnight between August 21 and August 22).
uint32_t	systemMsec	Time in to the current week at reference tick. Unit: Millisecond. Range: 0 to 604799999. Check for systemClkTimeUncMs before use
float	systemClk↔ TimeBias	System clock time bias (sub-millisecond) Units: Millisecond Note: System time (TOW Millisecond) = systemMsec - systemClkTimeBias. Check for systemClkTimeUncMs before use.
float	systemClk↔ TimeUncMs	Single sided maximum time bias uncertainty Units: Millisecond
uint32_t	refFCount	FCount (free running HW timer) value. Don't use for relative time purpose due to possible discontinuities. Unit: Millisecond
uint32_t	numClock↔ Resets	Number of clock resets/discontinuities detected, affecting the local hardware counter value.

**5.7.1.7 struct telux::loc::GlonassTimeInfo****Data fields**

Type	Field	Description
uint16_t	gloDays	GLONASS day number in four years. Refer to GLONASS ICD. Applicable only for GLONASS and shall be ignored for other constellations. If unknown shall be set to 65535
<a href="#">TimeValidity</a>	validityMask	Validity mask for below fields
uint32_t	gloMsec	GLONASS time of day in Millisecond. Refer to GLONASS ICD. Units: Millisecond Check for gloClkTimeUncMs before use



Type	Field	Description
float	gloClkTime↔ Bias	GLONASS clock time bias (sub-millisecond) Units: Millisecond Note: GLO time (TOD Millisecond) = gloMsec - gloClkTimeBias. Check for gloClkTimeUncMs before use.
float	gloClkTime↔ UncMs	Single sided maximum time bias uncertainty Units: Millisecond
uint32_t	refFCount	FCount (free running HW timer) value. Don't use for relative time purpose due to possible discontinuities. Unit: Millisecond
uint32_t	numClock↔ Resets	Number of clock resets/discontinuities detected, affecting the local hardware counter value.
uint8_t	gloFourYear	GLONASS four year number from 1996. Refer to GLONASS ICD. Applicable only for GLONASS and shall be ignored for other constellations. If unknown shall be set to 255

### 5.7.1.8 union telux::loc::SystemTimeInfo

#### Data fields

Type	Field	Description
<a href="#">TimeInfo</a>	gps	
<a href="#">TimeInfo</a>	gal	
<a href="#">TimeInfo</a>	bds	
<a href="#">TimeInfo</a>	qzss	
<a href="#">GlonassTime↔ Info</a>	glo	

### 5.7.1.9 struct telux::loc::SystemTime

#### Data fields

Type	Field	Description
<a href="#">GnssSystem</a>	gnssSystem↔ TimeSrc	Specifies GNSS system time reported. Mandatory field
<a href="#">SystemTime↔ Info</a>	time	Reporting of GPS system time is recommended. If GPS time is unknown & other satellite system time is known, it should be reported. Mandatory field

Type	Field	Description
------	-------	-------------

### 5.7.1.10 struct telux::loc::GnssMeasurementInfo

#### Data fields

Type	Field	Description
GnssSignal	gnssSignalType	GnssSignalType mask
GnssSystem	gnss↔ Constellation	Specifies GNSS Constellation Type
uint16_t	gnssSvId	GNSS SV ID. For GPS: 1 to 32 For GLONASS: 65 to 96. When slot-number to SV ID mapping is unknown, set as 255. For SBAS: 120 to 151 For QZSS-L1CA:193 to 197 For BDS: 201 to 237 For GAL: 301 to 336

### 5.7.1.11 struct telux::loc::SvUsedInPosition

Specify the set of SVs that are used to calculate GnssLocation.

#### Data fields

Type	Field	Description
uint64_t	gps	Specify the set of SVs from GPS constellation that are used to compute the position. Bit 0 to Bit 31 corresponds to GPS SV id 1 to 32.
uint64_t	glo	Specify the set of SVs from GLONASS constellation that are used to compute the position. Bit 0 to Bit 31 corresponds to GLO SV id 65 to 96.
uint64_t	gal	Specify the set of SVs from GALILEO constellation that are used to compute the position. Bit 0 to Bit 35 corresponds to GAL SV id 301 to 336.
uint64_t	bds	Specify the set of SVs from BEIDOU constellation that are used to compute the position. Bit 0 to Bit 36 corresponds to BDS SV id 201 to 237.
uint64_t	qzss	Specify the set of SVs from QZSS constellation that are used to compute the position. Bit 0 to Bit 4 corresponds to BDS SV id 193 to 197.

### 5.7.1.12 struct telux::loc::GnssData

**Data fields**

Type	Field	Description
GnssData↔ Validity	gnssData↔ Mask[Gnss↔ DataSignal↔ Types::GNSS↔ _DATA_MA↔ X_NUMBER↔ _OF_SIGNA↔ L_TYPES]	bitwise OR of GnssDataValidityType
double	jammer↔ Ind[GnssData↔ SignalTypes::↔ GNSS_DAT↔ A_MAX_NU↔ MBER_OF_↔ SIGNAL_TY↔ PES]	Jammer Indication Each index represents the measurement for the signal type in GnssDataSignalTypes
double	agc[Gnss↔ DataSignal↔ Types::GNSS↔ _DATA_MA↔ X_NUMBER↔ _OF_SIGNA↔ L_TYPES]	Automatic gain control Each index represents the measurement for the signal type in GnssDataSignalTypes

**5.7.1.13 struct telux::loc::SvBlackListInfo**

Specify parameters related to enable/disable SVs

**Data fields**

Type	Field	Description
Gnss↔ Constellation↔ Type	constellation	constellation for the sv
uint32_t	svId	sv id for the constellation: 0 means blacklist for all SVIDs of a given constellation type GLONASS SV id range: 65 to 96 QZSS SV id range: 193 to 197 BDS SV id range: 201 to 237 GAL SV id range: 301 to 336 SBAS SV id range: 120 to 158 and 183 to 191

**5.7.1.14 struct telux::loc::LeverArmParams**

Specify parameters related to lever arm

**Data fields**

Type	Field	Description
float	forwardOffset	Offset along the vehicle forward axis, in unit of meters

Type	Field	Description
float	sidewaysOffset	Offset along the vehicle starboard axis, in unit of meters
float	upOffset	Offset along the vehicle up axis, in unit of meters

### 5.7.1.15 struct telux::loc::ExternalLocationInfo

Specifies parameters related to External Location information.

#### Data fields

Type	Field	Description
uint64_t	timestamp	UTC timestamp elapsed since January 1, 1970, in unit of milliseconds.
double	latitude	Latitude, in unit of degrees, range [-90.0, 90.0].
double	longitude	Longitude, in unit of degrees, range [-180.0, 180.0].
float	horizontal↔ Accuracy	Horizontal accuracy, in unit of meters. The best results are expected when the accuracy is estimated with 68% horizontal confidence.

### 5.7.1.16 struct telux::loc::GnssMeasurementsData

Specify the signal measurement information such as satellite vehicle pseudo range, satellite vehicle time, carrier phase measurement etc. from GNSS positioning engine.

#### Data fields

Type	Field	Description
<a href="#">Gnss↔ Measurements↔ DataValidity</a>	valid	Bitwise OR of GnssMeasurementsDataValidityType to specify the valid fields in <a href="#">GnssMeasurementsData</a> .
int16_t	svId	Specify satellite vehicle ID number.
<a href="#">Gnss↔ Constellation↔ Type</a>	svType	SV constellation type.
double	timeOffsetNs	Time offset when the measurement was taken, in unit of nanoseconds.
<a href="#">Gnss↔ Measurements↔ StateValidity</a>	stateMask	Bitwise OR of GnssMeasurementsStateValidityType to specify the GNSS measurement state.
int64_t	receivedSv↔ TimeNs	Received GNSS time of the week in nanoseconds when the measurement was taken.
int64_t	receivedSv↔ Time↔ UncertaintyNs	Satellite time. All SV times in the current measurement block are already propagated to a common reference time epoch, in unit of nano seconds.
double	carrierTo↔ NoiseDbHz	Signal strength, carrier to noise ratio, in unit of dB-Hz.
double	pseudorange↔ RateMps	Uncorrected pseudorange rate, in unit of metres/second.

Type	Field	Description
double	pseudorange↔ Rate↔ Uncertainty↔ Mps	Uncorrected pseudorange rate uncertainty, in unit of meters/second.
<a href="#">Gnss↔ Measurements↔ AdrState↔ Validity</a>	adrStateMask	Bitwise OR of GnssMeasurementsAdrStateValidityType.
double	adrMeters	Accumulated delta range, in unit of meters.
double	adr↔ Uncertainty↔ Meters	Accumulated delta range uncertainty, in unit of meters.
float	carrier↔ FrequencyHz	Carrier frequency of the tracked signal, in unit of Hertz.
int64_t	carrierCycles	The number of full carrier cycles between the receiver and the satellite.
double	carrierPhase	The RF carrier phase that the receiver has detected.
double	carrierPhase↔ Uncertainty	The RF carrier phase uncertainty.
<a href="#">Gnss↔ Measurements↔ Multipath↔ Indicator</a>	multipath↔ Indicator	Multipath indicator, could be unknown, present or not present.
double	signalTo↔ NoiseRatioDb	Signal to noise ratio, in unit of dB.
double	agcLevelDb	Automatic gain control level, in unit of dB.
<a href="#">GnssSignal</a>	gnssSignalType	GnssSignalType mask
double	baseband↔ CarrierToNoise	Carrier-to-noise ratio of the signal measured at baseband, in unit of dB-Hz.
double	fullInter↔ SignalBias	The full inter-signal bias (ISB) in nanoseconds. This value is the sum of the estimated receiver-side and the space-segment-side inter-system bias, inter-frequency bias and inter-code bias.
double	fullInter↔ SignalBias↔ Uncertainty	Uncertainty associated with the full inter-signal bias in nanoseconds.

### 5.7.1.17 struct telux::loc::GnssMeasurementsClock

Specify GNSS measurements clock. The main equation describing the relationship between various components is:  $utcTimeNs = timeNs - (fullBiasNs + biasNs) - leapSecond * 1,000,000,000$

#### Data fields

Type	Field	Description
<a href="#">Gnss↔ Measurements↔ ClockValidity</a>	valid	Bitwise OR of GnssMeasurementsClockValidityType.
int16_t	leapSecond	Leap second, in unit of seconds.

Type	Field	Description
int64_t	timeNs	Time, monotonically increasing as long as the power is on, in unit of nanoseconds.
double	time↔ UncertaintyNs	Time uncertainty (one sigma), in unit of nanoseconds.
int64_t	fullBiasNs	Full bias, in unit of nanoseconds.
double	biasNs	Sub-nanoseconds bias, in unit of nanoseconds.
double	bias↔ UncertaintyNs	Bias uncertainty (one sigma), in unit of nanoseconds.
double	driftNsps	Clock drift, in unit of nanoseconds/second.
double	drift↔ Uncertainty↔ Nsps	Clock drift uncertainty (one sigma), in unit of nanoseconds/second.
uint32_t	hwClock↔ Discontinuity↔ Count	HW clock discontinuity count - incremented for each discontinuity in HW clock.

### 5.7.1.18 struct telux::loc::GnssMeasurements

Specify GNSS measurements clock and data. [GnssMeasurementInfo](#) is used to convey the satellite vehicle info whose measurements are actually used to generate the current position report. While [GnssMeasurements](#) contains the satellite measurements that device observed during tracking session, regardless the measurement is used or not used to compute the fix. Furthermore [GnssMeasurements](#) contains much richer set of information which can enable other third party engines to utilize the measurements and compute the position by itself.

#### Data fields

Type	Field	Description
<a href="#">Gnss↔ Measurements↔ Clock</a>	clock	GNSS measurements clock info.
vector< <a href="#">Gnss↔ Measurements↔ Data</a> >	measurements	GNSS measurements data.

### 5.7.1.19 struct telux::loc::LeapSecondChangeInfo

#### Data fields

Type	Field	Description
<a href="#">TimeInfo</a>	timeInfo	GPS timestamp that corresponds to the last known leap second change event. The info can be available on two scenario: 1: This leap second change event has been scheduled and yet to happen 2: This leap second change event has already happened and next leap second change event has not yet been scheduled.
uint8_t	leapSeconds↔ BeforeChange	Number of leap seconds prior to the leap second change event that corresponds to the timestamp at timeInfo.

Type	Field	Description
uint8_t	leapSeconds↔ AfterChange	Number of leap seconds after the leap second change event that corresponds to the timestamp at timeInfo.

### 5.7.1.20 struct telux::loc::LeapSecondInfo

#### Data fields

Type	Field	Description
<a href="#">LeapSecond↔ InfoValidity</a>	valid	
uint8_t	current	Current leap seconds, in unit of seconds. This info will only be available only if the leap second change info is not available.
<a href="#">LeapSecond↔ ChangeInfo</a>	info	

### 5.7.1.21 struct telux::loc::LocationSystemInfo

#### Data fields

Type	Field	Description
<a href="#">Location↔ SystemInfo↔ Validity</a>	valid	
<a href="#">LeapSecond↔ Info</a>	info	

### 5.7.1.22 struct telux::loc::GnssEnergyConsumedInfo

Specify the info regarding energy consumed by GNSS engine.

#### Data fields

Type	Field	Description
<a href="#">GnssEnergy↔ Consumed↔ InfoValidity</a>	valid	Bitwise OR of GnssEnergyConsumedInfoValidityType to specify the valid fields in <a href="#">GnssEnergyConsumedInfo</a> .
uint64_t	energySince↔ FirstBoot	Energy consumed by the modem GNSS engine since device first ever bootup, in unit of 0.1 milli watt seconds.

### 5.7.1.23 struct telux::loc::RobustLocationVersion

Specify the versioning info of robust location module for the GNSS standard position engine (SPE).

#### Data fields

Type	Field	Description
uint8_t	major	Major version number.

Type	Field	Description
uint16_t	minor	Minor version number.

#### 5.7.1.24 struct telux::loc::RobustLocationConfiguration

Specify the robust location configuration used by the GNSS standard position engine (SPE)

##### Data fields

Type	Field	Description
<a href="#">RobustLocationConfig</a>	validMask	Validity mask
bool	enabled	Specify whether robust location feature is enabled or not.
bool	enabledForE911	Specify whether robust location feature is enabled or not when device is on E911 call.
<a href="#">RobustLocationVersion</a>	version	Specify the version info of robust location module used by the GNSS standard position engine (SPE).

#### 5.7.1.25 struct telux::loc::NmeaConfig

Specify the Nmea Config Parameters

##### Data fields

Type	Field	Description
<a href="#">NmeaSentenceConfig</a>	sentenceConfig	Specify the sentences to be configured.
<a href="#">GeodeticDatumType</a>	datumType	Specify the datum type to be configured.

#### 5.7.1.26 struct telux::loc::BodyToSensorMountParams

Specify vehicle body-to-Sensor mount parameters for use by dead reckoning positioning engine.

##### Data fields

Type	Field	Description
float	rollOffset	The misalignment of the sensor board along the horizontal plane of the vehicle chassis measured looking from the vehicle to forward direction. In unit of degrees. Range: [-180.0, 180.0].
float	yawOffset	The misalignment along the horizontal plane of the vehicle chassis measured looking from the vehicle to the right side. Positive pitch indicates vehicle is inclined such that forward wheels are at higher elevation than rear wheels. In unit of degrees. Range: [-180.0, 180.0].



Type	Field	Description
float	pitchOffset	The angle between the vehicle forward direction and the sensor axis as seen from the top of the vehicle, and measured in counterclockwise direction. In unit of degrees. Range: [-180.0, 180.0].
float	offsetUnc	Single uncertainty number that may be the largest of the uncertainties for roll offset, pitch offset and yaw offset. In unit of degrees. Range: [-180.0, 180.0].

### 5.7.1.27 struct telux::loc::DREngineConfiguration

Specify the dead reckoning engine configuration parameters.

#### Data fields

Type	Field	Description
<a href="#">DRConfig</a> ↔ <a href="#">Validity</a>	validMask	Specify the valid fields.
<a href="#">BodyTo</a> ↔ <a href="#">SensorMount</a> ↔ <a href="#">Params</a>	mountParam	Body to sensor mount parameters used by dead reckoning positioning engine.
float	speedFactor	Vehicle Speed Scale Factor configuration input for the dead reckoning positioning engine. The multiplicative scale factor is applied to the received Vehicle Speed value (in meter/second) to obtain the true Vehicle Speed. Range is [0.9 to 1.1]. Note: The scale factor is specific to a given vehicle make & model.
float	speedFactor↔ Unc	Vehicle Speed Scale Factor Uncertainty (68% confidence) configuration input for the dead reckoning positioning engine. Range is [0.0 to 0.1]. Note: The scale factor uncertainty is specific to a given vehicle make & model.
float	gyroFactor	Gyroscope Scale Factor configuration input for the dead reckoning positioning engine. The multiplicative scale factor is applied to received gyroscope value to obtain the true value. Range is [0.9 to 1.1]. Note: The scale factor is specific to the Gyroscope sensor and typically derived from either sensor data-sheet or from actual calibration.
float	gyroFactorUnc	Gyroscope Scale Factor uncertainty (68% confidence) configuration input for the dead reckoning positioning engine. Range is [0.0 to 0.1]. Note: The scale factor uncertainty is specific to the Gyroscope sensor and typically derived from either sensor data-sheet or from actual calibration.

### 5.7.1.28 struct telux::loc::XtraConfig

Xtra feature configuration parameters

**Data fields**

Type	Field	Description
uint32_t	download↔ IntervalMinute	Number of minutes between periodic, consecutive successful XTRA assistance data downloads.  If 0 is specified, modem default download for XTRA assistance data will be performed.
uint32_t	download↔ TimeoutSec	Connection timeout when connecting backend for both xtra assistance data download and NTP time download.  If 0 is specified, the download timeout value will use device default values.
uint32_t	download↔ RetryInterval↔ Minute	Interval to wait before retrying for xtra assistance data's download in case of failure.  If 0 is specified, XTRA download retry will follow device default behavior and downloadRetryAttempts will also use device default value.
uint32_t	download↔ RetryAttempts	Total number of allowed retry attempts for assistance data's download in case of failure.  If 0 is specified, XTRA download retry will follow device default behavior and downloadRetryIntervalMinute will also use device default value.
string	caPath	Path to the certificate authority (CA) repository that needs to be used for XTRA assistance data download. If empty string is specified, device default CA repository will be used.
vector< string >	serverURLs	URLs from which XTRA assistance data will be fetched. At least one and up to three URLs need to be configured when this API is used.  The URLs, if provided, shall include the port number to be used for download.  Valid xtra server URLs should start with "https://".  Example of a valid URL : <a href="https://path.exampleserver.net:443">https://path.exampleserver.net:443</a>
vector< string >	ntpServerURLs	URLs for NTP server to fetch current time.  If no NTP server URL is provided, then device will use the default NTP server.  The URLs, if provided, shall include the port number to be used for download.  Example of a valid ntp server URL is: ntp.exampleserver.com:123.
bool	isIntegrity↔ Download↔ Enabled	Enable or disable XTRA integrity download.  true: enable XTRA integrity download. false: disable XTRA integrity download.
uint32_t	integrity↔ Download↔ IntervalMinute	Download interval for xtra integrity, only applicable if XTRA integrity download is enabled.  If 0 is specified, the download timeout value will use device default value.

Type	Field	Description
<a href="#">DebugLogLevel</a>	daemon↔ DebugLogLevel	Level of debug log messages that will be logged.

### 5.7.1.29 struct telux::loc::XtraStatus

Specify Xtra assistant data's current status, validity and whether it is enabled.

#### Data fields

Type	Field	Description
bool	featureEnabled	XTRA assistance data and NTP time download is enabled or disabled.
<a href="#">XtraDataStatus</a>	xtraDataStatus	XTRA assistance data status. If XTRA assistance data download is not enabled, this field will be set to XTRA_DATA_STATUS_UNKNOWN.
uint32_t	xtraValidFor↔ Hours	Number of hours that xtra assistance data will remain valid.  This field will be valid when xtraDataStatus is set to XTRA_DATA_STATUS_VALID. For all other XtraDataStatus, this field will be set to 0.

### 5.7.1.30 class telux::loc::ILocationInfoBase

[ILocationInfoBase](#) provides interface to get basic position related information like latitude, longitude, altitude, timestamp.

#### Public member functions

- virtual [LocationInfoValidity](#) [getLocationInfoValidity](#) ()=0
- virtual [LocationTechnology](#) [getTechMask](#) ()=0
- virtual float [getSpeed](#) ()=0
- virtual double [getLatitude](#) ()=0
- virtual double [getLongitude](#) ()=0
- virtual double [getAltitude](#) ()=0
- virtual float [getHeading](#) ()=0
- virtual float [getHorizontalUncertainty](#) ()=0
- virtual float [getVerticalUncertainty](#) ()=0
- virtual uint64\_t [getTimeStamp](#) ()=0
- virtual float [getSpeedUncertainty](#) ()=0
- virtual float [getHeadingUncertainty](#) ()=0
- virtual uint64\_t [getElapsedRealTime](#) ()=0

- virtual uint64\_t `getElapsedRealTimeUncertainty ()=0`

### 5.7.1.30.1 Member Function Documentation

#### 5.7.1.30.1.1 virtual LocationInfoValidity telux::loc::ILocationInfoBase::getLocationInfoValidity ( ) [pure virtual]

Retrieves the validity of the Location basic Info.

#### Returns

Location basic validity mask.

#### 5.7.1.30.1.2 virtual LocationTechnology telux::loc::ILocationInfoBase::getTechMask ( ) [pure virtual]

Retrieves technology used in computing this fix.

#### Returns

Location technology mask.

#### 5.7.1.30.1.3 virtual float telux::loc::ILocationInfoBase::getSpeed ( ) [pure virtual]

Retrieves Speed.

#### Returns

speed in meters per second.

#### 5.7.1.30.1.4 virtual double telux::loc::ILocationInfoBase::getLatitude ( ) [pure virtual]

Retrieves latitude. Positive and negative values indicate northern and southern latitude respectively

- Units: Degrees
- Range: -90.0 to 90.0

#### Returns

Latitude if available else returns NaN.

#### 5.7.1.30.1.5 virtual double telux::loc::ILocationInfoBase::getLongitude ( ) [pure virtual]

Retrieves longitude. Positive and negative values indicate eastern and western longitude respectively

- Units: Degrees
- Range: -180.0 to 180.0

**Returns**

Longitude if available else returns NaN.

**5.7.1.30.1.6 virtual double telux::loc::ILocationInfoBase::getAltitude ( ) [pure virtual]**

Retrieves altitude above the WGS 84 reference ellipsoid.

- Units: Meters

**Returns**

Altitude if available else returns NaN.

**5.7.1.30.1.7 virtual float telux::loc::ILocationInfoBase::getHeading ( ) [pure virtual]**

Retrieves heading/bearing.

- Units: Degrees
- Range: 0 to 359.999

**Returns**

Heading if available else returns NaN.

**5.7.1.30.1.8 virtual float telux::loc::ILocationInfoBase::getHorizontalUncertainty ( ) [pure virtual]**

Retrieves the horizontal uncertainty.

**Returns**

Horizontal uncertainty.

**5.7.1.30.1.9 virtual float telux::loc::ILocationInfoBase::getVerticalUncertainty ( ) [pure virtual]**

Retrieves the vertical uncertainty.

- Units: Meters

**Returns**

Vertical uncertainty if available else returns NaN.

**5.7.1.30.1.10 virtual uint64\_t telux::loc::ILocationInfoBase::getTimeStamp ( ) [pure virtual]**

Retrieves UTC timeInfo for the location fix.

- Units: Milliseconds since Jan 1, 1970

**Returns**

TimeStamp in milliseconds if available else returns UNKNOWN\_TIMESTAMP which is zero(as UTC timeStamp has elapsed since January 1, 1970, it cannot be 0)

**5.7.1.30.1.11 virtual float telux::loc::ILocationInfoBase::getSpeedUncertainty ( ) [pure virtual]**

Retrieves 3-D speed uncertainty/accuracy.

- Units: Meters per Second

**Returns**

Speed uncertainty if available else returns NaN.

**5.7.1.30.1.12 virtual float telux::loc::ILocationInfoBase::getHeadingUncertainty ( ) [pure virtual]**

Retrieves heading uncertainty.

- Units: Degrees
- Range: 0 to 359.999

**Returns**

Heading uncertainty if available else returns NaN.

**5.7.1.30.1.13 virtual uint64\_t telux::loc::ILocationInfoBase::getElapsedRealTime ( ) [pure virtual]**

Boot timestamp corresponding to the UTC timestamp for Location fix.

- Units: Nano-second

**Returns**

elapsed real time.

**5.7.1.30.1.14 virtual uint64\_t telux::loc::ILocationInfoBase::getElapsedRealTimeUncertainty ( ) [pure virtual]**

Retrieves elapsed real time uncertainty.

- Units: Nano-second

**Returns**

elapsed real time uncertainty.

**5.7.1.31 class telux::loc::ILocationInfoEx**

[ILocationInfoEx](#) provides interface to get richer position related information like latitude, longitude, altitude and other information like time stamp, session status, dop, reliabilities, uncertainties etc.

**Public member functions**

- virtual [LocationInfoExValidity](#) [getLocationInfoExValidity](#) ()=0
- virtual float [getAltitudeMeanSeaLevel](#) ()=0
- virtual float [getPositionDop](#) ()=0
- virtual float [getHorizontalDop](#) ()=0
- virtual float [getVerticalDop](#) ()=0
- virtual float [getGeometricDop](#) ()=0
- virtual float [getTimeDop](#) ()=0
- virtual float [getMagneticDeviation](#) ()=0
- virtual [LocationReliability](#) [getHorizontalReliability](#) ()=0
- virtual [LocationReliability](#) [getVerticalReliability](#) ()=0
- virtual float [getHorizontalUncertaintySemiMajor](#) ()=0
- virtual float [getHorizontalUncertaintySemiMinor](#) ()=0
- virtual float [getHorizontalUncertaintyAzimuth](#) ()=0
- virtual float [getEastStandardDeviation](#) ()=0
- virtual float [getNorthStandardDeviation](#) ()=0
- virtual uint16\_t [getNumSvUsed](#) ()=0
- virtual [SvUsedInPosition](#) [getSvUsedInPosition](#) ()=0
- virtual void [getSVIds](#) (std::vector< uint16\_t > &idsOfUsedSVs)=0
- virtual [SbasCorrection](#) [getSbasCorrection](#) ()=0
- virtual [GnssPositionTech](#) [getPositionTechnology](#) ()=0
- virtual [GnssKinematicsData](#) [getBodyFrameData](#) ()=0
- virtual std::vector< [GnssMeasurementInfo](#) > [getmeasUsageInfo](#) ()=0
- virtual [SystemTime](#) [getGnssSystemTime](#) ()=0
- virtual float [getTimeUncMs](#) ()=0
- virtual [telux::common::Status](#) [getLeapSeconds](#) (uint8\_t &leapSeconds)=0
- virtual [telux::common::Status](#) [getVelocityEastNorthUp](#) (std::vector< float >

&velocityEastNorthUp)=0

- virtual `telux::common::Status` `getVelocityUncertaintyEastNorthUp` (std::vector< float > &velocityUncertaintyEastNorthUp)=0
- virtual `uint8_t` `getCalibrationConfidencePercent` ()=0
- virtual `DrCalibrationStatus` `getCalibrationStatus` ()=0
- virtual `LocationAggregationType` `getLocOutputEngType` ()=0
- virtual `PositioningEngine` `getLocOutputEngMask` ()=0
- virtual `float` `getConformityIndex` ()=0
- virtual `LLAInfo` `getVRPBasedLLA` ()=0
- virtual `std::vector< float >` `getVRPBasedENUVelocity` ()=0
- virtual `AltitudeType` `getAltitudeType` ()=0
- virtual `ReportStatus` `getReportStatus` ()=0
- virtual `uint32_t` `getIntegrityRiskUsed` ()=0
- virtual `float` `getProtectionLevelAlongTrack` ()=0
- virtual `float` `getProtectionLevelCrossTrack` ()=0
- virtual `float` `getProtectionLevelVertical` ()=0
- virtual `std::vector< uint16_t >` `getDgnssStationIds` ()=0

### 5.7.1.31.1 Member Function Documentation

#### 5.7.1.31.1.1 virtual `LocationInfoExValidity` `telux::loc::ILocationInfoEx::getLocationInfoExValidity` ( ) [pure virtual]

Retrieves the validity of the location info ex. It provides the validity of various information like dop, reliabilities, uncertainties etc.

#### Returns

Location ex validity mask

#### 5.7.1.31.1.2 virtual `float` `telux::loc::ILocationInfoEx::getAltitudeMeanSeaLevel` ( ) [pure virtual]

Retrieves the altitude with respect to mean sea level.

- Units: Meters

#### Returns

Altitude with respect to mean sea level if available else returns NaN.



**5.7.1.31.1.3 virtual float telux::loc::ILocationInfoEx::getPositionDop ( ) [pure virtual]**

Retrieves position dilution of precision.

**Returns**

Position dilution of precision if available else returns NaN. Range: 1 (highest accuracy) to 50 (lowest accuracy)

**5.7.1.31.1.4 virtual float telux::loc::ILocationInfoEx::getHorizontalDop ( ) [pure virtual]**

Retrieves horizontal dilution of precision.

**Returns**

Horizontal dilution of precision if available else returns NaN. Range: 1 (highest accuracy) to 50 (lowest accuracy)

**5.7.1.31.1.5 virtual float telux::loc::ILocationInfoEx::getVerticalDop ( ) [pure virtual]**

Retrieves vertical dilution of precision.

**Returns**

Vertical dilution of precision if available else returns NaN Range: 1 (highest accuracy) to 50 (lowest accuracy)

**5.7.1.31.1.6 virtual float telux::loc::ILocationInfoEx::getGeometricDop ( ) [pure virtual]**

Retrieves geometric dilution of precision.

**Returns**

geometric dilution of precision.

**5.7.1.31.1.7 virtual float telux::loc::ILocationInfoEx::getTimeDop ( ) [pure virtual]**

Retrieves time dilution of precision.

**Returns**

Time dilution of precision.

**5.7.1.31.1.8 virtual float telux::loc::ILocationInfoEx::getMagneticDeviation ( ) [pure virtual]**

Retrieves the difference between the bearing to true north and the bearing shown on magnetic compass. The deviation is positive when the magnetic north is east of true north.

- Units: Degrees

**Returns**

Magnetic Deviation if available else returns NaN

**5.7.1.31.1.9 virtual LocationReliability telux::loc::ILocationInfoEx::getHorizontalReliability ( ) [pure virtual]**

Specifies the reliability of the horizontal position.

**Returns**

[LocationReliability](#) of the horizontal position if available else returns UNKNOWN.

**5.7.1.31.1.10 virtual LocationReliability telux::loc::ILocationInfoEx::getVerticalReliability ( ) [pure virtual]**

Specifies the reliability of the vertical position.

**Returns**

[LocationReliability](#) of the vertical position if available else returns UNKNOWN.

**5.7.1.31.1.11 virtual float telux::loc::ILocationInfoEx::getHorizontalUncertaintySemiMajor ( ) [pure virtual]**

Retrieves semi-major axis of horizontal elliptical uncertainty.

- Units: Meters

**Returns**

Semi-major horizontal elliptical uncertainty if available else returns NaN.

**5.7.1.31.1.12 virtual float telux::loc::ILocationInfoEx::getHorizontalUncertaintySemiMinor ( ) [pure virtual]**

Retrieves semi-minor axis of horizontal elliptical uncertainty.

- Units: Meters

**Returns**

Semi-minor horizontal elliptical uncertainty if available else returns NaN.

**5.7.1.31.1.13 virtual float telux::loc::ILocationInfoEx::getHorizontalUncertaintyAzimuth ( ) [pure virtual]**

Retrieves elliptical horizontal uncertainty azimuth of orientation.

- Units: Decimal degrees

- Range: 0 to 180

**Returns**

Elliptical horizontal uncertainty azimuth of orientation if available else returns NaN.

**5.7.1.31.1.14 virtual float telux::loc::ILocationInfoEx::getEastStandardDeviation ( ) [pure virtual]**

Retrieves east standard deviation.

- Units: Meters

**Returns**

East Standard Deviation.

**5.7.1.31.1.15 virtual float telux::loc::ILocationInfoEx::getNorthStandardDeviation ( ) [pure virtual]**

Retrieves north standard deviation.

- Units: Meters

**Returns**

North Standard Deviation.

**5.7.1.31.1.16 virtual uint16\_t telux::loc::ILocationInfoEx::getNumSvUsed ( ) [pure virtual]**

Retrieves number of satellite vehicles used in position report.

**Returns**

number of Sv used.

**5.7.1.31.1.17 virtual SvUsedInPosition telux::loc::ILocationInfoEx::getSvUsedInPosition ( ) [pure virtual]**

Retrives the set of satellite vehicles that are used to calculate position.

**Returns**

set of satellite vehicles for different constellations.

**5.7.1.31.1.18 virtual void telux::loc::ILocationInfoEx::getSVIds ( std::vector< uint16\_t > & idsOfUsedSVs ) [pure virtual]**

Retrieves GNSS Satellite Vehicles used in position data.

out	<i>idsOfUsedSVs</i>	Vector of Satellite Vehicle identifiers.
-----	---------------------	--

**5.7.1.31.1.19 virtual SbasCorrection telux::loc::ILocationInfoEx::getSbasCorrection ( ) [pure virtual]**

Retrieves navigation solution mask used to indicate SBAS corrections.

**Returns**

- SBAS (Satellite Based Augmentation System) Correction mask used.

**5.7.1.31.1.20 virtual GnssPositionTech telux::loc::ILocationInfoEx::getPositionTechnology ( ) [pure virtual]**

Retrieves position technology mask used to indicate which technology is used.

**Returns**

- Position technology used in computing this fix.

**5.7.1.31.1.21 virtual GnssKinematicsData telux::loc::ILocationInfoEx::getBodyFrameData ( ) [pure virtual]**

Retrieves position related information.

**5.7.1.31.1.22 virtual std::vector<GnssMeasurementInfo> telux::loc::ILocationInfoEx::getmeasUsageInfo ( ) [pure virtual]**

Retrieves gnss measurement usage info.

**5.7.1.31.1.23 virtual SystemTime telux::loc::ILocationInfoEx::getGnssSystemTime ( ) [pure virtual]**

Retrieves type of gnss system.

**Returns**

- Type of Gnss System.

**5.7.1.31.1.24 virtual float telux::loc::ILocationInfoEx::getTimeUncMs ( ) [pure virtual]**

Retrieves time uncertainty.

**Returns**

- Time uncertainty in milliseconds.

**5.7.1.31.1.25 virtual telux::common::Status telux::loc::ILocationInfoEx::getLeapSeconds ( uint8\_t & leapSeconds ) [pure virtual]**

Retrieves leap seconds if available.

**Parameters**

out	<i>leapSeconds</i>	- leap seconds • Units: Seconds
-----	--------------------	------------------------------------

**Returns**

Status of leap seconds.

**5.7.1.31.1.26 virtual telux::common::Status telux::loc::ILocationInfoEx::getVelocityEastNorthUp ( std::vector< float > & velocityEastNorthUp ) [pure virtual]**

Retrieves east, North, Up velocity if available.

**Parameters**

out	<i>velocityEastNorthUp</i>	- east, North, Up velocity • Units: Meters/second
-----	----------------------------	--

**Returns**

Status of availability of east, North, Up velocity.

**5.7.1.31.1.27 virtual telux::common::Status telux::loc::ILocationInfoEx::getVelocityUncertainty↔ EastNorthUp ( std::vector< float > & velocityUncertaintyEastNorthUp ) [pure virtual]**

Retrieves east, North, Up velocity uncertainty if available.

**Parameters**

out	<i>velocity↔ UncertaintyEast↔ NorthUp</i>	- east, North, Up velocity uncertainty • Units: Meters/second
-----	---	--

**Returns**

Status of availability of east, North, Up velocity uncertainty.

**5.7.1.31.1.28** `virtual uint8_t telux::loc::ILocationInfoEx::getCalibrationConfidencePercent ( ) [pure virtual]`

Sensor calibration confidence percent, range [0, 100].

**Returns**

the percentage of calibration taking all the parameters into account.

**5.7.1.31.1.29** `virtual DrCalibrationStatus telux::loc::ILocationInfoEx::getCalibrationStatus ( ) [pure virtual]`

Sensor calibration status.

**Returns**

mask indicating the calibration status with respect to different parameters.

**5.7.1.31.1.30** `virtual LocationAggregationType telux::loc::ILocationInfoEx::getLocOutputEngType ( ) [pure virtual]`

Location engine type. When the type is set to LOC\_ENGINE\_SRC\_FUSED, the fix is the propagated/aggregated reports from all engines running on the system (e.g.: DR/SPE/PPE) based QTI algorithm. To check which location engine contributes to the fused output, check for locOutputEngMask.

**Returns**

the type of engine that was used for calculating the position fix.

**5.7.1.31.1.31** `virtual PositioningEngine telux::loc::ILocationInfoEx::getLocOutputEngMask ( ) [pure virtual]`

When loc output eng type is set to fused, this field indicates the set of engines contribute to the fix.

**Returns**

the combination of position engines used in calculating the position report when the loc output end type is set to fused.

**5.7.1.31.1.32** `virtual float telux::loc::ILocationInfoEx::getConformityIndex ( ) [pure virtual]`

When robust location is enabled, this field will indicate how well the various input data considered for navigation solution conforms to expectations.

**Returns**

values in the range [0.0, 1.0], with 0.0 for least conforming and 1.0 for most conforming.

**5.7.1.31.1.33 virtual LLAInfo telux::loc::ILocationInfoEx::getVRPBasedLLA ( ) [pure virtual]**

Vehicle Reference Point(VRP) based latitude, longitude and altitude information.

**5.7.1.31.1.34 virtual std::vector<float> telux::loc::ILocationInfoEx::getVRPBasedENUVelocity ( ) [pure virtual]**

VRP-based east, north and up velocity information.

#### Returns

- vector of directional velocities in this order {east velocity, north velocity, up velocity}

**5.7.1.31.1.35 virtual AltitudeType telux::loc::ILocationInfoEx::getAltitudeType ( ) [pure virtual]**

Determination of altitude is assumed or calculated. ASSUMED means there may not be enough satellites to determine the precise altitude.

#### Returns

altitude type ASSUMED/CALCULATED or if not available then UNKNOWN.

**5.7.1.31.1.36 virtual ReportStatus telux::loc::ILocationInfoEx::getReportStatus ( ) [pure virtual]**

Indicates the status of this report in terms of how optimally the report was calculated by the engine.

#### Returns

Status of the report. Returns [ReportStatus::UNKNOWN](#) if status is unavailable.

**5.7.1.31.1.37 virtual uint32\_t telux::loc::ILocationInfoEx::getIntegrityRiskUsed ( ) [pure virtual]**

Integrity risk used for protection level parameters. Unit of 2.5e-10. Valid range is [1 to (4e9-1)]. Values other than valid range means integrity risk is disabled and [ILocationInfoEx::getProtectionLevelAlongTrack](#), [ILocationInfoEx::getProtectionLevelCrossTrack](#) and [ILocationInfoEx::getProtectionLevelVertical](#) will not be available.

**5.7.1.31.1.38 virtual float telux::loc::ILocationInfoEx::getProtectionLevelAlongTrack ( ) [pure virtual]**

Along-track protection level at specified integrity risk, in unit of meter.

**5.7.1.31.1.39 virtual float telux::loc::ILocationInfoEx::getProtectionLevelCrossTrack ( ) [pure virtual]**

Cross-track protection level at specified integrity risk, in unit of meter.

**5.7.1.31.1.40** virtual float telux::loc::ILocationInfoEx::getProtectionLevelVertical ( ) [pure virtual]

Vertical component protection level at specified integrity risk, in unit of meter.

**5.7.1.31.1.41** virtual std::vector<uint16\_t> telux::loc::ILocationInfoEx::getDgnssStationIds ( ) [pure virtual]

List of DGNSS station IDs providing corrections. Range:

- SBAS – 120 to 158 and 183 to 191
- Monitoring station – 1000-2023 (Station ID biased by 1000)
- Other values reserved.

### 5.7.1.32 class telux::loc::ISVInfo

[ISVInfo](#) provides interface to retrieve information about Satellite Vehicles, their position and health status.

#### Public member functions

- virtual [GnssConstellationType](#) getConstellation ()=0
- virtual uint16\_t getId ()=0
- virtual [SVHealthStatus](#) getSVHealthStatus ()=0
- virtual [SVStatus](#) getStatus ()=0
- virtual [SVInfoAvailability](#) getHasEphemeris ()=0
- virtual [SVInfoAvailability](#) getHasAlmanac ()=0
- virtual [SVInfoAvailability](#) getHasFix ()=0
- virtual float getElevation ()=0
- virtual float getAzimuth ()=0
- virtual float getSnr ()=0
- virtual float getCarrierFrequency ()=0
- virtual [GnssSignal](#) getSignalType ()=0

#### 5.7.1.32.1 Member Function Documentation

**5.7.1.32.1.1** virtual [GnssConstellationType](#) telux::loc::ISVInfo::getConstellation ( ) [pure virtual]

Indicates to which constellation this satellite vehicle belongs.

#### Returns

[GnssConstellationType](#) if available else returns UNKNOWN.



**5.7.1.32.1.2 virtual uint16\_t telux::loc::ISVInfo::getId ( ) [pure virtual]**

GNSS satellite vehicle ID.

**Returns**

Identifier of the satellite vehicle otherwise 0(as 0 is not an ID for any of the SVs)

**5.7.1.32.1.3 virtual SVHealthStatus telux::loc::ISVInfo::getSVHealthStatus ( ) [pure virtual]**

Health status of satellite vehicle.

**Returns**

HealthStatus of Satellite Vehicle if available else returns UNKNOWN.

- [SVHealthStatus](#)

**5.7.1.32.1.4 virtual SVStatus telux::loc::ISVInfo::getStatus ( ) [pure virtual]**

Status of satellite vehicle.

**Note**

This API is work-in-progress and is subject to change.

**Returns**

Satellite Vehicle Status if available else returns UNKNOWN.

- [SVStatus](#)

**5.7.1.32.1.5 virtual SVInfoAvailability telux::loc::ISVInfo::getHasEphemeris ( ) [pure virtual]**

Indicates whether ephemeris information(which allows the receiver to calculate the satellite's position) is available.

**Returns**

[SVInfoAvailability](#) if Ephemeris exists or not else returns UNKNOWN.

**5.7.1.32.1.6 virtual SVInfoAvailability telux::loc::ISVInfo::getHasAlmanac ( ) [pure virtual]**

Indicates whether almanac information(which allows receivers to know which satellites are available for tracking) is available.

**Returns**

[SVInfoAvailability](#) if almanac exists or not else returns UNKNOWN.

**5.7.1.32.1.7 virtual SVInfoAvailability telux::loc::ISVInfo::getHasFix ( ) [pure virtual]**

Indicates whether the satellite is used in computing the fix.

**Returns**

[SVInfoAvailability](#), if satellite used or not else returns UNKNOWN.

**5.7.1.32.1.8 virtual float telux::loc::ISVInfo::getElevation ( ) [pure virtual]**

Retrieves satellite vehicle elevation angle.

- Units: Degrees
- Range: 0 to 90

**Returns**

Elevation if available else returns NaN.

**5.7.1.32.1.9 virtual float telux::loc::ISVInfo::getAzimuth ( ) [pure virtual]**

Retrieves satellite vehicle azimuth angle.

- Units: Degrees
- Range: 0 to 360

**Returns**

Azimuth if available else returns NaN.

**5.7.1.32.1.10 virtual float telux::loc::ISVInfo::getSnr ( ) [pure virtual]**

Retrieves satellite vehicle signal-to-noise ratio.

- Units: dB-Hz

**Returns**

SNR if available else returns NaN.

**5.7.1.32.1.11 virtual float telux::loc::ISVInfo::getCarrierFrequency ( ) [pure virtual]**

Indicates the carrier frequency of the signal tracked.

**Returns**

carrier frequency in Hz else returns UNKNOWN\_CARRIER\_FREQ frequency when not supported.

### 5.7.1.32.1.12 virtual GnssSignal telux::loc::ISVInfo::getSignalType ( ) [pure virtual]

Indicates the validity for different types of signal for gps, galileo, beidou etc.

#### Returns

signalType mask else return UNKNOWN\_SIGNAL\_MASK when not supported.

### 5.7.1.33 class telux::loc::IGnssSVInfo

[IGnssSVInfo](#) provides interface to retrieve the list of SV info available and whether altitude is assumed or calculated.

#### Public member functions

- virtual [AltitudeType](#) getAltitudeType ()=0
- virtual std::vector< std::shared\_ptr< [ISVInfo](#) > > getSVInfoList ()=0

#### 5.7.1.33.1 Member Function Documentation

##### 5.7.1.33.1.1 virtual AltitudeType telux::loc::IGnssSVInfo::getAltitudeType ( ) [pure virtual]

Indicates whether altitude is assumed or calculated.

#### Returns

[AltitudeType](#) if available else returns UNKNOWN.

##### 5.7.1.33.1.2 virtual std::vector<std::shared\_ptr<ISVInfo> > telux::loc::IGnssSVInfo::getSVInfoList ( ) [pure virtual]

Pointer to satellite vehicles information for all GNSS constellations except GPS.

#### Returns

Vector of pointer of [ISVInfo](#) object if available else returns empty vector.

### 5.7.1.34 class telux::loc::IGnssSignalInfo

[IGnssSignalInfo](#) provides interface to retrieve GNSS data information like jammer metrics and automatic gain control for satellite signal type.

#### Public member functions

- virtual [GnssData](#) getGnssData ()=0

### 5.7.1.34.1 Member Function Documentation

#### 5.7.1.34.1.1 virtual GnssData telux::loc::IGnssSignalInfo::getGnssData ( ) [pure virtual]

Retrieves jammer metric and Automatic Gain Control(AGC) corresponding to signal types. Jammer metric is linearly proportional to the sum of jammer and noise power at the GNSS antenna port.

#### Returns

List of jammer metric and a list of automatic gain control for signal type.

### 5.7.1.35 class telux::loc::LocationFactory

[LocationFactory](#) allows creation of location manager.

#### Public member functions

- `std::shared_ptr< ILocationManager > getLocationManager ( )`
- `std::shared_ptr< ILocationConfigurator > getLocationConfigurator ( )`
- `std::shared_ptr< IDgnssManager > getDgnssManager (DgnssDataFormat dataFormat=DgnssDataFormat::DATA_FORMAT_RTCM_3)`

#### Static Public Member Functions

- `static LocationFactory & getInstance ( )`

### 5.7.1.35.1 Member Function Documentation

#### 5.7.1.35.1.1 static LocationFactory& telux::loc::LocationFactory::getInstance ( ) [static]

Get Location Factory instance.

#### 5.7.1.35.1.2 std::shared\_ptr<ILocationManager> telux::loc::LocationFactory::getLocationManager ( )

Get instance of Location Manager

#### Returns

Pointer of [ILocationManager](#) object.

#### 5.7.1.35.1.3 std::shared\_ptr<ILocationConfigurator> telux::loc::LocationFactory::getLocationConfigurator ( )

Get instance of Location Configurator.

#### Returns

Pointer of [ILocationConfigurator](#) object.

#### 5.7.1.35.1.4 `std::shared_ptr<IDgnssManager> telux::loc::LocationFactory::getDgnssManager ( DgnssDataFormat dataFormat = DgnssDataFormat::DATA_FORMAT_RTCM_3 )`

Get instance of Dgnss manager

#### Returns

Pointer of [IDgnssManager](#) object.

### 5.7.1.36 class `telux::loc::ILocationListener`

Listener class for getting location updates and satellite vehicle information.

The methods in listener can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe.

#### Public member functions

- virtual void `onBasicLocationUpdate` (const `std::shared_ptr< ILocationInfoBase >` &locationInfo)
- virtual void `onDetailedLocationUpdate` (const `std::shared_ptr< ILocationInfoEx >` &locationInfo)
- virtual void `onDetailedEngineLocationUpdate` (const `std::vector< std::shared_ptr< ILocationInfoEx > >` &locationEngineInfo)
- virtual void `onGnssSVInfo` (const `std::shared_ptr< IGnssSVInfo >` &gnssSVInfo)
- virtual void `onGnssSignalInfo` (const `std::shared_ptr< IGnssSignalInfo >` &info)
- virtual void `onGnssNmeaInfo` (uint64\_t timestamp, const `std::string` &nmea)
- virtual void `onGnssMeasurementsInfo` (const `telux::loc::GnssMeasurements` &measurementInfo)
- virtual `~ILocationListener` ()

#### 5.7.1.36.1 Constructors and Destructors

##### 5.7.1.36.1.1 `virtual telux::loc::ILocationListener::~~ILocationListener ( ) [virtual]`

Destructor of [ILocationListener](#)

#### 5.7.1.36.2 Member Function Documentation

##### 5.7.1.36.2.1 `virtual void telux::loc::ILocationListener::onBasicLocationUpdate ( const std::shared_ptr< ILocationInfoBase > & locationInfo ) [virtual]`

This function is called when device receives location update.

#### Parameters

in	<i>locationInfo</i>	- Location information like latitude, longitude, timeInfo other information such as heading, altitude and velocity etc.
----	---------------------	---

### 5.7.1.36.2.2 virtual void telux::loc::ILocationListener::onDetailedLocationUpdate ( const std::shared\_ptr< ILocationInfoEx > & locationInfo ) [virtual]

This function is called when device receives Gnss location update.

#### Parameters

in	<i>locationInfo</i>	- Contains richer set of location information like latitude, longitude, timeInfo, heading, altitude, velocity and other information such as deviations, elliptical accuracies etc.
----	---------------------	--

### 5.7.1.36.2.3 virtual void telux::loc::ILocationListener::onDetailedEngineLocationUpdate ( const std::vector< std::shared\_ptr< ILocationInfoEx > > & locationEngineInfo ) [virtual]

This function is called when device receives multiple Gnss location update from the different engine types requested, which are SPE/PPE/FUSED. This API will be called ONLY if we use startDetailedEngineReports.

#### Parameters

in	<i>locationInfo</i>	- Contains a list of location infos. Each element in the list corresponds to one of SPE/PPE/FUSED.
----	---------------------	--

### 5.7.1.36.2.4 virtual void telux::loc::ILocationListener::onGnssSVInfo ( const std::shared\_ptr< IGnssSVInfo > & gnssSVInfo ) [virtual]

This function is called when device receives GNSS satellite information.

#### Parameters

in	<i>gnssSVInfo</i>	- GNSS satellite information
----	-------------------	------------------------------

### 5.7.1.36.2.5 virtual void telux::loc::ILocationListener::onGnssSignalInfo ( const std::shared\_ptr< IGnssSignalInfo > & info ) [virtual]

This function is called when device receives GNSS data information like jammer metrics and automatic gain control for satellite signal type.

#### Parameters

in	<i>info</i>	- GNSS signal info
----	-------------	--------------------

### 5.7.1.36.2.6 virtual void telux::loc::ILocationListener::onGnssNmeaInfo ( uint64\_t timestamp, const std::string & nmea ) [virtual]

This function is called when device receives GNSS NMEA sentences.

in	<i>timestamp</i>	- Timestamp
in	<i>nmea</i>	- Nmea sentence

#### 5.7.1.36.2.7 virtual void telux::loc::ILocationListener::onGnssMeasurementsInfo ( const telux::loc::ILocationListener::ILocationListener & locationListener, const GnssMeasurements & measurementInfo ) [virtual]

This function is called when device receives signal measurement information such as satellite vehicle pseudo range, satellite vehicle clock time, carrier phase measurement etc.

#### Parameters

in	<i>measurementInfo</i>	- GNSS measurement information
----	------------------------	--------------------------------

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.7.1.37 class telux::loc::ILocationSystemInfoListener

#### Public member functions

- virtual void [onLocationSystemInfo](#) (const [LocationSystemInfo](#) &locationSystemInfo)
- virtual [~ILocationSystemInfoListener](#) ()

#### 5.7.1.37.1 Constructors and Destructors

##### 5.7.1.37.1.1 virtual telux::loc::ILocationSystemInfoListener::~~ILocationSystemInfoListener ( ) [virtual]

Destructor of [ILocationSystemInfoListener](#)

#### 5.7.1.37.2 Member Function Documentation

##### 5.7.1.37.2.1 virtual void telux::loc::ILocationSystemInfoListener::onLocationSystemInfo ( const LocationSystemInfo & locationSystemInfo ) [virtual]

This function is called when device receives location related system information such as leap second change.

#### Parameters

in	<i>locationSystemInfo</i>	- contains location system information such as current leap seconds change
----	---------------------------	--

### 5.7.1.38 class telux::loc::ILocationInjectionListener

#### Public member functions

- virtual void [onStartInjection](#) (const uint32\_t timeInMilliseconds)
- virtual void [onStopInjection](#) ()
- virtual [~ILocationInjectionListener](#) ()

#### 5.7.1.38.1 Constructors and Destructors

5.7.1.38.1.1 virtual telux::loc::ILocationInjectionListener::~~ILocationInjectionListener ( ) [virtual]

Destructor of [ILocationSystemInfoListener](#)

#### 5.7.1.38.2 Member Function Documentation

5.7.1.38.2.1 virtual void telux::loc::ILocationInjectionListener::onStartInjection ( const uint32\_t *timeInMilliseconds* ) [virtual]

This function is called when the Location service is ready to start receiving the location reports using the LocationConfigurator::injectLocationData API.

#### Parameters

in	<i>timeInMillis</i>	- contains the time in milliseconds which represents the Max rate of at which the LocationConfigurator::injectLocationData API can be called.
----	---------------------	---

5.7.1.38.2.2 virtual void telux::loc::ILocationInjectionListener::onStopInjection ( ) [virtual]

This function notifies the Location service's intent to stop injecting the location reports using the LocationConfigurator::injectLocationData API.

### 5.7.1.39 class telux::loc::ILocationConfigListener

[ILocationConfigListener](#) interface is used to receive notifications related to configuration events.

The listener method can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe.

#### Public member functions

- virtual void [onXtraStatusUpdate](#) (const [XtraStatus](#) xtraStatus)
- virtual [~ILocationConfigListener](#) ()



### 5.7.1.39.1 Constructors and Destructors

5.7.1.39.1.1 virtual `telux::loc::ILocationConfigListener::~~ILocationConfigListener ( )` [virtual]

### 5.7.1.39.2 Member Function Documentation

5.7.1.39.2.1 virtual void `telux::loc::ILocationConfigListener::onXtraStatusUpdate ( const XtraStatus xtraStatus )` [virtual]

The API is invoked when there is any update in the Xtra assistance data.

#### Parameters

in	<i>xtraStatus</i>	- Xtra assistant data's current status, validity and whether it is enabled.
----	-------------------	---

### 5.7.1.40 class `telux::loc::ILocationManager`

`ILocationManager` provides interface to register and remove listeners. It also allows to set and get configuration/ criteria for position reports. The new APIs(`registerListenerEx`, `deRegisterListenerEx`, `startDetailedReports`, `startBasicReports`) and old/deprecated APIs(`registerListener`, `removeListener`, `setPositionReportTimeout`, `setHorizontalAccuracyLevel`, `setMinIntervalForReports`) should not be used interchangeably, either the new APIs should be used or the old APIs should be used.

#### Public Types

- using `GetEnergyConsumedCallback` = `std::function< void(telux::loc::GnssEnergyConsumedInfo energyConsumed, telux::common::ErrorCode error)>`
- using `GetTerrestrialInfoCallback` = `std::function< void(const std::shared_ptr< ILocationInfoBase > terrestrialInfo)>`

#### Public member functions

- virtual bool `isSubsystemReady ()=0`
- virtual `std::future< bool > onSubsystemReady ()=0`
- virtual `telux::common::Status registerListenerEx (std::weak_ptr< ILocationListener > listener)=0`
- virtual `telux::common::Status deRegisterListenerEx (std::weak_ptr< ILocationListener > listener)=0`
- virtual `telux::common::Status startDetailedReports (uint32_t interval, telux::common::ResponseCallback callback=nullptr, GnssReportTypeMask reportMask=DEFAULT_GNSS_REPORT)=0`
- virtual `telux::common::Status startDetailedEngineReports (uint32_t interval, LocReqEngine engineType, telux::common::ResponseCallback callback=nullptr, GnssReportTypeMask reportMask=DEFAULT_GNSS_REPORT)=0`
- virtual `telux::common::Status startBasicReports (uint32_t distanceInMeters, uint32_t intervalInMs, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status registerForSystemInfoUpdates (std::weak_ptr<`

[ILocationSystemInfoListener](#) > listener, [telux::common::ResponseCallback](#) callback=nullptr)=0

- virtual [telux::common::Status](#) deRegisterForSystemInfoUpdates (std::weak\_ptr<  
[ILocationSystemInfoListener](#) > listener, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) requestEnergyConsumedInfo ([GetEnergyConsumedCallback](#) cb)=0
- virtual [telux::common::Status](#) stopReports ([telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) getTerrestrialPosition (uint32\_t timeoutMsec, [TerrestrialTechnology](#)  
techMask, [GetTerrestrialInfoCallback](#) cb, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) cancelTerrestrialPositionRequest ([telux::common::ResponseCallback](#)  
callback=nullptr)=0
- virtual [~ILocationManager](#) ()

### 5.7.1.40.1 Member Typedef Documentation

**5.7.1.40.1.1** using [telux::loc::ILocationManager::GetEnergyConsumedCallback](#) = std::function<void([telux::loc::GnssEnergyConsumedInfo](#) energyConsumed, [telux::common::ErrorCode](#) error)>

This function is called with the response to [getEnergyConsumedInfoUpdate](#) API.

#### Parameters

in	<i>energyConsumed</i>	- Information regarding energy consumed by Gnss engine.
in	<i>error</i>	- Return code which indicates whether the operation succeeded or not.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.7.1.40.1.2** using [telux::loc::ILocationManager::GetTerrestrialInfoCallback](#) = std::function<void( const std::shared\_ptr<[ILocationInfoBase](#)> terrestrialInfo)>

This function is called with the response to [getTerrestrialPosition](#) API.

#### Parameters

in	<i>terrestrialInfo</i>	- basic position related information.
----	------------------------	---------------------------------------

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

## 5.7.1.40.2 Constructors and Destructors

### 5.7.1.40.2.1 virtual telux::loc::ILocationManager::~~ILocationManager ( ) [virtual]

Destructor of [ILocationManager](#)

## 5.7.1.40.3 Member Function Documentation

### 5.7.1.40.3.1 virtual bool telux::loc::ILocationManager::isSubsystemReady ( ) [pure virtual]

Checks the status of location subsystems and returns the result.

#### Returns

True if location subsystem is ready for service otherwise false.

### 5.7.1.40.3.2 virtual std::future<bool> telux::loc::ILocationManager::onSubsystemReady ( ) [pure virtual]

Wait for location subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when location subsystem is ready.

### 5.7.1.40.3.3 virtual telux::common::Status telux::loc::ILocationManager::registerListenerEx ( std::weak\_ptr< ILocationListener > *listener* ) [pure virtual]

Register a listener for specific updates from location manager like location, jamming info and satellite vehicle info. If enhanced position, using Dead Reckoning etc., is enabled, enhanced fixes will be provided. Otherwise raw GNSS fixes will be provided. The position reports will start only when startDetailedReports or startBasicReports is invoked.

#### Parameters

in	<i>listener</i>	- Pointer of <a href="#">ILocationListener</a> object that processes the notification.
----	-----------------	--

#### Returns

Status of registerListener i.e success or suitable status code.

### 5.7.1.40.3.4 virtual telux::common::Status telux::loc::ILocationManager::deRegisterListenerEx ( std::weak\_ptr< ILocationListener > *listener* ) [pure virtual]

Remove a previously registered listener.

in	<i>listener</i>	- Previously registered <a href="#">ILocationListener</a> that needs to be removed.
----	-----------------	---

## Returns

Status of removeListener success or suitable status code

**5.7.1.40.3.5 virtual telux::common::Status telux::loc::ILocationManager::startDetailedReports ( uint32\_t interval, telux::common::ResponseCallback callback = nullptr, GnsReportTypeMask reportMask = DEFAULT\_GNSS\_REPORT ) [pure virtual]**

Starts the richer location reports by configuring the time between them as the interval. Any of the 3 APIs that is startDetailedReports or startDetailedEngineReports or startBasicReports can be called one after the other irrespective of order, without calling stopReports in between any of them and the API which is called last will be honored for providing the callbacks. If multiple clients invoke this API with different interval, then all the clients will be benefited with interval which is smallest among all the intervals. Calling this Api will result in [ILocationListener::onDetailedLocationUpdate](#), [ILocationListener::onGnssSVInfo](#), [ILocationListener::onGnssSignalInfo](#), [ILocationListener::onGnssNmeaInfo](#) and [ILocationListener::onGnssMeasurementsInfo](#) APIs on the listener being invoked, assuming they have not been disabled using the GnsReportTypeMask. If a client issues second request to this API then new request for GnsReportTypeMask will over write the previous call to this API.

## Parameters

in	<i>interval</i>	- Minimum time interval between two consecutive reports in milliseconds.
----	-----------------	--

E.g. If minInterval is 1000 milliseconds, reports will be provided with a periodicity of 1 second or more depending on the number of applications listening to location updates.

## Parameters

in	<i>callback</i>	- Optional callback to get the response of set minimum interval for reports.
in	<i>reportMask</i>	- Optional field to specify which reports a client is interested in. By default all the reports will be enabled.

## Returns

Status of startDetailedReports i.e. success or suitable status code.

**5.7.1.40.3.6 virtual telux::common::Status telux::loc::ILocationManager::startDetailedEngineReports ( uint32\_t interval, LocReqEngine engineType, telux::common::ResponseCallback callback = nullptr, GnsReportTypeMask reportMask = DEFAULT\_GNSS\_REPORT ) [pure virtual]**

Starts a session which may provide richer default combined position reports and position reports from other engines. The fused position report type will always be supported if at least one engine in the system is producing valid report. Any of the 3 APIs that is startDetailedReports or startDetailedEngineReports or

startBasicReports can be called one after the other irrespective of order, without calling stopReports in between any of them and the API which is called last will be honored for providing the callbacks. If multiple clients invoke this API with different interval, then all the clients will be benefited with interval which is smallest among all the intervals. Calling this Api will result in [ILocationListener::onDetailedEngineLocationUpdate](#), [ILocationListener::onGnssSVInfo](#), [ILocationListener::onGnssSignalInfo](#), [ILocationListener::onGnssNmeaInfo](#) and [ILocationListener::onGnssMeasurementsInfo](#) APIs on the listener being invoked, assuming they have not been disabled using the GnssReportTypeMask. If a client issues second request to this API then new request for GnssReportTypeMask will over write the previous call to this API.

#### Parameters

in	<i>interval</i>	- Minimum time interval between two consecutive reports in milliseconds.
----	-----------------	--

E.g. If minInterval is 1000 milliseconds, reports will be provided with a periodicity of 1 second or more depending on the number of applications listening to location updates.

#### Parameters

in	<i>engineType</i>	- The type of engine requested for fixes such as SPE or PPE or FUSED. The FUSED includes all the engines that are running to generate the fixes such as reports from SPE, PPE and DRE.
in	<i>callback</i>	- Optional callback to get the response of set minimum interval for reports.
in	<i>reportMask</i>	- Optional field to specify which reports a client is interested in. By default all the reports will be enabled.

#### Returns

Status of startDetailedEngineReports i.e. success or suitable status code.

**5.7.1.40.3.7 virtual telux::common::Status telux::loc::ILocationManager::startBasicReports ( uint32\_t distanceInMeters, uint32\_t intervalInMs, telux::common::ResponseCallback callback = nullptr ) [pure virtual]**

Starts the Location report by configuring the time and distance between the consecutive reports. Any of the 3 APIs that is startDetailedReports or startDetailedEngineReports or startBasicReports can be called one after the other irrespective of order, without calling stopReports in between any of them and the API which is called last will be honored for providing the callbacks. If multiple clients invoke this API with different interval, then all the clients will be benefited with interval which is smallest among all the intervals.

This Api enables the onBasicLocationUpdate Api on the listener.

#### Parameters

in	<i>distanceInMeters</i>	- distanceInMeters between two consecutive reports in meters. intervalInMs - Minimum time interval between two consecutive reports in milliseconds.
----	-------------------------	--

E.g. If `intervalInMs` is 1000 milliseconds and `distanceInMeters` is 100m, reports will be provided according to the condition that happens first. So we need to provide both the parameters for evaluating the report.

The underlying system may have a minimum distance threshold(e.g. 1 meter). Effective distance will not be smaller than this lower bound.

The effective distance may have a granularity level higher than 1 m, e.g. 5 m. So `distanceInMeters` being 59 may be honored at 60 m, depending on the system.

Where there is another application in the system having a session with shorter distance, this client may benefit and receive reports at that distance.

### Parameters

<code>in</code>	<code>callback</code>	- Optional callback to get the response of set minimum distance for reports.
-----------------	-----------------------	--

### Returns

Status of `startBasicReports` i.e. success or suitable status code.

**5.7.1.40.3.8** `virtual telux::common::Status telux::loc::ILocationManager::registerForSystemInfoUpdates ( std::weak_ptr< ILocationSystemInfoListener > listener, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

This API registers a [ILocationSystemInfoListener](#) listener and will receive information related to location system that are not tied with location fix session, e.g.: next leap second event. The `startBasicReports`, `startDetailedReports`, `startDetailedEngineReports` does not need to be called before calling this API, in order to receive updates.

### Parameters

<code>in</code>	<code>listener</code>	- Pointer of <a href="#">ILocationSystemInfoListener</a> object.
<code>in</code>	<code>callback</code>	- Optional callback to get the response of location system info.

### Returns

Status of `getLocationSystemInfo` i.e success or suitable status code.

### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.7.1.40.3.9** `virtual telux::common::Status telux::loc::ILocationManager::deRegisterForSystemInfoUpdates ( std::weak_ptr< ILocationSystemInfoListener > listener, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

This API removes a previously registered listener and will also stop receiving informations related to location system for that particular listener.

in	<i>listener</i>	- Previously registered <a href="#">ILocationSystemInfoListener</a> that needs to be removed.
in	<i>callback</i>	- Optional callback to get the response of location system info.

**Returns**

Status of deRegisterForSystemInfoUpdates success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.7.1.40.3.10 virtual telux::common::Status telux::loc::ILocationManager::requestEnergyConsumedInfo ( GetEnergyConsumedCallback *cb* ) [pure virtual]

This API receives information on energy consumed by modem GNSS engine. If this API is called on this object while this is already a pending request, then it will overwrite the callback to be invoked and the callback from the previous invocation will not be called.

**Parameters**

in	<i>cb</i>	- callback to get the information of Gnss energy consumed.
----	-----------	--

**Returns**

Status of requestEnergyConsumedInfo i.e success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.7.1.40.3.11 virtual telux::common::Status telux::loc::ILocationManager::stopReports ( telux::common::ResponseCallback *callback = nullptr* ) [pure virtual]

This API will stop reports started using startDetailedReports or startBasicReports or registerListener or setMinIntervalForReports.

**Parameters**

in	<i>callback</i>	- Optional callback to get the response of stop reports.
----	-----------------	--

**Returns**

Status of stopReports i.e. success or suitable status code.

**5.7.1.40.3.12** `virtual telux::common::Status telux::loc::ILocationManager::getTerrestrialPosition ( uint32_t timeoutMsec, TerrestrialTechnology techMask, GetTerrestrialInfoCallback cb, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

This API retrieves single-shot terrestrial position using the set of specified terrestrial technologies. This API can be invoked even while there is an on-going tracking session that was started using `startBasicReports/startDetailedReports/startDetailedEngineReports`. If this API is invoked while there is already a pending request for terrestrial position, the request will fail and `ResponseCallback` will get invoked with `ErrorCode::OP_IN_PROGRESS`. To cancel a pending request, use [ILocationManager::cancelTerrestrialPositionRequest](#). Before using this API, user consent needs to be set true via `ILocationConfigurator::provideConsentForTerrestrialPositioning`.

#### Parameters

in	<i>timeoutMsec</i>	- the time in milliseconds within which the client is expecting a response. If the system is unable to provide a report within this time, the <code>ResponseCallback</code> will be invoked with <code>ErrorCode::OPERATION_TIMEOUT</code> .
in	<i>techMask</i>	- the set of terrestrial technologies that are allowed to be used for producing the position.
in	<i>cb</i>	- callback to receive terrestrial position. This callback will only be invoked when <code>ResponseCallback</code> is invoked with <code>SUCCESS</code> .
in	<i>callback</i>	- Optional callback to get the response of <code>getTerrestrialPosition</code> .

#### Returns

Status of `getTerrestrialPosition` i.e success or suitable status code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.7.1.40.3.13** `virtual telux::common::Status telux::loc::ILocationManager::cancelTerrestrialPositionRequest ( telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

This API cancels the pending request invoked by [ILocationManager::getTerrestrialPosition](#). If this API is invoked while there is no pending request for terrestrial position from [ILocationManager::getTerrestrialPosition](#), then `ResponseCallback` will be invoked with `ErrorCode::INVALID_ARGUMENTS`.

#### Parameters

in	<i>callback</i>	- Optional callback to get the response of <code>cancelTerrestrialPositionRequest</code> .
----	-----------------	--

#### Returns

Status of `cancelTerrestrialPositionRequest` i.e success or suitable status code.



**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.7.2 Enumeration Type Documentation****5.7.2.1 enum telux::loc::DgnssDataFormat [strong]**

Defines RTCM injection data format

**Enumerator**

**DATA\_FORMAT\_UNKNOWN** Source data format is unknown  
**DATA\_FORMAT\_RTCM\_3** Source data format is RTCM\_3  
**DATA\_FORMAT\_3GPP\_RTK\_R15** Source data format is 3GPP RTK Rel-15

**5.7.2.2 enum telux::loc::DgnssStatus [strong]**

Defines status reported by cdfw for RTCM injection.

**Enumerator**

**DATA\_SOURCE\_NOT\_SUPPORTED** Dgnss subsystem doesn't support the data source  
**DATA\_FORMAT\_NOT\_SUPPORTED** Dgnss subsystem doesn't support the data format  
**OTHER\_SOURCE\_IN\_USE** After the source injects the data, dgnss subsystem discovers there is another higher priority source injecting the data at the same time, and the current injected data is dropped  
**MESSAGE\_PARSE\_ERROR** There is a parsing error such as unrecognized format, CRC check failure, value range check failure, etc.; the injected data is dropped  
**DATA\_SOURCE\_NOT\_USABLE** Data source is not usable anymore

**5.7.2.3 enum telux::loc::HorizontalAccuracyLevel [strong]**

Defines the horizontal accuracy level of the fix.

**Enumerator**

**LOW** Client requires low horizontal accuracy  
**MEDIUM** Client requires medium horizontal accuracy  
**HIGH** Client requires high horizontal accuracy

**5.7.2.4 enum telux::loc::LocationReliability [strong]**

Specifies the reliability of the position.

**Enumerator**

**UNKNOWN** Unknown location reliability  
**NOT\_SET** Location reliability is not set  
**VERY\_LOW** Location reliability is very low  
**LOW** Location reliability is low, little or no cross-checking is possible

**MEDIUM** Location reliability is medium, limited cross-check passed

**HIGH** Location reliability is high, strong cross-check passed

### 5.7.2.5 enum telux::loc::SbasCorrectionType

Defines Satellite Based Augmentation System(SBAS) corrections. SBAS contributes to improve the performance of GNSS system.

#### Enumerator

**SBAS\_CORRECTION\_IONO** Bit mask to specify whether SBAS ionospheric correction is used

**SBAS\_CORRECTION\_FAST** Bit mask to specify whether SBAS fast correction is used

**SBAS\_CORRECTION\_LONG** Bit mask to specify whether SBAS long correction is used

**SBAS\_INTEGRITY** Bit mask to specify whether SBAS integrity information is used

**SBAS\_CORRECTION\_DGNSS** Bit mask to specify whether SBAS DGNSS correction is used

**SBAS\_CORRECTION\_RTK** Bit mask to specify whether SBAS RTK correction is used

**SBAS\_CORRECTION\_PPP** Bit mask to specify whether SBAS PPP correction is used

**SBAS\_CORRECTION\_RTK\_FIXED** Bit mask to specify whether SBAS RTK fixed correction is used

**SBAS\_CORRECTION\_ONLY\_SBAS\_CORRECTED\_SV\_USED\_** Bit mask to specify only SBAS corrected SV is used

**SBAS\_COUNT** Bitset

### 5.7.2.6 enum telux::loc::AltitudeType [strong]

Indicates whether altitude is assumed or calculated.

#### Enumerator

**UNKNOWN** Unknown altitude type

**CALCULATED** Altitude is calculated

**ASSUMED** Altitude is assumed, there may not be enough satellites to determine the precise altitude

### 5.7.2.7 enum telux::loc::GnssConstellationType [strong]

Defines constellation type of GNSS.

#### Enumerator

**UNKNOWN** Unknown constellation type

**GPS** GPS satellite

**GALILEO** GALILEO satellite

**SBAS** SBAS satellite

**COMPASS** COMPASS satellite.

**Deprecated** constellation type is not supported.

**GLONASS** GLONASS satellite

**BDS** BDS satellite

**QZSS** QZSS satellite

**NAVIC** NAVIC satellite

### 5.7.2.8 enum telux::loc::SVHealthStatus [strong]

Health status indicates whether satellite is operational or not. This information comes from the most recent data transmitted in satellite almanacs.

#### Enumerator

**UNKNOWN** Unknown sv health status  
**UNHEALTHY** satellite is not operational and cannot be used in position calculations  
**HEALTHY** satellite is fully operational

### 5.7.2.9 enum telux::loc::SVStatus [strong]

Satellite vehicle processing status.

#### Enumerator

**UNKNOWN** Unknown sv status  
**IDLE** SV is not being actively processed  
**SEARCH** The system is searching for this SV  
**TRACK** SV is being tracked

### 5.7.2.10 enum telux::loc::SVInfoAvailability [strong]

Indicates whether Satellite Vehicle info like ephemeris and almanac are present or not

#### Enumerator

**UNKNOWN** Unknown sv info availability  
**YES** Ephemeris or Almanac exists  
**NO** Ephemeris or Almanac doesn't exist

### 5.7.2.11 enum telux::loc::GnssPositionTechType

Specifies which position technology was used to generate location information in the [ILocationInfoEx](#).

#### Enumerator

**GNSS\_DEFAULT** Technology used to generate location info is unknown.  
**GNSS\_SATELLITE** Satellites-based technology was used to generate location info.  
**GNSS\_CELLID** Cell towers were used to generate location info.  
**GNSS\_WIFI** Wi-Fi access points were used to generate location info.  
**GNSS\_SENSORS** Sensors were used to generate location info.  
**GNSS\_REFERENCE\_LOCATION** Reference location was used to generate location info.  
**GNSS\_INJECTED\_COARSE\_POSITION** Coarse position injected into the location engine was used to generate location info.  
**GNSS\_AFLT** AFLT was used to generate location info.  
**GNSS\_HYBRID** GNSS and network-provided measurements were used to generate location info.  
**GNSS\_PPE** Precise position engine was used to generate location info.  
**GNSS\_VEHICLE** Location was calculated using Vehicular data.  
**GNSS\_VISUAL** Location was calculated using Visual data.  
**GNSS\_PROPAGATED** Location was calculated using Propagation logic, which uses cached

measurements.

### 5.7.2.12 enum telux::loc::KinematicDataValidityType

Specifies related kinematics mask

#### Enumerator

**HAS\_LONG\_ACCEL** Navigation data has Forward Acceleration  
**HAS\_LAT\_ACCEL** Navigation data has Sideward Acceleration  
**HAS\_VERT\_ACCEL** Navigation data has Vertical Acceleration  
**HAS\_YAW\_RATE** Navigation data has Heading Rate  
**HAS\_PITCH** Navigation data has Body pitch  
**HAS\_LONG\_ACCEL\_UNC** Navigation data has Forward Acceleration  
**HAS\_LAT\_ACCEL\_UNC** Navigation data has Sideward Acceleration  
**HAS\_VERT\_ACCEL\_UNC** Navigation data has Vertical Acceleration  
**HAS\_YAW\_RATE\_UNC** Navigation data has Heading Rate  
**HAS\_PITCH\_UNC** Navigation data has Body pitch

### 5.7.2.13 enum telux::loc::GnssSystem [strong]

Specifies type of system.

#### Enumerator

**GNSS\_LOC\_SV\_SYSTEM\_UNKNOWN** UNKNOWN satellite.  
**GNSS\_LOC\_SV\_SYSTEM\_GPS** GPS satellite.  
**GNSS\_LOC\_SV\_SYSTEM\_GALILEO** GALILEO satellite.  
**GNSS\_LOC\_SV\_SYSTEM\_SBAS** SBAS satellite.  
**GNSS\_LOC\_SV\_SYSTEM\_COMPASS** COMPASS satellite.  
**Deprecated** constellation type is not supported.  
**GNSS\_LOC\_SV\_SYSTEM\_GLONASS** GLONASS satellite.  
**GNSS\_LOC\_SV\_SYSTEM\_BDS** BDS satellite.  
**GNSS\_LOC\_SV\_SYSTEM\_QZSS** QZSS satellite.  
**GNSS\_LOC\_SV\_SYSTEM\_NAVIC** NAVIC satellite.

### 5.7.2.14 enum telux::loc::GnssTimeValidityType

Validity field for different system time.

#### Enumerator

**GNSS\_SYSTEM\_TIME\_WEEK\_VALID**  
**GNSS\_SYSTEM\_TIME\_WEEK\_MS\_VALID**  
**GNSS\_SYSTEM\_CLK\_TIME\_BIAS\_VALID**  
**GNSS\_SYSTEM\_CLK\_TIME\_BIAS\_UNC\_VALID**  
**GNSS\_SYSTEM\_REF\_FCOUNT\_VALID**  
**GNSS\_SYSTEM\_NUM\_CLOCK\_RESETS\_VALID**

### 5.7.2.15 enum telux::loc::GlonassTimeValidity

Validity field for GLONASS time.

#### Enumerator

**GNSS\_CLO\_DAYS\_VALID**  
**GNSS\_GLOS\_MSEC\_VALID**  
**GNSS\_GLO\_CLK\_TIME\_BIAS\_VALID**  
**GNSS\_GLO\_CLK\_TIME\_BIAS\_UNC\_VALID**  
**GNSS\_GLO\_REF\_FCOUNT\_VALID**  
**GNSS\_GLO\_NUM\_CLOCK\_RESETS\_VALID**  
**GNSS\_GLO\_FOUR\_YEAR\_VALID**

### 5.7.2.16 enum telux::loc::GnssSignalType

GNSS Signal Type and RF Band

#### Enumerator

**GPS\_L1CA** GPS L1CA Signal  
**GPS\_L1C** GPS L1C Signal  
**GPS\_L2** GPS L2 RF Band  
**GPS\_L5** GPS L5 RF Band  
**GLONASS\_G1** GLONASS G1 (L1OF) RF Band  
**GLONASS\_G2** GLONASS G2 (L2OF) RF Band  
**GALILEO\_E1** GALILEO E1 RF Band  
**GALILEO\_E5A** GALILEO E5A RF Band  
**GALILEO\_E5B** GALILEO E5B RF Band  
**BEIDOU\_B1** BEIDOU B1 RF Band  
**BEIDOU\_B2** BEIDOU B2 RF Band  
**QZSS\_L1CA** QZSS L1CA RF Band  
**QZSS\_L1S** QZSS L1S RF Band  
**QZSS\_L2** QZSS L2 RF Band  
**QZSS\_L5** QZSS L5 RF Band  
**SBAS\_L1** SBAS L1 RF Band  
**BEIDOU\_B1I** BEIDOU B1I RF Band  
**BEIDOU\_B1C** BEIDOU B1C RF Band  
**BEIDOU\_B2I** BEIDOU B2I RF Band  
**BEIDOU\_B2AI** BEIDOU B2AI RF Band  
**NAVIC\_L5** NAVIC L5 RF Band  
**BEIDOU\_B2AQ** BEIDOU B2A\_Q RF Band

### 5.7.2.17 enum telux::loc::LocationTechnologyType

#### Enumerator

**LOC\_GNSS** location was calculated using GNSS  
**LOC\_CELL** location was calculated using Cell  
**LOC\_WIFI** location was calculated using WiFi  
**LOC\_SENSORS** location was calculated using Sensors

**LOC\_REFERENCE\_LOCATION** Location was calculated using Reference location.

**LOC\_INJECTED\_COARSE\_POSITION** Location was calculated using Coarse position injected into the location engine.

**LOC\_AFLT** Location was calculated using AFLT.

**LOC\_HYBRID** Location was calculated using GNSS and network-provided measurements.

**LOC\_PPE** Location was calculated using Precise position engine.

**LOC\_VEH** Location was calculated using Vehicular data.

**LOC\_VIS** Location was calculated using Visual data.

**LOC\_PROPAGATED** Location was calculated using Propagation logic, which uses cached measurements.

### 5.7.2.18 enum telux::loc::LocationValidityType

#### Enumerator

**HAS\_LAT\_LONG\_BIT** Location has valid latitude and longitude.

**HAS\_ALTITUDE\_BIT** Location has valid altitude.

**HAS\_SPEED\_BIT** Location has valid speed.

**HAS\_HEADING\_BIT** Location has valid heading.

**HAS\_HORIZONTAL\_ACCURACY\_BIT**

**HAS\_VERTICAL\_ACCURACY\_BIT** Location has valid vertical accuracy.

**HAS\_SPEED\_ACCURACY\_BIT** Location has valid speed accuracy.

**HAS\_HEADING\_ACCURACY\_BIT** Location has valid heading accuracy.

**HAS\_TIMESTAMP\_BIT** Location has valid timestamp.

**HAS\_ELAPSED\_REAL\_TIME\_BIT** Location has valid elapsed real time.

**HAS\_ELAPSED\_REAL\_TIME\_UNC\_BIT** Location has valid elapsed real time uncertainty.

### 5.7.2.19 enum telux::loc::LocationInfoExValidityType

Gnss Location Information mask flags

#### Enumerator

**HAS\_ALTITUDE\_MEAN\_SEA\_LEVEL** valid altitude mean sea level

**HAS\_DOP** valid pdop, hdop, and vdop

**HAS\_MAGNETIC\_DEVIATION** valid magnetic deviation

**HAS\_HOR\_RELIABILITY** valid horizontal reliability

**HAS\_VER\_RELIABILITY** valid vertical reliability

**HAS\_HOR\_ACCURACY\_ELIP\_SEMI\_MAJOR** valid ellipsoid semi major

**HAS\_HOR\_ACCURACY\_ELIP\_SEMI\_MINOR** valid ellipsoid semi minor

**HAS\_HOR\_ACCURACY\_ELIP\_AZIMUTH** valid accuracy ellipsoid azimuth

**HAS\_GNSS\_SV\_USED\_DATA** valid gnss sv used in pos data

**HAS\_NAV\_SOLUTION\_MASK** valid navSolutionMask

**HAS\_POS\_TECH\_MASK** valid LocPosTechMask

**HAS\_SV\_SOURCE\_INFO** valid LocSvInfoSource

**HAS\_POS\_DYNAMICS\_DATA** valid position dynamics data

**HAS\_EXT\_DOP** valid gdop, tdop

**HAS\_NORTH\_STD\_DEV** valid North standard deviation

**HAS\_EAST\_STD\_DEV** valid East standard deviation

**HAS\_NORTH\_VEL** valid North Velocity

**HAS\_EAST\_VEL** valid East Velocity

**HAS\_UP\_VEL** valid Up Velocity  
**HAS\_NORTH\_VEL\_UNC** valid North Velocity Uncertainty  
**HAS\_EAST\_VEL\_UNC** valid East Velocity Uncertainty  
**HAS\_UP\_VEL\_UNC** valid Up Velocity Uncertainty  
**HAS\_LEAP\_SECONDS** valid leap\_seconds  
**HAS\_TIME\_UNC** valid timeUncMs  
**HAS\_NUM\_SV\_USED\_IN\_POSITION** valid number of sv used  
**HAS\_CALIBRATION\_CONFIDENCE\_PERCENT** valid sensor calibrationConfidencePercent  
**HAS\_CALIBRATION\_STATUS** valid sensor calibrationConfidence  
**HAS\_OUTPUT\_ENG\_TYPE** valid output engine type  
**HAS\_OUTPUT\_ENG\_MASK** valid output engine mask  
**HAS\_CONFORMITY\_INDEX\_FIX** valid conformity index  
**HAS\_LLA\_VRP\_BASED** valid lla vrp based  
**HAS\_ENU\_VELOCITY\_VRP\_BASED** valid enu velocity vrp based  
**HAS\_ALTITUDE\_TYPE** valid altitude type  
**HAS\_REPORT\_STATUS** valid report status  
**HAS\_INTEGRITY\_RISK\_USED** valid integrity risk  
**HAS\_PROTECT\_LEVEL\_ALONG\_TRACK** valid protect level along track  
**HAS\_PROTECT\_LEVEL\_CROSS\_TRACK** valid protect level cross track  
**HAS\_PROTECT\_LEVEL\_VERTICAL** valid protect level vertical  
**HAS\_DGNSS\_STATION\_ID** valid dgnssStationId

### 5.7.2.20 enum telux::loc::GnssDataSignalTypes

#### Enumerator

**GNSS\_DATA\_SIGNAL\_TYPE\_GPS\_L1CA** GPS L1CA Signal  
**GNSS\_DATA\_SIGNAL\_TYPE\_GPS\_L1C** GPS L1C Signal  
**GNSS\_DATA\_SIGNAL\_TYPE\_GPS\_L2C\_L** GPS L2C\_L RF Band  
**GNSS\_DATA\_SIGNAL\_TYPE\_GPS\_L5\_Q** GPS L5\_Q RF Band  
**GNSS\_DATA\_SIGNAL\_TYPE\_GLONASS\_G1** GLONASS G1 (L1OF) RF Band  
**GNSS\_DATA\_SIGNAL\_TYPE\_GLONASS\_G2** GLONASS G2 (L2OF) RF Band  
**GNSS\_DATA\_SIGNAL\_TYPE\_GALILEO\_E1\_C** GALILEO E1\_C RF Band  
**GNSS\_DATA\_SIGNAL\_TYPE\_GALILEO\_E5A\_Q** GALILEO E5A\_Q RF Band  
**GNSS\_DATA\_SIGNAL\_TYPE\_GALILEO\_E5B\_Q** GALILEO E5B\_Q RF Band  
**GNSS\_DATA\_SIGNAL\_TYPE\_BEIDOU\_B1\_I** BEIDOU B1\_I RF Band  
**GNSS\_DATA\_SIGNAL\_TYPE\_BEIDOU\_B1C** BEIDOU B1C RF Band  
**GNSS\_DATA\_SIGNAL\_TYPE\_BEIDOU\_B2\_I** BEIDOU B2\_I RF Band  
**GNSS\_DATA\_SIGNAL\_TYPE\_BEIDOU\_B2A\_I** BEIDOU B2A\_I RF Band  
**GNSS\_DATA\_SIGNAL\_TYPE\_QZSS\_L1CA** QZSS L1CA RF Band  
**GNSS\_DATA\_SIGNAL\_TYPE\_QZSS\_L1S** QZSS L1S RF Band  
**GNSS\_DATA\_SIGNAL\_TYPE\_QZSS\_L2C\_L** QZSS L2C\_L RF Band  
**GNSS\_DATA\_SIGNAL\_TYPE\_QZSS\_L5\_Q** QZSS L5\_Q RF Band  
**GNSS\_DATA\_SIGNAL\_TYPE\_SBAS\_L1\_CA** SBAS L1\_CA RF Band  
**GNSS\_DATA\_SIGNAL\_TYPE\_NAVIC\_L5** NAVIC L5 RF Band  
**GNSS\_DATA\_SIGNAL\_TYPE\_BEIDOU\_B2A\_Q** BEIDOU B2A\_Q RF Band  
**GNSS\_DATA\_MAX\_NUMBER\_OF\_SIGNAL\_TYPES** Maximum number of signal types

### 5.7.2.21 enum telux::loc::GnssDataValidityType

#### Enumerator

**HAS\_JAMMER** Jammer Indicator is available  
**HAS\_AGC** AGC is available

### 5.7.2.22 enum telux::loc::DrCalibrationStatusType

#### Enumerator

**DR\_ROLL\_CALIBRATION\_NEEDED** Indicate that roll calibration is needed. Need to take more turns on level ground  
**DR\_PITCH\_CALIBRATION\_NEEDED** Indicate that pitch calibration is needed. Need to take more turns on level ground  
**DR\_YAW\_CALIBRATION\_NEEDED** Indicate that yaw calibration is needed. Need to accelerate in a straight line  
**DR\_ODO\_CALIBRATION\_NEEDED** Indicate that odo calibration is needed. Need to accelerate in a straight line  
**DR\_GYRO\_CALIBRATION\_NEEDED** Indicate that gyro calibration is needed. Need to take more turns on level ground

### 5.7.2.23 enum telux::loc::LocReqEngineType

Specifies the type of engine requested for fixes

#### Enumerator

**LOC\_REQ\_ENGINE\_FUSED\_BIT** Indicate that the fused/default position is needed to be reported back for the tracking sessions. The default position is the propagated/aggregated reports from all engines running on the system (e.g.: DR/SPE/PPE) according to QTI algorithm.  
**LOC\_REQ\_ENGINE\_SPE\_BIT** Indicate that the unmodified SPE position is needed to be reported back for the tracking sessions.  
**LOC\_REQ\_ENGINE\_PPE\_BIT** Indicate that the unmodified PPE position is needed to be reported back for the tracking sessions.  
**LOC\_REQ\_ENGINE\_VPE\_BIT** Indicate that the unmodified VPE position is needed to be reported back for the tracking sessions.

### 5.7.2.24 enum telux::loc::LocationAggregationType

Specifies the type of engine for the reported fixes

#### Enumerator

**LOC\_OUTPUT\_ENGINE\_FUSED** This is the propagated/aggregated report from the fixes of all engines running on the system (e.g.: DR/SPE/PPE).  
**LOC\_OUTPUT\_ENGINE\_SPE** This fix is the unmodified fix from modem GNSS engine  
**LOC\_OUTPUT\_ENGINE\_PPE** This is the unmodified fix from PPP engine  
**LOC\_OUTPUT\_ENGINE\_VPE** This is the unmodified fix from VPE engine.



### 5.7.2.25 enum telux::loc::PositioningEngineType

Specifies the type of engine responsible for fixes when the engine type is fused

#### Enumerator

**STANDARD\_POSITIONING\_ENGINE**  
**DEAD\_RECKONING\_ENGINE**  
**PRECISE\_POSITIONING\_ENGINE** For precise position engines.  
**VP\_POSITIONING\_ENGINE** For VP position engine.

### 5.7.2.26 enum telux::loc::LeverArmType

Lever ARM type

#### Enumerator

**LEVER\_ARM\_TYPE\_GNSS\_TO\_VRP** Lever arm parameters regarding the VRP (Vehicle Reference Point) w.r.t the origin (at the GNSS Antenna)  
**LEVER\_ARM\_TYPE\_DR\_IMU\_TO\_GNSS** Lever arm regarding GNSS Antenna w.r.t the origin at the IMU (inertial measurement unit) for DR (dead reckoning engine)  
**LEVER\_ARM\_TYPE\_VEPP\_IMU\_TO\_GNSS** Lever arm regarding GNSS Antenna w.r.t the origin at the IMU (inertial measurement unit) for VEPP (vision enhanced precise positioning engine)

### 5.7.2.27 enum telux::loc::GnssMeasurementsDataValidityType

Specify valid fields in [GnssMeasurementsData](#).

#### Enumerator

**SV\_ID\_BIT** Validity of svId.  
**SV\_TYPE\_BIT** Validity of svType.  
**STATE\_BIT** Validity of stateMask.  
**RECEIVED\_SV\_TIME\_BIT** Validity of receivedSvTimeNs.  
**RECEIVED\_SV\_TIME\_UNCERTAINTY\_BIT** Validity of receivedSvTimeUncertaintyNs.  
**CARRIER\_TO\_NOISE\_BIT** Validity of carrierToNoiseDbHz.  
**PSEUDORANGE\_RATE\_BIT** Validity of pseudorangeRateMps.  
**PSEUDORANGE\_RATE\_UNCERTAINTY\_BIT** Validity of pseudorangeRateUncertaintyMps.  
**ADR\_STATE\_BIT** Validity of adrStateMask.  
**ADR\_BIT** Validity of adrMeters.  
**ADR\_UNCERTAINTY\_BIT** Validity of adrUncertaintyMeters.  
**CARRIER\_FREQUENCY\_BIT** Validity of carrierFrequencyHz.  
**CARRIER\_CYCLES\_BIT** Validity of carrierCycles.  
**CARRIER\_PHASE\_BIT** Validity of carrierPhase.  
**CARRIER\_PHASE\_UNCERTAINTY\_BIT** Validity of carrierPhaseUncertainty.  
**MULTIPATH\_INDICATOR\_BIT** Validity of multipathIndicator.  
**SIGNAL\_TO\_NOISE\_RATIO\_BIT** Validity of signalToNoiseRatioDb.  
**AUTOMATIC\_GAIN\_CONTROL\_BIT** Validity of agcLevelDb.  
**GNSS\_SIGNAL\_TYPE** Validity of signal type.  
**BASEBAND\_CARRIER\_TO\_NOISE** Validity of basebandCarrierToNoise.  
**FULL\_ISB** Validity of fullInterSignalBias.

**FULL\_ISB\_UNCERTAINTY** Validity of fullInterSignalBiasUncertainty.

### 5.7.2.28 enum telux::loc::GnssMeasurementsStateValidityType

Specify GNSS measurement state in [GnssMeasurementsData::stateMask](#).

#### Enumerator

**UNKNOWN\_BIT** State is unknown.  
**CODE\_LOCK\_BIT** State is "code lock".  
**BIT\_SYNC\_BIT** State is "bit sync".  
**SUBFRAME\_SYNC\_BIT** State is "subframe sync".  
**TOW\_DECODED\_BIT** State is "tow decoded".  
**MSEC\_AMBIGUOUS\_BIT** State is "msec ambiguous".  
**SYMBOL\_SYNC\_BIT** State is "symbol sync".  
**GLO\_STRING\_SYNC\_BIT** State is "GLONASS string sync".  
**GLO\_TOD\_DECODED\_BIT** State is "GLONASS TOD decoded".  
**BDS\_D2\_BIT\_SYNC\_BIT** State is "BDS D2 bit sync".  
**BDS\_D2\_SUBFRAME\_SYNC\_BIT** State is "BDS D2 subframe sync".  
**GAL\_E1BC\_CODE\_LOCK\_BIT** State is "Galileo E1BC code lock".  
**GAL\_E1C\_2ND\_CODE\_LOCK\_BIT** State is "Galileo E1C second code lock".  
**GAL\_E1B\_PAGE\_SYNC\_BIT** State is "Galileo E1B page sync".  
**SBAS\_SYNC\_BIT** State is "SBAS sync".

### 5.7.2.29 enum telux::loc::GnssMeasurementsAdrStateValidityType

Specify accumulated delta range state in [GnssMeasurementsData::adrStateMask](#).

#### Enumerator

**UNKNOWN\_STATE** State is unknown.  
**VALID\_BIT** State is valid.  
**RESET\_BIT** State is "reset".  
**CYCLE\_SLIP\_BIT** State is "cycle slip".

### 5.7.2.30 enum telux::loc::GnssMeasurementsMultipathIndicator

Specify the GNSS multipath indicator state in [GnssMeasurementsData::multipathIndicator](#).

#### Enumerator

**UNKNOWN\_INDICATOR** Multipath indicator is unknown.  
**PRESENT** Multipath indicator is present.  
**NOT\_PRESENT** Multipath indicator is not present.

### 5.7.2.31 enum telux::loc::GnssMeasurementsClockValidityType

Specify the valid fields in [GnssMeasurementsClock](#).

**Enumerator**

**LEAP\_SECOND\_BIT** Validity of leapSecond.  
**TIME\_BIT** Validity of timeNs.  
**TIME\_UNCERTAINTY\_BIT** Validity of timeUncertaintyNs.  
**FULL\_BIAS\_BIT** Validity of fullBiasNs.  
**BIAS\_BIT** Validity of biasNs.  
**BIAS\_UNCERTAINTY\_BIT** Validity of biasUncertaintyNs.  
**DRIFT\_BIT** Validity of driftNsps.  
**DRIFT\_UNCERTAINTY\_BIT** Validity of driftUncertaintyNsps.  
**HW\_CLOCK\_DISCONTINUITY\_COUNT\_BIT** Validity of hwClockDiscontinuityCount.

**5.7.2.32 enum telux::loc::LeapSecondInfoValidityType**

Specify the valid fields in [LeapSecondInfo](#)

**Enumerator**

**LEAP\_SECOND\_SYS\_INFO\_CURRENT\_LEAP\_SECONDS\_BIT** Current leap second info is available. This info will only be available if the leap second change info is not available. If leap second change info is available, to figure out the current leap second info, compare current gps time with the gps timestamp of leap second change to know whether to choose leapSecondBefore or leapSecondAfter as current leap second.  
**LEAP\_SECOND\_SYS\_INFO\_LEAP\_SECOND\_CHANGE\_BIT** The last known leap change event is available. The info can be available on two scenarios: 1: This leap second change event has been scheduled and yet to happen 2: This leap second change event has already happened and next leap second change event has not yet been scheduled.

**5.7.2.33 enum telux::loc::LocationSystemInfoValidityType**

Specify the set of valid fields in [LocationSystemInfo](#)

**Enumerator**

**LOCATION\_SYS\_INFO\_LEAP\_SECOND** contains current leap second or leap second change info

**5.7.2.34 enum telux::loc::GnssEnergyConsumedInfoValidityType**

Specify the valid fields in [GnssEnergyConsumedInfo](#).

**Enumerator**

**ENERGY\_CONSUMED\_SINCE\_FIRST\_BOOT\_BIT** validity of [GnssEnergyConsumedInfo](#)

**5.7.2.35 enum telux::loc::AidingDataType**

Specifies the set of aiding data. This is referenced in the deleteAidingData for deleting any aiding data.

**Enumerator**

**AIDING\_DATA\_EPHEMERIS** Mask to delete ephemeris aiding data

**AIDING\_DATA\_DR\_SENSOR\_CALIBRATION** Mask to delete calibration data from dead reckoning position engine

**5.7.2.36 enum telux::loc::TerrestrialTechnologyType**

Specifies the set of terrestrial technologies.

**Enumerator**

**GTP\_WWAN** Cell-based technology

**5.7.2.37 enum telux::loc::RobustLocationConfigType**

Specify the valid mask for robust location configuration used by the GNSS standard position engine (SPE).

**Enumerator**

**VALID\_ENABLED** Validity of enabled

**VALID\_ENABLED\_FOR\_E911** Validity of enabledForE911.

**VALID\_VERSION** Validity of version.

**5.7.2.38 enum telux::loc::NmeaSentenceType**

Specifies the HLOS generated NMEA sentence types.

**Enumerator**

**GGA** GGA NMEA sentence

**RMC** RMC NMEA sentence

**GSA** GSA NMEA sentence

**VTG** VTG NMEA sentence

**GNS** GNS NMEA sentence

**DTM** DTM NMEA sentence

**GPGSV** GPGSV NMEA sentence for SVs from GPS constellation

**GLGSV** GLGSV NMEA sentence for SVs from GLONASS constellation

**GAGSV** GAGSV NMEA sentence for SVs from GALILEO constellation

**GQGSV** GQGSV NMEA sentence for SVs from QZSS constellation

**GBGSV** GBGSV NMEA sentence for SVs from BEIDOU constellation

**GIGSV** GIGSV NMEA sentence for SVs from NAVIC constellation

**ALL** All NMEA sentences

**5.7.2.39 enum telux::loc::GeodeticDatumType [strong]**

Specify the Geodetic datum for NMEA sentence types that are generated.

**Enumerator**

**GEODETIC\_TYPE\_NONE** No type

**GEODETIC\_TYPE\_WGS\_84** Geodetic datum type to indicate the use of World Geodetic System 1984 (WGS84) system

**GEODETIC\_TYPE\_PZ\_90** Geodetic datum type to indicate the use of PZ90/GLONASS system

**5.7.2.40 enum telux::loc::DRConfigValidityType**

Specify the valid mask for the configuration parameters of dead reckoning position engine

**Enumerator**

**BODY\_TO\_SENSOR\_MOUNT\_PARAMS\_VALID** Validity of body to sensor mount parameters.

**VEHICLE\_SPEED\_SCALE\_FACTOR\_VALID** Validity of vehicle speed scale factor.

**VEHICLE\_SPEED\_SCALE\_FACTOR\_UNC\_VALID** Validity of vehicle speed scale factor uncertainty.

**GYRO\_SCALE\_FACTOR\_VALID** Validity of gyro scale factor.

**GYRO\_SCALE\_FACTOR\_UNC\_VALID** Validity of gyro scale factor uncertainty.

**5.7.2.41 enum telux::loc::GnssReportType**

Specifies the set of gnss reports.

**Enumerator**

**LOCATION** Location reports

**SATELLITE\_VEHICLE** Satellite reports

**NMEA** Nmea reports

**DATA** Data reports

**MEASUREMENT** 1Hz measurement reports

**5.7.2.42 enum telux::loc::EngineType [strong]**

Specify the position engine types

**Enumerator**

**UNKNOWN** Unknown engine type.

**SPE** Standard GNSS position engine.

**PPE** Precise position engine.

**DRE** Dead reckoning position engine.

**VPE** Vision positioning engine.

**5.7.2.43 enum telux::loc::ReportStatus [strong]**

Specify the status of the report

**Enumerator**

**UNKNOWN** Report status is unknown.

**SUCCESS** Report status is successful. The engine is able to calculate the desired fix. Most of the fields in [ILocationInfoEx](#) will be valid.

**INTERMEDIATE** Report is still in progress. The engine has not completed its calculations when this report was generated. Accuracy of various fields is non-optimal. Only some of the fields in [ILocationInfoEx](#) will be valid.

**FAILURE** Report status has failed. The engine is not able to calculate the fix. Most of the fields in [ILocationInfoEx](#) will be invalid.

#### 5.7.2.44 enum telux::loc::DebugLogLevel [strong]

Specify the logcat debug level during XTRA's param configuration. Currently, only XTRA daemon will support the runtime configuration of the debug log level.

##### Enumerator

**DEBUG\_LOG\_LEVEL\_NONE** No message is logged.

**DEBUG\_LOG\_LEVEL\_ERROR** Only error level debug messages will get logged.

**DEBUG\_LOG\_LEVEL\_WARNING** Only warning and error level debug messages will get logged.

**DEBUG\_LOG\_LEVEL\_INFO** Only info, warning and error level debug messages will get logged.

**DEBUG\_LOG\_LEVEL\_DEBUG** Only debug, info, warning and error level debug messages will get logged.

**DEBUG\_LOG\_LEVEL\_VERBOSE** Verbose, debug, info, warning and error level debug messages will get logged.

#### 5.7.2.45 enum telux::loc::XtraDataStatus [strong]

Provides the status of the previously downloaded Xtra data.

##### Enumerator

**STATUS\_UNKNOWN** If XTRA feature is disabled or if XTRA feature is enabled, but XTRA daemon has not yet retrieved the assistance data status from modem on early stage of device bootup, xtra data status will be unknown.

**STATUS\_NOT\_AVAIL** If XTRA feature is enabled, but XTRA data is not present on the device.

**STATUS\_NOT\_VALID** If XTRA feature is enabled, XTRA data has been downloaded ever but no longer valid.

**STATUS\_VALID** If XTRA feature is enabled, XTRA data has been downloaded and is currently valid.

#### 5.7.2.46 enum telux::loc::LocConfigIndicationsType

Enum of all the possible indications invoked by a Location Configurator listener.

##### Enumerator

**LOC\_CONF\_IND\_XTRA\_STATUS**

### 5.7.3 Variable Documentation

#### 5.7.3.1 const float telux::loc::UNKNOWN\_CARRIER\_FREQ = -1

**5.7.3.2 const int telux::loc::UNKNOWN\_SIGNAL\_MASK = 0**

**5.7.3.3 const uint64\_t telux::loc::UNKNOWN\_TIMESTAMP = 0**

**5.7.3.4 const float telux::loc::DEFAULT\_TUNC\_THRESHOLD = 0.0**

Default value for threshold of time uncertainty. Units: milli-seconds.

**5.7.3.5 const int telux::loc::DEFAULT\_TUNC\_ENERGY\_THRESHOLD = 0**

Default value for energy consumed of time uncertainty. The default here means that the engine is allowed to use infinite power. Units: 100 micro watt second.

**5.7.3.6 const uint32\_t telux::loc::DEFAULT\_GNSS\_REPORT = 0xffffffff**

0xffffffff indicates all the reports are enabled.

## 5.8 Data Services

This section contains APIs related to Cellular Data Services.

### 5.8.1 Define Documentation

#### 5.8.1.1 #define PROFILE\_ID\_MAX 0x7FFFFFFF

Default data profile id.

#### 5.8.1.2 #define MAX\_QOS\_FILTERS 16

Max filters in one flow

#### 5.8.1.3 #define IP\_PROT\_UNKNOWN 0xFF

Default IP Protocol number in IPv4 or IPv6 headers.

### 5.8.2 Data Structure Documentation

#### 5.8.2.1 class telux::data::IDataConnectionManager

[IDataConnectionManager](#) is a primary interface for cellular connectivity. This interface provides APIs for start and stop data call connections, get data call information and listener for monitoring data calls. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready.

##### Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status](#) [setDefaultProfile](#) ([OperationType](#) oprType, uint8\_t profileId, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [startDataCall](#) (int profileId, [IpFamilyType](#) ipFamilyType=[IpFamilyType::IPV4V6](#), [DataCallResponseCb](#) callback=nullptr, [OperationType](#) operationType=[OperationType::DATA\\_LOCAL](#), std::string interfaceName="")=0
- virtual [telux::common::Status](#) [stopDataCall](#) (int profileId, [IpFamilyType](#) ipFamilyType=[IpFamilyType::IPV4V6](#), [DataCallResponseCb](#) callback=nullptr, [OperationType](#) operationType=[OperationType::DATA\\_LOCAL](#))=0
- virtual [telux::common::Status](#) [stopDataCall](#) (std::string interfaceName, [telux::common::ResponseCallback](#) callback=nullptr, [OperationType](#) operationType=[OperationType::DATA\\_LOCAL](#))=0
- virtual [telux::common::Status](#) [registerListener](#) (std::weak\_ptr< [IDataConnectionListener](#) > listener)=0
- virtual [telux::common::Status](#) [deregisterListener](#) (std::weak\_ptr< [IDataConnectionListener](#) > listener)=0



- virtual int `getSlotId ()=0`
- virtual `telux::common::Status requestDataCallList (OperationType type, DataCallListResponseCb callback)=0`
- virtual `~IDataConnectionManager ()`

### 5.8.2.1.1 Constructors and Destructors

5.8.2.1.1.1 virtual `telux::data::IDataConnectionManager::~~IDataConnectionManager ( ) [virtual]`

Destructor for `IDataConnectionManager`

### 5.8.2.1.2 Member Function Documentation

5.8.2.1.2.1 virtual `bool telux::data::IDataConnectionManager::isSubsystemReady ( ) [pure virtual]`

Checks if the data subsystem is ready.

#### Returns

True if Data Connection Manager is ready for service, otherwise returns false.

5.8.2.1.2.2 virtual `std::future<bool> telux::data::IDataConnectionManager::onSubsystemReady ( ) [pure virtual]`

Wait for data subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when card manager is ready.

5.8.2.1.2.3 virtual `telux::common::Status telux::data::IDataConnectionManager::setDefaultProfile ( OperationType oprType, uint8_t profileId, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Set a profile as default which results in the following:

- Default routes in the system will route traffic over the network interface associated with this profile.
- Bridge 0 will be associated with traffic from this profile.

#### Parameters

in	<i>operationType</i>	<code>telux::data::OperationType</code>
in	<i>profileId</i>	Profile identifier to be associated with default handler
in	<i>callback</i>	optional callback to get the response setDefaultProfile

**Returns**

Immediate status of `setDefaultProfile` i.e. success or suitable status.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.8.2.1.2.4** `virtual telux::common::Status telux::data::IDataConnectionManager::startDataCall ( int profileId, IpFamilyType ipFamilyType = IpFamilyType::IPV4V6, DataCallResponseCb callback = nullptr, OperationType operationType = OperationType::DATA_LOCAL, std::string interfaceName = "" ) [pure virtual]`

Starts a data call corresponding to default or specified profile identifier.

This will bring up data call connection based on specified profile identifier, IP family type, and operation type (local/remote). Clients can also specify network interface name that will be associated with data call to be started. This is an asynchronous API. If `telux::common::Status::SUCCESS` is returned, client provided callback will be invoked at later time with error code and DataCall object associated with requested call. Clients might receive additional notification for the final data call status. For details see `telux::data::DataCallResponseCb`.

**Note**

if application starts data call on IPV4V6 then it's expected to stop the data call on same ip family type (i.e IPV4V6).

**Parameters**

in	<i>profileId</i>	Profile identifier corresponding to which data call bring up will be done. Use <code>IDataProfileManager::requestProfileList</code> to get list of available profiles.
in	<i>ipFamilyType</i>	Identifies IP family type
out	<i>callback</i>	Optional callback to get the response of start data call.
in	<i>operationType</i>	Optional <code>telux::data::OperationType</code>
in	<i>interfaceName</i>	Optional name to be assigned to the network interface representing this data call. This option is not supported on all platforms. See documentation of <code>telux::data::DataCallResponseCb</code> for error returned on such platforms.

**Returns**

Immediate status of `startDataCall()` request sent i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.8.2.1.2.5** `virtual telux::common::Status telux::data::IDataConnectionManager::stopDataCall ( int profileId, IpFamilyType ipFamilyType = IpFamilyType::IPV4V6, DataCallResponseCb callback = nullptr, OperationType operationType = OperationType::DATA_LOCAL ) [pure virtual]`

Stops a data call corresponding to default or specified profile identifier.

This will tear down specific data call connection based on profile identifier.

#### Note

If application starts data call on IPV4V6 then it's expected to stop the data call on same ip family type (i.e IPV4V6). Client can only stop data call it started.

#### Parameters

in	<i>profileId</i>	Profile identifier corresponding to which data call tear down will be done. Use data profile manager to get the list of available profiles.
in	<i>ipFamilyType</i>	Identifies IP family type
out	<i>callback</i>	Optional callback to get the response of stop data call
in	<i>operationType</i>	Optional <a href="#">telux::data::OperationType</a>

#### Returns

Immediate status of [stopDataCall\(\)](#) request sent i.e. success or suitable status code. The client receives asynchronous notifications indicating the data call tear-down.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.8.2.1.2.6** `virtual telux::common::Status telux::data::IDataConnectionManager::stopDataCall ( std::string interfaceName, telux::common::ResponseCallback callback = nullptr, OperationType operationType = OperationType::DATA_LOCAL ) [pure virtual]`

Tear down data call connection based on network interface name associated with data call. This is an asynchronous API. If [telux::common::Status::SUCCESS](#) is returned, client provided callback will be invoked at later time with error code and DataCall object associated with requested call. Clients might receive additional notification for the final data call status. For details see [telux::data::DataCallResponseCb](#).

#### Note

This API will tear down data calls accross all IP families associated with the provided network interface. If application starts data call on IPV4V6 then it's expected to stop both IPv4 and IPv6 ip families when this API is called.

#### Parameters

in	<i>interfaceName</i>	network interface name associated with data call to be torn down
----	----------------------	--

out	<i>callback</i>	Optional callback to get the response of stop data call
in	<i>operationType</i>	Optional <a href="#">telux::data::OperationType</a>

**Returns**

Immediate status of [stopDataCall\(\)](#) request sent i.e. success or suitable status code. The client receives asynchronous notifications indicating the data call tear-down.

#### 5.8.2.1.2.7 virtual `telux::common::Status telux::data::IDataConnectionManager::registerListener ( std::weak_ptr< IDataConnectionListener > listener ) [pure virtual]`

Register a listener for specific events in the Connection Manager like establishment of new data call, data call info change and call failure.

**Parameters**

in	<i>listener</i>	pointer of <a href="#">IDataConnectionListener</a> object that processes the notification
----	-----------------	---

**Returns**

Status of registerListener success or suitable status code

#### 5.8.2.1.2.8 virtual `telux::common::Status telux::data::IDataConnectionManager::deregisterListener ( std::weak_ptr< IDataConnectionListener > listener ) [pure virtual]`

Removes a previously added listener.

**Parameters**

in	<i>listener</i>	pointer of <a href="#">IDataConnectionListener</a> object that needs to be removed
----	-----------------	--

**Returns**

Status of deregisterListener success or suitable status code

#### 5.8.2.1.2.9 virtual `int telux::data::IDataConnectionManager::getSlotId ( ) [pure virtual]`

Get associated slot id for the Data Connection Manager.

**Returns**

SlotId

### 5.8.2.1.2.10 virtual telux::common::Status telux::data::IDataConnectionManager::requestDataCallList ( OperationType type, DataCallListResponseCb callback ) [pure virtual]

Request list of data calls available in the system

#### Parameters

out	<i>OperationType</i>	<a href="#">telux::data::OperationType</a>
out	<i>callback</i>	Callback with list of supported data calls

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.8.2.2 class telux::data::IDataCall

Represents single established data call on the device.

#### Public member functions

- virtual const std::string & [getInterfaceName](#) ()=0
- virtual [DataBearerTechnology](#) [getCurrentBearerTech](#) ()=0
- virtual [DataCallEndReason](#) [getDataCallEndReason](#) ()=0
- virtual [DataCallStatus](#) [getDataCallStatus](#) ()=0
- virtual [IpFamilyInfo](#) [getIpv4Info](#) ()=0
- virtual [IpFamilyInfo](#) [getIpv6Info](#) ()=0
- virtual [TechPreference](#) [getTechPreference](#) ()=0
- virtual std::list< [IpAddrInfo](#) > [getIpAddressInfo](#) ()=0
- virtual [IpFamilyType](#) [getIpFamilyType](#) ()=0
- virtual int [getProfileId](#) ()=0
- virtual [OperationType](#) [getOperationType](#) ()=0
- virtual telux::common::Status [requestTrafficFlowTemplate](#) ([IpFamilyType](#) ipFamilyType, [TrafficFlowTemplateCb](#) callback)=0
- virtual telux::common::Status [requestDataCallStatistics](#) ([StatisticsResponseCb](#) callback=nullptr)=0
- virtual telux::common::Status [resetDataCallStatistics](#) (telux::common::ResponseCallback callback=nullptr)=0
- virtual [~IDataCall](#) ()

### 5.8.2.2.1 Constructors and Destructors

#### 5.8.2.2.1.1 virtual telux::data::IDataCall::~IDataCall ( ) [virtual]

Destructor for [IDataCall](#)

### 5.8.2.2.2 Member Function Documentation

#### 5.8.2.2.2.1 virtual const std::string& telux::data::IDataCall::getInterfaceName ( ) [pure virtual]

Get interface name for the data call associated.

##### Returns

Interface Name.

#### 5.8.2.2.2.2 virtual DataBearerTechnology telux::data::IDataCall::getCurrentBearerTech ( ) [pure virtual]

Get the bearer technology on which earlier data call was brought up like LTE, WCDMA and etc. This is synchronous API called by client to get bearer technology corresponding to data call.

##### Returns

[DataBearerTechnology](#)

#### 5.8.2.2.2.3 virtual DataCallEndReason telux::data::IDataCall::getDataCallEndReason ( ) [pure virtual]

Get failure reason for the data call.

##### Returns

DataCallFailReason.

#### 5.8.2.2.2.4 virtual DataCallStatus telux::data::IDataCall::getDataCallStatus ( ) [pure virtual]

Get data call status like connected, disconnected and IP address changes.

##### Returns

[DataCallStatus](#).

**5.8.2.2.2.5 virtual IpFamilyInfo telux::data::IDataCall::getIpv4Info ( ) [pure virtual]**

Get IPv4 Family info like connected, disconnected and IP address changes.

**Returns**

[IpFamilyInfo](#).

**5.8.2.2.2.6 virtual IpFamilyInfo telux::data::IDataCall::getIpv6Info ( ) [pure virtual]**

Get IPv6 Family info like connected, disconnected and IP address changes.

**Returns**

[IpFamilyInfo](#).

**5.8.2.2.2.7 virtual TechPreference telux::data::IDataCall::getTechPreference ( ) [pure virtual]**

Get the technology on which the call was brought up.

**Returns**

[TechPreference](#).

**5.8.2.2.2.8 virtual std::list<IpAddrInfo> telux::data::IDataCall::getIpAddressInfo ( ) [pure virtual]**

Get list of IP address information.

**Returns**

List of IP address details.

**5.8.2.2.2.9 virtual IpFamilyType telux::data::IDataCall::getIpFamilyType ( ) [pure virtual]**

Get IP Family Type i.e. IPv4, IPv6 or Both

**Returns**

[IpFamilyType](#).

**5.8.2.2.2.10 virtual int telux::data::IDataCall::getProfileId ( ) [pure virtual]**

Get Profile Id

**Returns**

Profile Identifier.

**5.8.2.2.2.11 virtual OperationType telux::data::IDataCall::getOperationType ( ) [pure virtual]**

Get data operation used for the DataCall.

**Returns**

[OperationType](#)

**5.8.2.2.2.12 virtual telux::common::Status telux::data::IDataCall::requestTrafficFlowTemplate ( IpFamilyType ipFamilyType, TrafficFlowTemplateCb callback ) [pure virtual]**

Get the current installed QOS Traffic flow template information.

**Parameters**

in	<i>ipFamilyType</i>	- IP Family type <a href="#">IpFamilyType</a> . TFT's are installed per IP Family.
in	<i>callback</i>	- callback function to get the result of API.

**Returns**

Status of requestTrafficFlowTemplate i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.8.2.2.2.13 virtual telux::common::Status telux::data::IDataCall::requestDataCallStatistics ( StatisticsResponseCb callback = nullptr ) [pure virtual]**

Request the data transfer statistics for data call corresponding to specified profile identifier.

**Parameters**

in	<i>callback</i>	Optional callback to get the response of request Data Call Statistics
----	-----------------	---

**Returns**

Status of getDataCallStatistics i.e. success or suitable status code.

**5.8.2.2.2.14 virtual telux::common::Status telux::data::IDataCall::resetDataCallStatistics ( telux::common::ResponseCallback callback = nullptr ) [pure virtual]**

Reset data transfer statistics for data call corresponding to specified profile identifier.



in	<i>callback</i>	optional callback to get the response of reset Data call statistics
----	-----------------	---

## Returns

Status of resetDataCallStatistics i.e. success or suitable status code.

### 5.8.2.3 class telux::data::IDataConnectionListener

Interface for Data connection listener object. Client needs to implement this interface to get access to data services notifications like onNewDataCall, onDataCallStatusChanged and onDataCallFailure.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

#### Public member functions

- virtual void [onDataCallInfoChanged](#) (const std::shared\_ptr< [IDataCall](#) > &dataCall)
- virtual void [onTrafficFlowTemplateChange](#) (const std::shared\_ptr< [IDataCall](#) > &dataCall, const std::vector< std::shared\_ptr< [TftChangeInfo](#) >> &tft)
- virtual void [onHwAccelerationChanged](#) (const [ServiceState](#) state)
- virtual [~IDataConnectionListener](#) ()

#### 5.8.2.3.1 Constructors and Destructors

**5.8.2.3.1.1 virtual telux::data::IDataConnectionListener::~~IDataConnectionListener ( ) [virtual]**

Destructor for [IDataConnectionListener](#)

#### 5.8.2.3.2 Member Function Documentation

**5.8.2.3.2.1 virtual void telux::data::IDataConnectionListener::onDataCallInfoChanged ( const std::shared\_ptr< [IDataCall](#) > & *dataCall* ) [virtual]**

This function is called when there is a change in the data call.

#### Parameters

in	<i>dataCall</i>	Pointer to <a href="#">IDataCall</a>
----	-----------------	--------------------------------------

**5.8.2.3.2.2 virtual void telux::data::IDataConnectionListener::onTrafficFlowTemplateChange ( const std::shared\_ptr< [IDataCall](#) > & *dataCall*, const std::vector< std::shared\_ptr< [TftChangeInfo](#) >> & *tft* ) [virtual]**

This function is called when the TFT's parameters are changed for a packet data session.

in	<i>dataCall</i>	Pointer to <a href="#">IDataCall</a>
in	<i>tft</i>	vector of <a href="#">TftChangeInfo</a> info <a href="#">TftChangeInfo</a>

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.8.2.3.2.3 virtual void telux::data::IDataConnectionListener::onHwAccelerationChanged ( const ServiceState *state* ) [virtual]

This function is called when a change occur in hardware acceleration service.

**Parameters**

in	<i>state</i>	New state of hardware Acceleration service (Active/Inactive)
----	--------------	--

#### 5.8.2.4 struct telux::data::DataRestrictMode

Defines the supported powersave filtering mode and autoexit for the packet data session. DataRestrictFilter

**Data fields**

Type	Field	Description
<a href="#">DataRestrictModeType</a>	filterMode	Disable or enable data filter mode. When disabled all the data packets will be forwarded from modem to the apps. When enabled only the data matching the filters will be forwarded from modem to the apps.
<a href="#">DataRestrictModeType</a>	filterAutoExit	Disable or enable autoexit feature. When enabled, once an incoming packet matching the filter is received, filter mode will be disabled automatically and any packet will be allowed to be forwarded from modem to apps.

#### 5.8.2.5 struct telux::data::PortInfo

Used to define the Port number and range (number of ports following port value) Ex- for ports ranging from 1000-3000 port = 1000 and range= 2000

for single port 5000 port = 5000 and range= 0

**Data fields**

Type	Field	Description
uint16_t	port	Port.
uint16_t	range	Range.

### 5.8.2.6 struct telux::data::ProfileParams

Profile Parameters used for profile creation, query and modify

#### Data fields

Type	Field	Description
string	profileName	Profile Name
string	apn	APN name
string	userName	APN user name (if any)
string	password	APN password (if any)
TechPreference	techPref	Technology preference, default is TechPreference::UNKNOWN
AuthProtocol↔ Type	authType	Authentication protocol type, default is AuthProtocolType::AUTH_NONE
IpFamilyType	ipFamilyType	Preferred IP family for the call, default is IpFamilyType::UNKNOWN

### 5.8.2.7 struct telux::data::DataCallStats

Data transfer statistics structure.

#### Data fields

Type	Field	Description
uint64_t	packetsTx	Number of packets transmitted
uint64_t	packetsRx	Number of packets received
uint64_t	bytesTx	Number of bytes transmitted
uint64_t	bytesRx	Number of bytes received
uint64_t	packets↔ DroppedTx	Number of transmit packets dropped
uint64_t	packets↔ DroppedRx	Number of receive packets dropped

### 5.8.2.8 struct telux::data::IpAddrInfo

IP address information structure

#### Data fields

Type	Field	Description
string	ifAddress	Interface IP address.
unsigned int	ifMask	Subnet mask.
string	gwAddress	Gateway IP address.
unsigned int	gwMask	Subnet mask.
string	primaryDns↔ Address	Primary DNS address.
string	secondary↔ DnsAddress	Secondary DNS address.

### 5.8.2.9 struct telux::data::DataCallEndReason

Structure represents data call failure reason type and code.

#### Data fields

Type	Field	Description
<a href="#">EndReason</a> ↔ <a href="#">Type</a>	type	Data call terminated due to reason type, default is CE_UNKNOWN
<a href="#">union Data</a> ↔ <a href="#">CallEndReason</a>	<a href="#">__unnamed_</a> ↔ _	

### 5.8.2.10 struct telux::data::LatencyConfig

Latency configuration

#### Data fields

Type	Field	Description
<a href="#">LatencyLevel</a>	uplink	

### 5.8.2.11 struct telux::data::BackhaulInfo

Encapsulate backhaul configuration parameters

#### Data fields

Type	Field	Description
<a href="#">BackhaulType</a>	backhaul	
SlotId	slotId	Backhaul type to apply configuration on.
int	profileId	Slot Id which has sim that contains profile id Needed only for WWAN backhaul

### 5.8.2.12 struct telux::data::VlanConfig

Structure for vlan configuration

#### Data fields

Type	Field	Description
<a href="#">InterfaceType</a>	iface	PHY interfaces (i.e. ETH, ECM and RNDIS)
int16_t	vlanId	Vlan identifier (i.e 1-4094)
bool	isAccelerated	is acceleration allowed

### 5.8.2.13 struct telux::data::FlowDataRate

QOS Flow data min max rate bits per seconds

**Data fields**

Type	Field	Description
uint64_t	maxRate	QOS Flow maximum data rate
uint64_t	minRate	QOS Flow minimum data rate

**5.8.2.14 struct telux::data::QosIPFlowInfo**

QOS Flow IP info

**Data fields**

Type	Field	Description
<a href="#">QosIPFlow</a> ↔ <a href="#">Mask</a>	mask	Valid parameters of <a href="#">QosIPFlowInfo</a> ref
<a href="#">IpTraffic</a> ↔ <a href="#">ClassType</a>	tfClass	IP Traffic class type <a href="#">IpFamilyType</a>
<a href="#">FlowDataRate</a>	dataRate	Flow data rate <a href="#">FlowDataRate</a>

**5.8.2.15 class telux::data::DataFactory**

[DataFactory](#) is the central factory to create all data classes.

**Public member functions**

- std::shared\_ptr< [IDataConnectionManager](#) > [getDataConnectionManager](#) (int slotId=DEFAULT\_SLOT\_ID)
- std::shared\_ptr< [IDataProfileManager](#) > [getDataProfileManager](#) (int slotId=DEFAULT\_SLOT\_ID)
- std::shared\_ptr< [IDataFilterManager](#) > [getDataFilterManager](#) (int slotId=DEFAULT\_SLOT\_ID)
- std::shared\_ptr< [IServingSystemManager](#) > [getServingSystemManager](#) (SlotId slotId=DEFAULT\_SLOT\_ID)
- std::shared\_ptr< [telux::data::net::INatManager](#) > [getNatManager](#) ([telux::data::OperationType](#) oprType)
- std::shared\_ptr< [telux::data::net::IFirewallManager](#) > [getFirewallManager](#) ([telux::data::OperationType](#) oprType)
- std::shared\_ptr< [telux::data::net::IFirewallEntry](#) > [getNewFirewallEntry](#) ([IpProtocol](#) proto, [Direction](#) direction, [IpFamilyType](#) ipFamilyType)
- std::shared\_ptr< [IipFilter](#) > [getNewIpFilter](#) ([IpProtocol](#) proto)
- std::shared\_ptr< [telux::data::net::IVlanManager](#) > [getVlanManager](#) ([telux::data::OperationType](#) oprType)
- std::shared\_ptr< [telux::data::net::ISocksManager](#) > [getSocksManager](#) ([telux::data::OperationType](#) oprType)
- std::shared\_ptr< [telux::data::net::IBridgeManager](#) > [getBridgeManager](#) ()
- std::shared\_ptr< [telux::data::net::IL2tpManager](#) > [getL2tpManager](#) ()

- virtual `std::shared_ptr< telux::data::IDataSettingsManager > getDataSettingsManager (telux::data::OperationType oprType)`

### Static Public Member Functions

- static `DataFactory & getInstance ()`

### 5.8.2.15.1 Member Function Documentation

#### 5.8.2.15.1.1 static `DataFactory& telux::data::DataFactory::getInstance ( ) [static]`

Get Data Factory instance.

#### 5.8.2.15.1.2 `std::shared_ptr<IDataConnectionManager> telux::data::DataFactory::getDataConnectionManager ( int slotId = DEFAULT_SLOT_ID )`

Get Data Connection Manager

#### Parameters

<code>in</code>	<code>slotId</code>	Unique identifier for the SIM slot
-----------------	---------------------	------------------------------------

#### Returns

instance of `IDataConnectionManager`

#### 5.8.2.15.1.3 `std::shared_ptr<IDataProfileManager> telux::data::DataFactory::getDataProfileManager ( int slotId = DEFAULT_SLOT_ID )`

Get Data Profile Manager

#### Parameters

<code>in</code>	<code>slotId</code>	Unique identifier for the SIM slot
-----------------	---------------------	------------------------------------

#### Returns

instance of `IDataProfileManager`

#### 5.8.2.15.1.4 `std::shared_ptr<IDataFilterManager> telux::data::DataFactory::getDataFilterManager ( int slotId = DEFAULT_SLOT_ID )`

Get Data Filter Manager instance

#### Parameters

<code>in</code>	<code>slotId</code>	Unique identifier for the SIM slot
-----------------	---------------------	------------------------------------

**Returns**

instance of [IDataFilterManager](#).

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.8.2.15.1.5 `std::shared_ptr<IServingSystemManager> telux::data::DataFactory::getServingSystemManager ( SlotId slotId = DEFAULT_SLOT_ID )`

Get Serving System Manager

**Parameters**

in	<i>slotId</i>	Unique identifier for the SIM slot
----	---------------	------------------------------------

**Returns**

instance of [IServingSystemManager](#)

#### 5.8.2.15.1.6 `std::shared_ptr<telux::data::net::INatManager> telux::data::DataFactory::getNatManager ( telux::data::OperationType oprType )`

Get Network Address Translation(NAT) Manager

**Parameters**

in	<i>oprType</i>	Required operation type <a href="#">telux::data::OperationType</a>
----	----------------	--

**Returns**

instance of [INatManager](#)

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.8.2.15.1.7 `std::shared_ptr<telux::data::net::IFirewallManager> telux::data::DataFactory::getFirewallManager ( telux::data::OperationType oprType )`

Get Firewall Manager

**Parameters**

in	<i>oprType</i>	Required operation type <a href="#">telux::data::OperationType</a>
----	----------------	--

**Returns**

instance of IFirewallManager

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.8.2.15.1.8** `std::shared_ptr<telux::data::net::IFirewallEntry> telux::data::DataFactory::getNewFirewallEntry ( IpProtocol proto, Direction direction, IpFamilyType ipFamilyType )`

Get Firewall entry based on IP protocol and set respective filter (i.e. TCP or UDP)

**Parameters**

in	<i>proto</i>	<a href="#">telux::data::IpProtocol</a>
in	<i>direction</i>	<a href="#">telux::data::Direction</a>
in	<i>ipFamilyType</i>	Identifies IP family type <a href="#">telux::data::IpFamilyType</a>

**Returns**

instance of IFirewallEntry

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.8.2.15.1.9** `std::shared_ptr<IIPFilter> telux::data::DataFactory::getNewIIPFilter ( IpProtocol proto )`

Get [IIPFilter](#) instance based on IP Protocol, This can be used in Firewall Manager and Data Filter Manager

**Parameters**

in	<i>proto</i>	<a href="#">telux::data::IpProtocol</a> Some sample protocol values are ICMP = 1 # Internet Control Message Protocol - RFC 792 IGMP = 2 # Internet Group Management Protocol - RFC 1112 TCP = 6 # Transmission Control Protocol - RFC 793 UDP = 17 # User Datagram Protocol - RFC 768 ESP = 50 # Encapsulating Security Payload - RFC 4303
----	--------------	--

**Returns**

instance of [IIPFilter](#) based on IpProtocol filter (i.e TCP, UDP)



**Note**

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

#### 5.8.2.15.1.10 **std::shared\_ptr<telux::data::net::IVlanManager> telux::data::DataFactory::getVlanManager ( telux::data::OperationType *oprType* )**

Get VLAN Manager

**Parameters**

in	<i>oprType</i>	Required operation type <a href="#">telux::data::OperationType</a>
----	----------------	--

**Returns**

instance of IVlanManager

**Note**

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

#### 5.8.2.15.1.11 **std::shared\_ptr<telux::data::net::ISocksManager> telux::data::DataFactory::getSocksManager ( telux::data::OperationType *oprType* )**

Get Socks Manager

**Parameters**

in	<i>oprType</i>	Required operation type <a href="#">telux::data::OperationType</a>
----	----------------	--

**Returns**

instance of ISocksManager

**Note**

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

#### 5.8.2.15.1.12 **std::shared\_ptr<telux::data::net::IBridgeManager> telux::data::DataFactory::getBridgeManager ( )**

Get Software Bridge Manager

**Returns**

instance of IBridgeManager

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.8.2.15.1.13 `std::shared_ptr<telux::data::net::IL2tpManager> telux::data::DataFactory::getL2tpManager ( )`

Get L2TP Manager

**Returns**

instance of `IL2tpManager`

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.8.2.15.1.14 `virtual std::shared_ptr<telux::data::IDataSettingsManager> telux::data::DataFactory::getDataSettingsManager ( telux::data::OperationType oprType ) [virtual]`

Gets the data settings manager instance.

**Parameters**

<code>in</code>	<code>oprType</code>	Required operation type <a href="#">telux::data::OperationType</a>
-----------------	----------------------	--

**Returns**

[IDataSettingsManager](#) instance

#### 5.8.2.16 `class telux::data::IDataFilterListener`

Listener class for listening to filtering mode notifications, like Data filtering mode change. Client need to implement these methods. The methods in listener can be invoked from multiple threads. So the client needs to make sure that the implementation is thread-safe.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**Public member functions**

- virtual void [onDataRestrictModeChange](#) ([DataRestrictMode](#) mode)
- virtual [~IDataFilterListener](#) ()

### 5.8.2.16.1 Constructors and Destructors

5.8.2.16.1.1 `virtual telux::data::IDataFilterListener::~IDataFilterListener ( ) [virtual]`

Destructor of [IDataFilterListener](#)

### 5.8.2.16.2 Member Function Documentation

5.8.2.16.2.1 `virtual void telux::data::IDataFilterListener::onDataRestrictModeChange ( DataRestrictMode mode ) [virtual]`

This function is called when the data filtering mode is changed for the packet data session.

#### Parameters

in	<i>state</i>	the current data filter mode
----	--------------	------------------------------

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.8.2.17 class telux::data::IDataFilterManager

[IDataFilterManager](#) class provides interface to enable/disable the data restrict filters and register for data restrict filter. The filtering can be done at any time. One such use case is to do it when we want the AP to suspend so that we are not waking up the AP due to spurious incoming messages. Also to make sure the DataRestrict mode is enabled.

In contrary to when DataRestrict mode is disabled, modem will forward all the incoming data packets to AP and might wake up AP unnecessarily.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### Public member functions

- virtual bool [isReady](#) ()=0
- virtual std::future< bool > [onReady](#) ()=0
- virtual [telux::common::Status registerListener](#) (std::weak\_ptr< [IDataFilterListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak\_ptr< [IDataFilterListener](#) > listener)=0
- virtual [telux::common::Status setDataRestrictMode](#) ([DataRestrictMode](#) mode, [telux::common::ResponseCallback](#) callback=nullptr, int profileId=PROFILE\_ID\_MAX, [IpFamilyType](#) ipFamilyType=[IpFamilyType::UNKNOWN](#))=0
- virtual [telux::common::Status requestDataRestrictMode](#) (std::string ifaceName, [DataRestrictModeCb](#) callback)=0

- virtual `telux::common::Status addDataRestrictFilter` (`std::shared_ptr< IipFilter > &filter`, `telux::common::ResponseCallback callback=nullptr`, `int profileId=PROFILE_ID_MAX`, `IpFamilyType ipFamilyType=IpFamilyType::UNKNOWN`)=0
- virtual `telux::common::Status removeAllDataRestrictFilters` (`telux::common::ResponseCallback callback=nullptr`, `int profileId=PROFILE_ID_MAX`, `IpFamilyType ipFamilyType=IpFamilyType::UNKNOWN`)=0
- virtual `int getSlotId` ()=0
- virtual `~IDataFilterManager` ()

### 5.8.2.17.1 Constructors and Destructors

**5.8.2.17.1.1** `virtual telux::data::IDataFilterManager::~IDataFilterManager ( ) [virtual]`

Destructor of `IDataFilterManager`

### 5.8.2.17.2 Member Function Documentation

**5.8.2.17.2.1** `virtual bool telux::data::IDataFilterManager::isReady ( ) [pure virtual]`

Checks the status of Data Filter Service and if the other APIs are ready for use, and returns the result.

#### Returns

True if the services are ready otherwise false.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.8.2.17.2.2** `virtual std::future<bool> telux::data::IDataFilterManager::onReady ( ) [pure virtual]`

Wait for Data Filter Service to be ready.

#### Returns

A future that caller can wait on to be notified when Data Filter Service are ready.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.8.2.17.2.3** `virtual telux::common::Status telux::data::IDataFilterManager::registerListener ( std::weak_ptr< IDataFilterListener > listener ) [pure virtual]`

Register a listener for powersave filtering mode notifications.

#### Parameters

in	<i>listener</i>	- Pointer of <a href="#">IDataFilterListener</a> object that processes the notification
----	-----------------	---

#### Returns

Status of registerListener i.e success or suitable status code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.8.2.17.2.4** `virtual telux::common::Status telux::data::IDataFilterManager::deregisterListener ( std::weak_ptr< IDataFilterListener > listener ) [pure virtual]`

Remove a previously registered listener.

#### Parameters

in	<i>listener</i>	- Previously registered <a href="#">IDataFilterListener</a> that needs to be removed
----	-----------------	--

#### Returns

Status of deregisterListener, success or suitable status code

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.8.2.17.2.5** `virtual telux::common::Status telux::data::IDataFilterManager::setDataRestrictMode ( DataRestrictMode mode, telux::common::ResponseCallback callback = nullptr, int profileId = PROFILE_ID_MAX, IpFamilyType ipFamilyType = IpFamilyType::UNKNOWN ) [pure virtual]`

Changes the Data Powersave filter mode and auto exit feature.

This API enables or disables the powersave filtering mode of the packet data session..

in	<i>mode</i>	- Enable or disable the powersave filtering mode.
in	<i>callback</i>	- Optional callback to get the response for the change in filter mode.
in	<i>profileId</i>	- Optional Profile ID for data connection. If user does not specify the profile id, then the API applies to all the currently running data connection. If user wants to apply the changes to any specific data connection, then its profile id can be specified as input.

**Note**

Currently, this API does not support enabling data restrict filters per profile. This API will only accept the default value (PROFILE\_ID\_MAX). Any other value will result in an error.

**Parameters**

in	<i>ipFamilyType</i>	- Optional IP Family type <a href="#">IpFamilyType</a> . If user does not specify the ip family type, then the API applies to all the currently running data connection. If user wants to apply the changes to any specific data connection, then its ip family type can be specified as input.
----	---------------------	---

**Note**

Currently, this API does not support enabling data restrict filters per IP Family type. This API will only accept the default value ([IpFamilyType::UNKNOWN](#)). Any other value will result in an error.

**Returns**

Status of setDataRestrictMode i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.8.2.17.2.6 virtual telux::common::Status telux::data::IDataFilterManager::requestDataRestrictMode ( std::string ifaceName, DataRestrictModeCb callback ) [pure virtual]

Get the current Data Powersave filter mode

**Parameters**

in	<i>ifaceName</i>	- Interface name for data connection. Note: Some platforms support filter mode that applies to all PDNs (i.e. global) and some platforms support filter mode per PDN. On platforms that support global filter mode, this parameter will be ignored.
in	<i>callback</i>	- callback function to get the result of API.

**Returns**

Status of requestDataRestrictMode i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.8.2.17.2.7 virtual telux::common::Status telux::data::IDataFilterManager::addDataRestrict↔  
Filter ( std::shared\_ptr< IIpFilter > & filter, telux::common::ResponseCallback  
callback = nullptr, int profileId = PROFILE\_ID\_MAX, IpFamilyType ipFamilyType =  
IpFamilyType::UNKNOWN ) [pure virtual]**

This API adds a filter rules for a packet data session to achieve power savings. In case when DataRestrict mode is enabled and AP is in suspended state, Modem will filter all the incoming data packet and route them to AP only if filter rules added via addDataRestrictFilter API matches the criteria, else they are queued at Modem itself and not forwarded to AP, until filter mode is disabled.

**Parameters**

in	<i>filter</i>	- Filter rule.
in	<i>callback</i>	- Optional callback to get the response.
in	<i>profileId</i>	- Optional Profile ID for data connection. If user does not specify the profile id, then the API applies to all the currently running data connection. If user wants to apply the changes to any specific data connection, then its profile id can be specified as input.

**Note**

Currently, this API does not support enabling data restrict filters per profile. This API will only accept the default value (PROFILE\_ID\_MAX). Any other value will result in an error.

**Parameters**

in	<i>ipFamilyType</i>	- Optional IP Family type <a href="#">IpFamilyType</a> . If user does not specify the ip family type, then the API applies to all the currently running data connection. If user wants to apply the changes to any specific data connection, then its ip family type can be specified as input.
----	---------------------	---

**Note**

Currently, this API does not support enabling data restrict filters per IP Family type. This API will only accept the default value ([IpFamilyType::UNKNOWN](#)). Any other value will result in an error.

**Returns**

Status of addDataRestrictFilter i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.8.2.17.2.8** `virtual telux::common::Status telux::data::IDataFilterManager::removeAllDataRestrictFilters ( telux::common::ResponseCallback callback = nullptr, int profileId = PROFILE_ID_MAX, IpFamilyType ipFamilyType = IpFamilyType::UNKNOWN ) [pure virtual]`

This API removes all the previous added powersave filter for a packet data session

**Parameters**

in	<i>callback</i>	- Optional callback to get the response.
in	<i>profileId</i>	- Optional Profile ID for data connection. If user does not specify the profile id, then the API applies to all the currently running data connection. If user wants to apply the changes to any specific data connection, then its profile id can be specified as input.

**Note**

Currently, this API does not support enabling data restrict filters per profile. This API will only accept the default value (PROFILE\_ID\_MAX). Any other value will result in an error.

**Parameters**

in	<i>ipFamilyType</i>	- Optional IP Family type <a href="#">IpFamilyType</a> . If user does not specify the ip family type, then the API applies to all the currently running data connection. If user wants to apply the changes to any specific data connection, then its ip family type can be specified as input.
----	---------------------	---

**Note**

Currently, this API does not support enabling data restrict filters per IP Family type. This API will only accept the default value ([IpFamilyType::UNKNOWN](#)). Any other value will result in an error.

**Returns**

Status of `removeAllDataRestrictFilters` i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.



#### 5.8.2.17.2.9 virtual int telux::data::IDataFilterManager::getSlotId ( ) [pure virtual]

Get associated slot id for the Data Filter Manager.

#### Returns

SlotId

### 5.8.2.18 class telux::data::DataProfile

[DataProfile](#) class represents single data profile on the modem.

#### Public member functions

- [DataProfile](#) (int id, const std::string &name, const std::string &apn, const std::string &username, const std::string &password, [IpFamilyType](#) ipFamilyType, [TechPreference](#) techPref, [AuthProtocolType](#) authType)
- int [getId](#) ()
- std::string [getName](#) ()
- std::string [getApn](#) ()
- std::string [getUserName](#) ()
- std::string [getPassword](#) ()
- [TechPreference](#) [getTechPreference](#) ()
- [AuthProtocolType](#) [getAuthProtocolType](#) ()
- [IpFamilyType](#) [getIpFamilyType](#) ()
- std::string [toString](#) ()

#### Static Public Attributes

- static constexpr int [PROFILE\\_ID\\_INVALID](#) = -1

#### 5.8.2.18.1 Constructors and Destructors

**5.8.2.18.1.1** [telux::data::DataProfile::DataProfile](#) ( int *id*, const std::string & *name*, const std::string & *apn*, const std::string & *username*, const std::string & *password*, [IpFamilyType](#) *ipFamilyType*, [TechPreference](#) *techPref*, [AuthProtocolType](#) *authType* )

#### 5.8.2.18.2 Member Function Documentation

**5.8.2.18.2.1 int telux::data::DataProfile::getId ( )**

Get profile identifier.

**Returns**

profile id

**5.8.2.18.2.2 std::string telux::data::DataProfile::getName ( )**

Get profile name.

**Returns**

profile name

**5.8.2.18.2.3 std::string telux::data::DataProfile::getApn ( )**

Get Access Point Name (APN) name.

**Returns**

APN name

**5.8.2.18.2.4 std::string telux::data::DataProfile::getUserName ( )**

Get profile user name.

**Returns**

user name

**5.8.2.18.2.5 std::string telux::data::DataProfile::getPassword ( )**

Get profile password.

**Returns**

profile password

**5.8.2.18.2.6 TechPreference telux::data::DataProfile::getTechPreference ( )**

Get technology preference.

**Returns**

TechPreference [TechPreference](#)

**5.8.2.18.2.7 AuthProtocolType telux::data::DataProfile::getAuthProtocolType ( )**

Get authentication preference.

**Returns**

AuthProtocolType [AuthProtocolType](#)

**5.8.2.18.2.8 IpFamilyType telux::data::DataProfile::getIpFamilyType ( )**

Get IP Family type.

**Returns**

IpFamilyType [IpFamilyType](#)

**5.8.2.18.2.9 std::string telux::data::DataProfile::toString ( )**

Get the text related informative representation of this object.

**Returns**

String containing informative string.

**5.8.2.18.3 Field Documentation****5.8.2.18.3.1 constexpr int telux::data::DataProfile::PROFILE\_ID\_INVALID = -1 [static]****5.8.2.19 class telux::data::IDataProfileListener**

Listener class for getting profile change notification.

The methods in the listener can be invoked from multiple threads. It is client's responsibility to make sure the implementation is thread safe.

**Public member functions**

- virtual void [onServiceStatusChange](#) (telux::common::ServiceStatus status, SlotId slotId)
- virtual void [onProfileUpdate](#) (int profileId, [TechPreference](#) techPreference, [ProfileChangeEvent](#) event)
- virtual [~IDataProfileListener](#) ()

**5.8.2.19.1 Constructors and Destructors****5.8.2.19.1.1 virtual telux::data::IDataProfileListener::~~IDataProfileListener ( ) [virtual]**

Destructor of [IDataProfileListener](#)

### 5.8.2.19.2 Member Function Documentation

#### 5.8.2.19.2.1 virtual void telux::data::IDataProfileListener::onServiceStatusChange ( telux::common::↔ ServiceStatus *status*, SlotId *slotId* ) [virtual]

This function is called when service status changes.

##### Parameters

in	<i>status</i>	- <a href="#">ServiceStatus</a>
in	<i>slotId</i>	- SlotId

#### 5.8.2.19.2.2 virtual void telux::data::IDataProfileListener::onProfileUpdate ( int *profileId*, Tech↔ Preference *techPreference*, ProfileChangeEvent *event* ) [virtual]

This function is called when profile change happens.

##### Parameters

in	<i>profileId</i>	- ID of the updated profile.
in	<i>techPreference</i>	- TechPreference.
in	<i>event</i>	- Event that caused the change in profile.

### 5.8.2.20 class telux::data::IDataProfileManager

[IDataProfileManager](#) is a primary interface for profile management.

##### Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status requestProfileList](#) (std::shared\_ptr< [IDataProfileListCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status createProfile](#) (const [ProfileParams](#) &profileParams, std::shared\_ptr< [IDataCreateProfileCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status deleteProfile](#) (uint8\_t profileId, [TechPreference](#) techPreference, std::shared\_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status modifyProfile](#) (uint8\_t profileId, const [ProfileParams](#) &profileParams, std::shared\_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status queryProfile](#) (const [ProfileParams](#) &profileParams, std::shared\_ptr< [IDataProfileListCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status requestProfile](#) (uint8\_t profileId, [TechPreference](#) techPreference, std::shared\_ptr< [IDataProfileCallback](#) > callback=nullptr)=0
- virtual int [getSlotId](#) ()=0

- virtual `telux::common::Status registerListener (std::weak_ptr< telux::data::IDataProfileListener > listener)=0`
- virtual `telux::common::Status deregisterListener (std::weak_ptr< telux::data::IDataProfileListener > listener)=0`
- virtual `~IDataProfileManager ()`

### 5.8.2.20.1 Constructors and Destructors

#### 5.8.2.20.1.1 virtual `telux::data::IDataProfileManager::~IDataProfileManager ( ) [virtual]`

Destructor for `IDataProfileManager`

### 5.8.2.20.2 Member Function Documentation

#### 5.8.2.20.2.1 virtual `bool telux::data::IDataProfileManager::isSubsystemReady ( ) [pure virtual]`

Checks if the data profile manager is ready.

#### Returns

True if data profile subsystem is ready for service otherwise false.

#### 5.8.2.20.2.2 virtual `std::future<bool> telux::data::IDataProfileManager::onSubsystemReady ( ) [pure virtual]`

Waits for data profile subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when data profile subsystem is ready.

#### 5.8.2.20.2.3 virtual `telux::common::Status telux::data::IDataProfileManager::requestProfileList ( std::shared_ptr< IDataProfileListCallback > callback = nullptr ) [pure virtual]`

Request list of profiles supported by the device.

#### Parameters

<code>in, out</code>	<code>callback</code>	Callback pointer to get the response.
----------------------	-----------------------	---------------------------------------

#### Returns

Status of request profile i.e. success or suitable error code.

**5.8.2.20.2.4** `virtual telux::common::Status telux::data::IDataProfileManager::createProfile ( const ProfileParams & profileParams, std::shared_ptr< IDataCreateProfileCallback > callback = nullptr ) [pure virtual]`

Create profile based on data profile params.

#### Parameters

in	<i>profileParams</i>	profileParams configuration to be passed for creating profile either for 3GPP or 3GPP2
in, out	<i>callback</i>	Callback pointer to get the result of create profile

#### Returns

Status of create profile i.e. success or suitable error code.

**5.8.2.20.2.5** `virtual telux::common::Status telux::data::IDataProfileManager::deleteProfile ( uint8_t profileId, TechPreference techPreference, std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]`

Delete profile corresponding to profile identifier.

The deletion of a profile does not affect profile index assignments.

#### Parameters

in	<i>profileId</i>	Profile identifier
in	<i>techPreference</i>	Technology Preference like 3GPP / 3GPP2
in	<i>callback</i>	Callback pointer to get the result of delete profile

#### Returns

Status of delete profile i.e. success or suitable error code.

**5.8.2.20.2.6** `virtual telux::common::Status telux::data::IDataProfileManager::modifyProfile ( uint8_t profileId, const ProfileParams & profileParams, std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]`

Modify existing profile with new profile params.

#### Parameters

in	<i>profileId</i>	Profile identifier of profile to be modified
in	<i>profileParams</i>	New profileParams configuration passed for updating existing profile
in	<i>callback</i>	Callback pointer to get the result of modify profile

#### Returns

Status of modify profile i.e. success or suitable error code.

**5.8.2.20.2.7** `virtual telux::common::Status telux::data::IDataProfileManager::queryProfile ( const ProfileParams & profileParams, std::shared_ptr< IDataProfileListCallback > callback = nullptr ) [pure virtual]`

Lookup modem profile/s based on given profile params.

#### Parameters

in	<i>profileParams</i>	<a href="#">ProfileParams</a> configuration to be passed
in	<i>callback</i>	Callback pointer to get the result of query profile

#### Returns

Status of query profile i.e. success or suitable error code.

**5.8.2.20.2.8** `virtual telux::common::Status telux::data::IDataProfileManager::requestProfile ( uint8_t profileId, TechPreference techPreference, std::shared_ptr< IDataProfileCallback > callback = nullptr ) [pure virtual]`

Get data profile corresponding to profile identifier.

#### Parameters

in	<i>profileId</i>	Profile identifier
in	<i>techPreference</i>	Technology preference <ul style="list-style-type: none"> <li>• <a href="#">TechPreference</a></li> </ul>
in	<i>callback</i>	Callback pointer to get the result of get profile by ID

#### Returns

Status of requestProfile i.e. success or suitable error code.

**5.8.2.20.2.9** `virtual int telux::data::IDataProfileManager::getSlotId ( ) [pure virtual]`

Get associated slot id for the Data Profile Manager.

#### Returns

SlotId

**5.8.2.20.2.10** `virtual telux::common::Status telux::data::IDataProfileManager::registerListener ( std::weak_ptr< telux::data::IDataProfileListener > listener ) [pure virtual]`

Listen for create, delete and modify profile events.

in	<i>listener</i>	- Listener that processes the notification.
----	-----------------	---

**Returns**

Status.

**5.8.2.20.2.11** `virtual telux::common::Status telux::data::IDataProfileManager::deregisterListener ( std::weak_ptr< telux::data::IDataProfileListener > listener ) [pure virtual]`

De-register listener.

**Parameters**

in	<i>listener</i>	- Listener to be de-registered.
----	-----------------	---------------------------------

**Returns**

Status.

**5.8.2.21 class telux::data::IDataCreateProfileCallback**

Interface for create profile callback object. Client needs to implement this interface to get single shot responses for command like create profile.

The methods in callback can be invoked from multiple different threads. The implementation should be thread safe.

**Public member functions**

- virtual void [onResponse](#) (int profileId, [telux::common::ErrorCode](#) error)

**5.8.2.21.1 Member Function Documentation**

**5.8.2.21.1.1** `virtual void telux::data::IDataCreateProfileCallback::onResponse ( int profileId, telux::common::ErrorCode error ) [virtual]`

This function is called with the response to [IDataProfileManager::createProfile](#) API.

**Parameters**

in	<i>profileId</i>	created profile Id for the response. Use <a href="#">IDataProfileManager::requestProfile</a> to get the data profile
in	<i>error</i>	<a href="#">telux::common::ErrorCode</a>



### 5.8.2.22 class telux::data::IDataProfileListCallback

Interface for getting list of [DataProfile](#) using callback. Client needs to implement this interface to get single shot responses for commands like get profile list and query profile.

The methods in callback can be invoked from different threads. The implementation should be thread safe.

#### Public member functions

- virtual void [onProfileListResponse](#) (const std::vector< std::shared\_ptr< [DataProfile](#) >> &profiles, [telux::common::ErrorCode](#) error)

#### 5.8.2.22.1 Member Function Documentation

**5.8.2.22.1.1** virtual void [telux::data::IDataProfileListCallback::onProfileListResponse](#) ( const std↵  
::vector< std::shared\_ptr< [DataProfile](#) >> & *profiles*, [telux::common::ErrorCode](#) *error* )  
[virtual]

This function is called with the response to requestProfileList API or queryProfile API.

#### Parameters

in	<i>profiles</i>	List of profiles supported by the device
in	<i>error</i>	<a href="#">telux::common::ErrorCode</a>

### 5.8.2.23 class telux::data::IDataProfileCallback

Interface for getting [DataProfile](#) using callback. Client needs to implement this interface to get single shot responses for command like create profile.

The methods in callback can be invoked from multiple different threads. The implementation should be thread safe.

#### Public member functions

- virtual void [onResponse](#) (const std::shared\_ptr< [DataProfile](#) > &profile, [telux::common::ErrorCode](#) error)

#### 5.8.2.23.1 Member Function Documentation

**5.8.2.23.1.1** virtual void [telux::data::IDataProfileCallback::onResponse](#) ( const std::shared\_ptr<  
[DataProfile](#) > & *profile*, [telux::common::ErrorCode](#) *error* ) [virtual]

This function is called with the response to [IDataProfileManager::requestProfile](#) API.

#### Parameters

in	<i>profile</i>	Response of data profile
in	<i>error</i>	<a href="#">telux::common::ErrorCode</a>

### 5.8.2.24 class telux::data::IDataSettingsManager

The Data Settings Manager class provides an interface to the data subsystem settings.

#### Public member functions

- virtual bool `isSubsystemReady ()=0`
- virtual `std::future< bool > onSubsystemReady ()=0`
- virtual `telux::common::Status switchBackHaul (BackhaulInfo source, BackhaulInfo dest, bool applyToAll=false, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::ErrorCode setLatencyConfig (LatencyConfig latencyConfig)=0`
- virtual `telux::common::ErrorCode getLatencyConfig (LatencyConfig &latencyConfig)=0`
- virtual `telux::common::Status registerListener (std::weak_ptr< IDataSettingsListener > listener)=0`
- virtual `telux::common::Status deregisterListener (std::weak_ptr< IDataSettingsListener > listener)=0`

#### 5.8.2.24.1 Member Function Documentation

##### 5.8.2.24.1.1 virtual bool telux::data::IDataSettingsManager::isSubsystemReady ( ) [pure virtual]

Checks if the data subsystem is ready.

#### Returns

if Settings Manager is ready, false otherwise.

##### 5.8.2.24.1.2 virtual std::future<bool> telux::data::IDataSettingsManager::onSubsystemReady ( ) [pure virtual]

Waits for the data subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when data settings manager is ready.

##### 5.8.2.24.1.3 virtual telux::common::Status telux::data::IDataSettingsManager::switchBackHaul ( BackhaulInfo source, BackhaulInfo dest, bool applyToAll = false, telux::common::ResponseCallback callback = nullptr ) [pure virtual]

Switch backhaul to be used by traffic. Provides the ability to re-route clients traffic from one backhaul to another. Clients must call this API for each backhaul switch. For instance, if the default bridge (bridge0) and the on-demand bridge (bridges created by VLANs) need to be re-routed to WLAN, this API must be called twice for the default profile ID and the on-demand profile ID.. If destination backhaul is WLAN (WLAN in Station Mode):

- Traffic associated with the default and on-demand bridges will be re-routed to WLAN backhaul.

- Client traffic can only be re-routed to WLAN backhaul if the station is connected to an external access point.
- VLANs mapped to WWAN backhaul will be automatically mapped to WLAN backhaul.
- Firewall and DMZ rules configured on WLAN backhaul (if configured before calling this API) will be automatically activated. If destination backhaul is WWAN:
  - Any VLAN profile ID mapping configured in the destination backhaul prior to calling this API will be applied automatically.
  - Any firewall or DMZ rule configured on WWAN backhaul before calling this API will be activated automatically.

### Parameters

in	<i>source</i>	Backhaul <a href="#">telux::data::BackhaulInfo</a> to re-route traffic from
in	<i>dest</i>	Backhaul <a href="#">telux::data::BackhaulType</a> to re-route traffic to
in	<i>applyToAll</i>	Traffic on all source backhauls will be routed to dest backhauls if the source backhaul type is <a href="#">telux::data::BackhaulType::WLAN</a> , traffic on all WWAN backhauls (default and on-demand) will be routed to dest backhaul. if dest backhaul type is <a href="#">telux::data::BackhaulType::WWAN</a> traffic on source backhaul will be routed to WWAN backhauls (default and on-demand) based on vlan-backhaul binding set by <a href="#">telux::data::net::IVlanManager::bindToBackhaul</a>
in	<i>callback</i>	Optional callback to get the response for switchBackHaul.

### Returns

Status of switchBackHaul, i.e., success or applicable status code

### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.8.2.24.1.4 virtual [telux::common::ErrorCode](#) [telux::data::IDataSettingsManager::setLatencyConfig \( LatencyConfig latencyConfig \)](#) [pure virtual]

Sets the latency level for data traffic that has been marked as prioritized data. This only affects the latency level in the modem on the UE. This does not impact network level latency or QoS. This API allows clients to set the uplink latency level of data they deem as time critical compared to the rest of the data flowing in the system. If the latency level is set to [LatencyLevel::LOW](#), then the implementation will prioritize this data over other non-prioritized data flows. Configuration can be reset by setting [LatencyLevel::NORMAL](#) (no priority).

This is a global setting applicable to all WWAN profiles. Configuration set via this API is not persistent over reboot or sub-system restart (updated via [IDataSettingsListener::onServiceStatusChange](#)). After reboot or SSR, configuration will reset to default, i.e., [LatencyLevel::NORMAL](#) (no priority).

in	<i>latencyConfig</i>	Latency/priority configuration
----	----------------------	--------------------------------

**Returns**

Immediate error code of setLatencyConfig, i.e., success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.8.2.24.1.5 virtual telux::common::ErrorCode telux::data::IDataSettingsManager::getLatencyConfig ( LatencyConfig & *latencyConfig* ) [pure virtual]

Get latency level. This API can be used to get the current latency configuration. See [setLatencyConfig](#) for more information.

**Parameters**

out	<i>latencyConfig</i>	Latency/priority configuration
-----	----------------------	--------------------------------

**Returns**

Immediate error code of getLatencyConfig, i.e., success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.8.2.24.1.6 virtual telux::common::Status telux::data::IDataSettingsManager::registerListener ( std::weak\_ptr< IDataSettingsListener > *listener* ) [pure virtual]

Registers a listener to receive data settings notifications.

**Parameters**

in	<i>listener</i>	Pointer to the <a href="#">IDataSettingsListener</a> object to process received notifications.
----	-----------------	--

**Returns**

Status of registerListener success or applicable status code

#### 5.8.2.24.1.7 virtual telux::common::Status telux::data::IDataSettingsManager::deregisterListener ( std::weak\_ptr< IDataSettingsListener > *listener* ) [pure virtual]

Removes a previously added listener.

in	<i>listener</i>	Point to the <a href="#">IDataSettingsListener</a> object to remove.
----	-----------------	--

## Returns

Status of deregisterListener success or applicable status code

### 5.8.2.25 class telux::data::IDataSettingsListener

Interface for the Data...Clients need to implement this interface to access Data Settings service notifications like onServiceStatusChange.

The listener methods can be invoked from multiple different threads. The implementation should be thread safe.

#### Public member functions

- virtual void [onServiceStatusChange](#) ([telux::common::ServiceStatus](#) status)
- virtual [~IDataSettingsListener](#) ()

#### 5.8.2.25.1 Constructors and Destructors

##### 5.8.2.25.1.1 virtual telux::data::IDataSettingsListener::~IDataSettingsListener ( ) [virtual]

Destructor for [IDataSettingsListener](#)

#### 5.8.2.25.2 Member Function Documentation

##### 5.8.2.25.2.1 virtual void telux::data::IDataSettingsListener::onServiceStatusChange ( [telux::common::ServiceStatus](#) *status* ) [virtual]

This function is called when service status changes.

#### Parameters

in	<i>status</i>	- <a href="#">ServiceStatus</a>
----	---------------	---------------------------------

### 5.8.2.26 struct telux::data::RoamingStatus

Roaming Status.

#### Data fields

Type	Field	Description
bool	isRoaming	True: Roaming on, False: Roaming off
<a href="#">RoamingType</a>	type	International/Domestic. Valid only if roaming is on

### 5.8.2.27 struct telux::data::ServiceStatus

Data Service Status Info.

#### Data fields

Type	Field	Description
<a href="#">DataService↔ State</a>	serviceState	
<a href="#">NetworkRat</a>	networkRat	

### 5.8.2.28 class telux::data::IServingSystemManager

Serving System Manager class provides APIs related to the serving system for data functionality. For example, ability to query or be notified about the state of the platform's WWAN PS data serving information.

#### Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status requestServiceStatus](#) ([RequestServiceStatusResponseCb](#) callback)=0
- virtual [telux::common::Status requestRoamingStatus](#) ([RequestRoamingStatusResponseCb](#) callback)=0
- virtual [telux::common::Status makeDormant](#) ([telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status registerListener](#) (std::weak\_ptr< [IServingSystemListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak\_ptr< [IServingSystemListener](#) > listener)=0
- virtual SlotId [getSlotId](#) ()=0
- virtual [~IServingSystemManager](#) ()

#### 5.8.2.28.1 Constructors and Destructors

5.8.2.28.1.1 virtual [telux::data::IServingSystemManager::~~IServingSystemManager](#) ( ) [[virtual](#)]

Destructor of [IServingSystemManager](#)

#### 5.8.2.28.2 Member Function Documentation

**5.8.2.28.2.1** `virtual bool telux::data::IServingSystemManager::isSubsystemReady ( ) [pure virtual]`

Checks if the data subsystem is ready.

#### Returns

True if Data Connection Manager is ready for service, otherwise returns false.

**5.8.2.28.2.2** `virtual std::future<bool> telux::data::IServingSystemManager::onSubsystemReady ( ) [pure virtual]`

Wait for data subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when card manager is ready.

**5.8.2.28.2.3** `virtual telux::common::Status telux::data::IServingSystemManager::requestServiceStatus ( RequestServiceStatusResponseCb callback ) [pure virtual]`

Queries the current serving network status

#### Parameters

in	<i>callback</i>	callback to get response for requestServiceStatus
----	-----------------	---

#### Returns

Status of requestServiceStatus i.e. success or suitable status code. if requestServiceStatus returns failure, callback will not be invoked.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.8.2.28.2.4** `virtual telux::common::Status telux::data::IServingSystemManager::requestRoamingStatus ( RequestRoamingStatusResponseCb callback ) [pure virtual]`

Queries the current roaming status

#### Parameters

in	<i>callback</i>	callback to get response for requestRoamingStatus
----	-----------------	---

#### Returns

Status of requestRoamingStatus i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.8.2.28.2.5 virtual telux::common::Status telux::data::IServingSystemManager::makeDormant ( telux::common::ResponseCallback *callback* = *nullptr* ) [pure virtual]

Request modem switch to dormant state: Certain network operations can only be performed when modem is in dormant state. This API provides an ability for clients to request modem to immediately transition to dormant state for such scenarios.

Clients must ensure no data calls are in process of bring up/tear down and there is no traffic on any active data calls when this API is called.

**Parameters**

in	<i>callback</i>	optional callback to get the response of makeDormancy
----	-----------------	---

**Returns**

[telux::common::ErrorCode::SUCCESS](#) if request is honored by network.

[telux::common::ErrorCode::INVALID\\_STATE](#) is returned if:

- There is no active data calls
- Any Data calls is going through bring up/tear down
- There is data traffic on any active data calls If API fails, application is responsible for re-attempting operation at later time once the above conditions are met.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.8.2.28.2.6 virtual telux::common::Status telux::data::IServingSystemManager::registerListener ( std::weak\_ptr< IServingSystemListener > *listener* ) [pure virtual]

Register a listener for specific updates from serving system.

**Parameters**

in	<i>listener</i>	Pointer of <a href="#">IServingSystemListener</a> object that processes the notification
----	-----------------	--

**Returns**

Status of registerListener i.e success or suitable status code.



**5.8.2.28.2.7** virtual telux::common::Status telux::data::IServingSystemManager::deregisterListener ( std::weak\_ptr< IServingSystemListener > *listener* ) [pure virtual]

Deregister the previously added listener.

#### Parameters

in	<i>listener</i>	Previously registered <a href="#">IServingSystemListener</a> that needs to be removed
----	-----------------	---

#### Returns

Status of removeListener i.e. success or suitable status code

**5.8.2.28.2.8** virtual SlotId telux::data::IServingSystemManager::getSlotId ( ) [pure virtual]

Get associated slot id for the Serving System Manager.

#### Returns

SlotId

### 5.8.2.29 class telux::data::IServingSystemListener

Listener class for data serving system change notification.

The listener method can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe.

#### Public member functions

- virtual void [onServiceStatusChange](#) (telux::common::ServiceStatus status)
- virtual void [onServiceStateChanged](#) (ServiceStatus status)
- virtual void [onRoamingStatusChanged](#) (RoamingStatus status)
- virtual [~IServingSystemListener](#) ()

#### 5.8.2.29.1 Constructors and Destructors

**5.8.2.29.1.1** virtual telux::data::IServingSystemListener::~~IServingSystemListener ( ) [virtual]

Destructor of [IServingSystemListener](#)

#### 5.8.2.29.2 Member Function Documentation

### 5.8.2.29.2.1 virtual void telux::data::IServingSystemListener::onServiceStatusChange ( telux↔ ::common::ServiceStatus *status* ) [virtual]

This function is called when service status changes.

#### Parameters

in	<i>status</i>	- <a href="#">ServiceStatus</a>
----	---------------	---------------------------------

### 5.8.2.29.2.2 virtual void telux::data::IServingSystemListener::onServiceStateChanged ( [ServiceStatus](#) *status* ) [virtual]

This function is called whenever service state is changed.

#### Parameters

in	<i>status</i>	<a href="#">ServiceStatus</a>
----	---------------	-------------------------------

### 5.8.2.29.2.3 virtual void telux::data::IServingSystemListener::onRoamingStatusChanged ( [RoamingStatus](#) *status* ) [virtual]

This function is called whenever roaming status is changed.

#### Parameters

in	<i>status</i>	<a href="#">RoamingStatus</a>
----	---------------	-------------------------------

### 5.8.2.30 union telux::data::DataCallEndReason. \_\_unnamed\_\_

#### Data fields

Type	Field	Description
<a href="#">MobileIp↔ ReasonCode</a>	IpCode	
<a href="#">Internal↔ ReasonCode</a>	internalCode	
<a href="#">CallManager↔ ReasonCode</a>	cmCode	
<a href="#">SpecReason↔ Code</a>	specCode	
<a href="#">PPPReason↔ Code</a>	pppCode	
<a href="#">EHRPD↔ ReasonCode</a>	ehrpdcCode	
<a href="#">Ipv6Reason↔ Code</a>	ipv6Code	
<a href="#">Handoff↔ ReasonCode</a>	handOffCode	

## 5.8.3 Enumeration Type Documentation

### 5.8.3.1 enum telux::data::IpFamilyType [strong]

Preferred IP family for the connection

#### Enumerator

**UNKNOWN**  
**IPV4** IPv4 data connection  
**IPV6** IPv6 data connection  
**IPV4V6** IPv4 and IPv6 data connection

### 5.8.3.2 enum telux::data::TechPreference [strong]

Technology Preference

#### Enumerator

**UNKNOWN**  
**TP\_3GPP** UMTS, LTE  
**TP\_3GPP2** CDMA  
**TP\_ANY** ANY (3GPP or 3GPP2)

### 5.8.3.3 enum telux::data::AuthProtocolType [strong]

Authentication protocol preference type to be used for PDP context.

#### Enumerator

**AUTH\_NONE**  
**AUTH\_PAP** Password Authentication Protocol  
**AUTH\_CHAP** Challenge Handshake Authentication Protocol  
**AUTH\_PAP\_CHAP**

### 5.8.3.4 enum telux::data::DataRestrictModeType [strong]

Defines the supported filtering mode of the packet data session. DataRestrictFilter

#### Enumerator

**UNKNOWN**  
**DISABLE**  
**ENABLE**

### 5.8.3.5 enum telux::data::DataCallStatus [strong]

Data call event status

#### Enumerator

**INVALID** Invalid

**NET\_CONNECTED** Call is connected  
**NET\_NO\_NET** Call is disconnected  
**NET\_IDLE** Call is in idle state  
**NET\_CONNECTING** Call is in connecting state  
**NET\_DISCONNECTING** Call is in disconnecting state  
**NET\_RECONFIGURED** Interface is reconfigured, IP Address got changed  
**NET\_NEWADDR** A new IP address was added on an existing call  
**NET\_DELADDR** An IP address was removed from the existing interface

### 5.8.3.6 enum telux::data::DataBearerTechnology [strong]

Bearer technology types (returned with getCurrentBearerTech).

#### Enumerator

**UNKNOWN** Unknown bearer.  
**CDMA\_1X** 1X technology.  
**EVDO\_REV0** CDMA Rev 0.  
**EVDO\_REVA** CDMA Rev A.  
**EVDO\_REVB** CDMA Rev B.  
**EHRPD** EHRPD.  
**FMC** Fixed mobile convergence.  
**HRPD** HRPD  
**BEARER\_TECH\_3GPP2\_WLAN** IWLAN  
**WCDMA** WCDMA.  
**GPRS** GPRS.  
**HSDPA** HSDPA.  
**HSUPA** HSUPA.  
**EDGE** EDGE.  
**LTE** LTE.  
**HSDPA\_PLUS** HSDPA+.  
**DC\_HSDPA\_PLUS** DC HSDPA+.  
**HSPA** HSPA  
**BEARER\_TECH\_64\_QAM** 64 QAM.  
**TDSCDMA** TD-SCDMA.  
**GSM** GSM  
**BEARER\_TECH\_3GPP\_WLAN** IWLAN  
**BEARER\_TECH\_5G** 5G

### 5.8.3.7 enum telux::data::EndReasonType [strong]

Data call end/termination due to reason type.

#### Enumerator

**CE\_UNKNOWN**  
**CE\_MOBILE\_IP**  
**CE\_INTERNAL**  
**CE\_CALL\_MANAGER\_DEFINED**  
**CE\_3GPP\_SPEC\_DEFINED**

**CE\_PPP**  
**CE\_EHRPD**  
**CE\_IPV6**  
**CE\_HANDOFF**

### 5.8.3.8 enum telux::data::MobileIpReasonCode [strong]

Data call end/termination reason code for [EndReasonType::CE\\_MOBILE\\_IP](#)

#### Enumerator

**CE\_MIP\_FA\_ERR\_REASON\_UNSPECIFIED**  
**CE\_MIP\_FA\_ERR\_ADMINISTRATIVELY\_PROHIBITED**  
**CE\_MIP\_FA\_ERR\_INSUFFICIENT\_RESOURCES**  
**CE\_MIP\_FA\_ERR\_MOBILE\_NODE\_AUTHENTICATION\_FAILURE**  
**CE\_MIP\_FA\_ERR\_HA\_AUTHENTICATION\_FAILURE**  
**CE\_MIP\_FA\_ERR\_REQUESTED\_LIFETIME\_TOO\_LONG**  
**CE\_MIP\_FA\_ERR\_MALFORMED\_REQUEST**  
**CE\_MIP\_FA\_ERR\_MALFORMED\_REPLY**  
**CE\_MIP\_FA\_ERR\_ENCAPSULATION\_UNAVAILABLE**  
**CE\_MIP\_FA\_ERR\_VJHC\_UNAVAILABLE**  
**CE\_MIP\_FA\_ERR\_REVERSE\_TUNNEL\_UNAVAILABLE**  
**CE\_MIP\_FA\_ERR\_REVERSE\_TUNNEL\_IS\_MANDATORY\_AND\_T\_BIT\_NOT\_SET**  
**CE\_MIP\_FA\_ERR\_DELIVERY\_STYLE\_NOT\_SUPPORTED**  
**CE\_MIP\_FA\_ERR\_MISSING\_NAI**  
**CE\_MIP\_FA\_ERR\_MISSING\_HA**  
**CE\_MIP\_FA\_ERR\_MISSING\_HOME\_ADDR**  
**CE\_MIP\_FA\_ERR\_UNKNOWN\_CHALLENGE**  
**CE\_MIP\_FA\_ERR\_MISSING\_CHALLENGE**  
**CE\_MIP\_FA\_ERR\_STALE\_CHALLENGE**  
**CE\_MIP\_HA\_ERR\_REASON\_UNSPECIFIED**  
**CE\_MIP\_HA\_ERR\_ADMINISTRATIVELY\_PROHIBITED**  
**CE\_MIP\_HA\_ERR\_INSUFFICIENT\_RESOURCES**  
**CE\_MIP\_HA\_ERR\_MOBILE\_NODE\_AUTHENTICATION\_FAILURE**  
**CE\_MIP\_HA\_ERR\_FA\_AUTHENTICATION\_FAILURE**  
**CE\_MIP\_HA\_ERR\_REGISTRATION\_ID\_MISMATCH**  
**CE\_MIP\_HA\_ERR\_MALFORMED\_REQUEST**  
**CE\_MIP\_HA\_ERR\_UNKNOWN\_HA\_ADDR**  
**CE\_MIP\_HA\_ERR\_REVERSE\_TUNNEL\_UNAVAILABLE**  
**CE\_MIP\_HA\_ERR\_REVERSE\_TUNNEL\_IS\_MANDATORY\_AND\_T\_BIT\_NOT\_SET**  
**CE\_MIP\_HA\_ERR\_ENCAPSULATION\_UNAVAILABLE**  
**CE\_MIP\_ERR\_REASON\_UNKNOWN**

### 5.8.3.9 enum telux::data::InternalReasonCode [strong]

Data call end/termination reason code for [EndReasonType::CE\\_INTERNAL](#)

**Enumerator**

**CE\_RETRY**  
**CE\_INTERNAL\_ERROR**  
**CE\_CALL\_ENDED**  
**CE\_INTERNAL\_UNKNOWN\_CAUSE\_CODE**  
**CE\_UNKNOWN\_CAUSE\_CODE**  
**CE\_CLOSE\_IN\_PROGRESS**  
**CE\_NW\_INITIATED\_TERMINATION**  
**CE\_APP\_PREEMPTED**  
**CE\_ERR\_PDN\_IPV4\_CALL\_DISALLOWED**  
**CE\_ERR\_PDN\_IPV4\_CALL\_THROTTLED**  
**CE\_ERR\_PDN\_IPV6\_CALL\_DISALLOWED**  
**CE\_ERR\_PDN\_IPV6\_CALL\_THROTTLED**  
**CE\_MODEM\_RESTART**  
**CE\_PDP\_PPP\_NOT\_SUPPORTED**  
**CE\_UNPREFERRED\_RAT**  
**CE\_PHYS\_LINK\_CLOSE\_IN\_PROGRESS**  
**CE\_APN\_PENDING\_HANDOVER**  
**CE\_PROFILE\_BEARER\_INCOMPATIBLE**  
**CE\_MMGSDI\_CARD\_EVT**  
**CE\_LPM\_OR\_PWR\_DOWN**  
**CE\_APN\_DISABLED**  
**CE\_MPIT\_EXPIRED**  
**CE\_IPV6\_ADDR\_TRANSFER\_FAILED**  
**CE\_TRAT\_SWAP\_FAILED**  
**CE\_EHRPD\_TO\_HRPD\_FALLBACK**  
**CE\_MANDATORY\_APN\_DISABLED**  
**CE\_MIP\_CONFIG\_FAILURE**  
**CE\_INTERNAL\_PDN\_INACTIVITY\_TIMER\_EXPIRED**  
**CE\_MAX\_V4\_CONNECTIONS**  
**CE\_MAX\_V6\_CONNECTIONS**  
**CE\_APN\_MISMATCH**  
**CE\_IP\_VERSION\_MISMATCH**  
**CE\_DUN\_CALL\_DISALLOWED**  
**CE\_INVALID\_PROFILE**  
**CE\_INTERNAL\_EPC\_NONEPC\_TRANSITION**  
**CE\_INVALID\_PROFILE\_ID**  
**CE\_INTERNAL\_CALL\_ALREADY\_PRESENT**  
**CE\_IFACE\_IN\_USE**  
**CE\_IP\_PDP\_MISMATCH**  
**CE\_APN\_DISALLOWED\_ON\_ROAMING**  
**CE\_APN\_PARAM\_CHANGE**  
**CE\_IFACE\_IN\_USE\_CFG\_MATCH**  
**CE\_NULL\_APN\_DISALLOWED**  
**CE\_THERMAL\_MITIGATION**  
**CE\_SUBS\_ID\_MISMATCH**  
**CE\_DATA\_SETTINGS\_DISABLED**  
**CE\_DATA\_ROAMING\_SETTINGS\_DISABLED**  
**CE\_APN\_FORMAT\_INVALID**  
**CE\_DDS\_CALL\_ABORT**

**CE\_VALIDATION\_FAILURE**  
**CE\_PROFILES\_NOT\_COMPATIBLE**  
**CE\_NULL\_RESOLVED\_APN\_NO\_MATCH**  
**CE\_INVALID\_APN\_NAME**

### 5.8.3.10 enum telux::data::CallManagerReasonCode [strong]

Data call end/termination reason code for [EndReasonType::CE\\_CALL\\_MANAGER\\_DEFINED](#)

#### Enumerator

**CE\_CDMA\_LOCK**  
**CE\_INTERCEPT**  
**CE\_REORDER**  
**CE\_REL\_SO\_REJ**  
**CE\_INCOM\_CALL**  
**CE\_ALERT\_STOP**  
**CE\_ACTIVATION**  
**CE\_MAX\_ACCESS\_PROBE**  
**CE\_CCS\_NOT\_SUPPORTED\_BY\_BS**  
**CE\_NO\_RESPONSE\_FROM\_BS**  
**CE\_REJECTED\_BY\_BS**  
**CE\_INCOMPATIBLE**  
**CE\_ALREADY\_IN\_TC**  
**CE\_USER\_CALL\_ORIG\_DURING\_GPS**  
**CE\_USER\_CALL\_ORIG\_DURING\_SMS**  
**CE\_NO\_CDMA\_SRV**  
**CE\_MC\_ABORT**  
**CE\_PSIST\_NG**  
**CE\_UIM\_NOT\_PRESENT**  
**CE\_RETRY\_ORDER**  
**CE\_ACCESS\_BLOCK**  
**CEACCESS\_BLOCK\_ALL**  
**CE\_IS707B\_MAX\_ACC**  
**CE\_THERMAL\_EMERGENCY**  
**CE\_CALL\_ORIG\_THROTTLED**  
**CE\_USER\_CALL\_ORIG\_DURING\_VOICE\_CALL**  
**CE\_CONF\_FAILED**  
**CE\_INCOM\_REJ**  
**CE\_NEW\_NO\_GW\_SRV**  
**CE\_NEW\_NO\_GPRS\_CONTEXT**  
**CE\_NEW\_ILLEGAL\_MS**  
**CE\_NEW\_ILLEGAL\_ME**  
**CE\_NEW\_GPRS\_SERVICES\_AND\_NON\_GPRS\_SERVICES\_NOT\_ALLOWED**  
**CE\_NEW\_GPRS\_SERVICES\_NOT\_ALLOWED**  
**CE\_NEW\_MS\_IDENTITY\_CANNOT\_BE\_DERIVED\_BY\_THE\_NETWORK**  
**CE\_NEW\_IMPLICITLY\_DETACHED**  
**CE\_NEW\_PLMN\_NOT\_ALLOWED**  
**CE\_NEW\_LA\_NOT\_ALLOWED**  
**CE\_NEW\_GPRS\_SERVICES\_NOT\_ALLOWED\_IN\_THIS\_PLMN**

**CE\_NEW\_PDP\_DUPLICATE**  
**CE\_NEW\_UE\_RAT\_CHANGE**  
**CE\_NEW\_CONGESTION**  
**CE\_NEW\_NO\_PDP\_CONTEXT\_ACTIVATED**  
**CE\_NEW\_ACCESS\_CLASS\_DSAC\_REJECTION**  
**CE\_PDP\_ACTIVATE\_MAX\_RETRY\_FAILED**  
**CE\_RAB\_FAILURE**  
**CE\_ESM\_UNKNOWN\_EPS\_BEARER\_CONTEXT**  
**CE\_DRB\_RELEASED\_AT\_RRC**  
**CE\_NAS\_SIG\_CONN\_RELEASED**  
**CE\_REASON\_EMM\_DETACHED**  
**CE\_EMM\_ATTACH\_FAILED**  
**CE\_EMM\_ATTACH\_STARTED**  
**CE\_LTE\_NAS\_SERVICE\_REQ\_FAILED**  
**CE\_ESM\_ACTIVE\_DEDICATED\_BEARER\_REACTIVATED\_BY\_NW**  
**CE\_ESM\_LOWER\_LAYER\_FAILURE**  
**CE\_ESM\_SYNC\_UP\_WITH\_NW**  
**CE\_ESM\_NW\_ACTIVATED\_DED\_BEARER\_WITH\_ID\_OF\_DEF\_BEARER**  
**CE\_ESM\_BAD\_OTA\_MESSAGE**  
**CE\_ESM\_DS\_REJECTED\_THE\_CALL**  
**CE\_ESM\_CONTEXT\_TRANSFERRED\_DUE\_TO\_IRAT**  
**CE\_DS\_EXPLICIT\_DEACT**  
**CE\_ESM\_LOCAL\_CAUSE\_NONE**  
**CE\_LTE\_NAS\_SERVICE\_REQ\_FAILED\_NO\_THROTTLE**  
**CE\_ACL\_FAILURE**  
**CE\_LTE\_NAS\_SERVICE\_REQ\_FAILED\_DS\_DISALLOW**  
**CE\_EMM\_T3417\_EXPIRED**  
**CE\_EMM\_T3417\_EXT\_EXPIRED**  
**CE\_LRRC\_UL\_DATA\_CNF\_FAILURE\_TXN**  
**CE\_LRRC\_UL\_DATA\_CNF\_FAILURE\_HO**  
**CE\_LRRC\_UL\_DATA\_CNF\_FAILURE\_CONN\_REL**  
**CE\_LRRC\_UL\_DATA\_CNF\_FAILURE\_RLF**  
**CE\_LRRC\_UL\_DATA\_CNF\_FAILURE\_CTRL\_NOT\_CONN**  
**CE\_LRRC\_CONN\_EST\_FAILURE**  
**CE\_LRRC\_CONN\_EST\_FAILURE\_ABORTED**  
**CE\_LRRC\_CONN\_EST\_FAILURE\_ACCESS\_BARRED**  
**CE\_LRRC\_CONN\_EST\_FAILURE\_CELL\_RESEL**  
**CE\_LRRC\_CONN\_EST\_FAILURE\_CONFIG\_FAILURE**  
**CE\_LRRC\_CONN\_EST\_FAILURE\_TIMER\_EXPIRED**  
**CE\_LRRC\_CONN\_EST\_FAILURE\_LINK\_FAILURE**  
**CE\_LRRC\_CONN\_EST\_FAILURE\_NOT\_CAMPED**  
**CE\_LRRC\_CONN\_EST\_FAILURE\_SI\_FAILURE**  
**CE\_LRRC\_CONN\_EST\_FAILURE\_CONN\_REJECT**  
**CE\_LRRC\_CONN\_REL\_NORMAL**  
**CE\_LRRC\_CONN\_REL\_RLF**  
**CE\_LRRC\_CONN\_REL\_CRE\_FAILURE**  
**CE\_LRRC\_CONN\_REL\_OOS\_DURING\_CRE**  
**CE\_LRRC\_CONN\_REL\_ABORTED**  
**CE\_LRRC\_CONN\_REL\_SIB\_READ\_ERROR**  
**CE\_DETACH\_WITH\_REATTACH\_LTE\_NW\_DETACH**



**CE\_DETACH\_WITH\_OUT\_REATTACH\_LTE\_NW\_DETACH**  
**CE\_ESM\_PROC\_TIME\_OUT**  
**CE\_INVALID\_CONNECTION\_ID**  
**CE\_INVALID\_NSAPI**  
**CE\_INVALID\_PRI\_NSAPI**  
**CE\_INVALID\_FIELD**  
**CE\_RAB\_SETUP\_FAILURE**  
**CE\_PDP\_ESTABLISH\_MAX\_TIMEOUT**  
**CE\_PDP\_MODIFY\_MAX\_TIMEOUT**  
**CE\_PDP\_INACTIVE\_MAX\_TIMEOUT**  
**CE\_PDP\_LOWERLAYER\_ERROR**  
**CE\_PPD\_UNKNOWN\_REASON**  
**CE\_PDP\_MODIFY\_COLLISION**  
**CE\_PDP\_MBMS\_REQUEST\_COLLISION**  
**CE\_MBMS\_DUPLICATE**  
**CE\_SM\_PS\_DETACHED**  
**CE\_SM\_NO\_RADIO\_AVAILABLE**  
**CE\_SM\_ABORT\_SERVICE\_NOT\_AVAILABLE**  
**CE\_MESSAGE\_EXCEED\_MAX\_L2\_LIMIT**  
**CE\_SM\_NAS\_SRV\_REQ\_FAILURE**  
**CE\_RRC\_CONN\_EST\_FAILURE\_REQ\_ERROR**  
**CE\_RRC\_CONN\_EST\_FAILURE\_TAI\_CHANGE**  
**CE\_RRC\_CONN\_EST\_FAILURE\_RF\_UNAVAILABLE**  
**CE\_RRC\_CONN\_REL\_ABORTED\_IRAT\_SUCCESS**  
**CE\_RRC\_CONN\_REL\_RLF\_SEC\_NOT\_ACTIVE**  
**CE\_RRC\_CONN\_REL\_IRAT\_TO\_LTE\_ABORTED**  
**CE\_RRC\_CONN\_REL\_IRAT\_FROM\_LTE\_TO\_G\_CCO\_SUCCESS**  
**CE\_RRC\_CONN\_REL\_IRAT\_FROM\_LTE\_TO\_G\_CCO\_ABORTED**  
**CE\_IMSI\_UNKNOWN\_IN\_HSS**  
**CE\_IMEI\_NOT\_ACCEPTED**  
**CE\_EPS\_SERVICES\_AND\_NON\_EPS\_SERVICES\_NOT\_ALLOWED**  
**CE\_EPS\_SERVICES\_NOT\_ALLOWED\_IN\_PLMN**  
**CE\_MSC\_TEMPORARILY\_NOT\_REACHABLE**  
**CE\_CS\_DOMAIN\_NOT\_AVAILABLE**  
**CE\_ESM\_FAILURE**  
**CE\_MAC\_FAILURE**  
**CE\_SYNCH\_FAILURE**  
**CE\_UE\_SECURITY\_CAPABILITIES\_MISMATCH**  
**CE\_SECURITY\_MODE\_REJ\_UNSPECIFIED**  
**CE\_NON\_EPS\_AUTH\_UNACCEPTABLE**  
**CE\_CS\_FALLBACK\_CALL\_EST\_NOT\_ALLOWED**  
**CE\_NO\_EPS\_BEARER\_CONTEXT\_ACTIVATED**  
**CE\_EMM\_INVALID\_STATE**  
**CE\_NAS\_LAYER\_FAILURE**  
**CE\_MULTI\_PDN\_NOT\_ALLOWED**  
**CE\_EMBMS\_NOT\_ENABLED**  
**CE\_PENDING\_REDIAL\_CALL\_CLEANUP**  
**CE\_EMBMS\_REGULAR\_DEACTIVATION**  
**CE\_TLB\_REGULAR\_DEACTIVATION**  
**CE\_LOWER\_LAYER\_REGISTRATION\_FAILURE**

**CE\_DETACH\_EPS\_SERVICES\_NOT\_ALLOWED**  
**CE\_SM\_INTERNAL\_PDP\_DEACTIVATION**  
**CE\_UNSUPPORTED\_1X\_PREV**  
**CE\_CD\_GEN\_OR\_BUSY**  
**CE\_CD\_BILL\_OR\_AUTH**  
**CE\_CHG\_HDR**  
**CE\_EXIT\_HDR**  
**CE\_HDR\_NO\_SESSION**  
**CE\_HDR\_ORIG\_DURING\_GPS\_FIX**  
**CE\_HDR\_CS\_TIMEOUT**  
**CE\_HDR\_RELEASED\_BY\_CM**  
**CE\_COLLOC\_ACQ\_FAIL**  
**CE\_OTASP\_COMMIT\_IN\_PROG**  
**CE\_NO\_HYBR\_HDR\_SRV**  
**CE\_HDR\_NO\_LOCK\_GRANTED**  
**CE\_HOLD\_OTHER\_IN\_PROG**  
**CE\_HDR\_FADE**  
**CE\_HDR\_ACC\_FAIL**  
**CE\_CLIENT\_END**  
**CE\_NO\_SRV**  
**CE\_FADE**  
**CE\_REL\_NORMAL**  
**CE\_ACC\_IN\_PROG**  
**CE\_ACC\_FAIL**  
**CE\_REDIR\_OR\_HANDOFF**  
**CE\_CM\_UNKNOWN\_ERROR**  
**CE\_OFFLINE**  
**CE\_EMERGENCY\_MODE**  
**CE\_PHONE\_IN\_USE**  
**CE\_INVALID\_MODE**  
**CE\_INVALID\_SIM\_STATE**  
**CE\_NO\_COLLOC\_HDR**  
**CE\_CALL\_CONTROL\_REJECTED**  
**CE\_UNKNOWN**

### 5.8.3.11 enum telux::data::SpecReasonCode [strong]

Data call end/termination reason code for [EndReasonType::CE\\_3GPP\\_SPEC\\_DEFINED](#)

#### Enumerator

**CE\_OPERATOR\_DETERMINED\_BARRING**  
**CE\_NAS\_SIGNALLING\_ERROR**  
**CE\_LLC\_SNDPCP\_FAILURE**  
**CE\_INSUFFICIENT\_RESOURCES**  
**CE\_UNKNOWN\_APN**  
**CE\_UNKNOWN\_PDP**  
**CE\_AUTH\_FAILED**  
**CE\_GGSN\_REJECT**  
**CE\_ACTIVATION\_REJECT**

**CE\_OPTION\_NOT\_SUPPORTED**  
**CE\_OPTION\_UNSUBSCRIBED**  
**CE\_OPTION\_TEMP\_OOO**  
**CE\_NSAPI\_ALREADY\_USED**  
**CE\_REGULAR\_DEACTIVATION**  
**CE\_QOS\_NOT\_ACCEPTED**  
**CE\_NETWORK\_FAILURE**  
**CE\_UMTS\_REACTIVATION\_REQ**  
**CE\_FEATURE\_NOT\_SUPPORTED**  
**CE\_TFT\_SEMANTIC\_ERROR**  
**CE\_TFT\_SYNTAX\_ERROR**  
**CE\_UNKNOWN\_PDP\_CONTEXT**  
**CE\_FILTER\_SEMANTIC\_ERROR**  
**CE\_FILTER\_SYNTAX\_ERROR**  
**CE\_PDP\_WITHOUT\_ACTIVE\_TFT**  
**CE\_IP\_V4\_ONLY\_ALLOWED**  
**CE\_IP\_V6\_ONLY\_ALLOWED**  
**CE\_SINGLE\_ADDR\_BEARER\_ONLY**  
**CE\_ESM\_INFO\_NOT\_RECEIVED**  
**CE\_PDN\_CONN\_DOES\_NOT\_EXIST**  
**CE\_MULTI\_CONN\_TO\_SAME\_PDN\_NOT\_ALLOWED**  
**CE\_MAX\_ACTIVE\_PDP\_CONTEXT\_REACHED**  
**CE\_UNSUPPORTED\_APN\_IN\_CURRENT\_PLMN**  
**CE\_INVALID\_TRANSACTION\_ID**  
**CE\_MESSAGE\_INCORRECT\_SEMANTIC**  
**CE\_INVALID\_MANDATORY\_INFO**  
**CE\_MESSAGE\_TYPE\_UNSUPPORTED**  
**CE\_MSG\_TYPE\_NONCOMPATIBLE\_STATE**  
**CE\_UNKNOWN\_INFO\_ELEMENT**  
**CE\_CONDITIONAL\_IE\_ERROR**  
**CE\_MSG\_AND\_PROTOCOL\_STATE\_UNCOMPATIBLE**  
**CE\_PROTOCOL\_ERROR**  
**CE\_APN\_TYPE\_CONFLICT**  
**CE\_INVALID\_PCSCF\_ADDRESS**  
**CE\_INTERNAL\_CALL\_PREEMPT\_BY\_HIGH\_PRIO\_APN**  
**CE\_EMM\_ACCESS\_BARRED**  
**CE\_EMERGENCY\_IFACE\_ONLY**  
**CE\_IFACE\_MISMATCH**  
**CE\_COMPANION\_IFACE\_IN\_USE**  
**CE\_IP\_ADDRESS\_MISMATCH**  
**CE\_IFACE\_AND\_POL\_FAMILY\_MISMATCH**  
**CE\_EMM\_ACCESS\_BARRED\_INFINITE\_RETRY**  
**CE\_AUTH\_FAILURE\_ON\_EMERGENCY\_CALL**  
**CE\_INVALID\_DNS\_ADDR**  
**CE\_INVALID\_PCSCF\_DNS\_ADDR**  
**CE\_TEST\_LOOPBACK\_MODE\_A\_OR\_B\_ENABLED**  
**CE\_UNKNOWN**

### 5.8.3.12 enum telux::data::PPPReasonCode [strong]

Data call end/termination reason code for [EndReasonType::CE\\_PPP](#)

#### Enumerator

*CE\_PPP\_TIMEOUT*  
*CE\_PPP\_AUTH\_FAILURE*  
*CE\_PPP\_OPTION\_MISMATCH*  
*CE\_PPP\_PAP\_FAILURE*  
*CE\_PPP\_CHAP\_FAILURE*  
*CE\_PPP\_CLOSE\_IN\_PROGRESS*  
*CE\_PPP\_NV\_REFRESH\_IN\_PROGRESS*  
*CE\_PPP\_UNKNOWN*

### 5.8.3.13 enum telux::data::EHRPDReasonCode [strong]

Data call end/termination reason code for [EndReasonType::CE\\_EHRPD](#)

#### Enumerator

*CE\_EHRPD\_SUBS\_LIMITED\_TO\_V4*  
*CE\_EHRPD\_SUBS\_LIMITED\_TO\_V6*  
*CE\_EHRPD\_VSNCP\_TIMEOUT*  
*CE\_EHRPD\_VSNCP\_FAILURE*  
*CE\_EHRPD\_VSNCP\_3GPP2I\_GEN\_ERROR*  
*CE\_EHRPD\_VSNCP\_3GPP2I\_UNAUTH\_APN*  
*CE\_EHRPD\_VSNCP\_3GPP2I\_PDN\_LIMIT\_EXCEED*  
*CE\_EHRPD\_VSNCP\_3GPP2I\_NO\_PDN\_GW*  
*CE\_EHRPD\_VSNCP\_3GPP2I\_PDN\_GW\_UNREACH*  
*CE\_EHRPD\_VSNCP\_3GPP2I\_PDN\_GW\_REJ*  
*CE\_EHRPD\_VSNCP\_3GPP2I\_INSUFF\_PARAM*  
*CE\_EHRPD\_VSNCP\_3GPP2I\_RESOURCE\_UNAVAIL*  
*CE\_EHRPD\_VSNCP\_3GPP2I\_ADMIN\_PROHIBIT*  
*CE\_EHRPD\_VSNCP\_3GPP2I\_PDN\_ID\_IN\_USE*  
*CE\_EHRPD\_VSNCP\_3GPP2I\_SUBSCR\_LIMITATION*  
*CE\_EHRPD\_VSNCP\_3GPP2I\_PDN\_EXISTS\_FOR\_THIS\_APN*  
*CE\_EHRPD\_VSNCP\_3GPP2I\_RECONNECT\_NOT\_ALLOWED*  
*CE\_EHRPD\_UNKNOWN*

### 5.8.3.14 enum telux::data::Ipv6ReasonCode [strong]

Data call end/termination reason code for [EndReasonType::CE\\_IPV6](#)

#### Enumerator

*CE\_PREFIX\_UNAVAILABLE*  
*CE\_IPV6\_ERR\_HRPD\_IPV6\_DISABLED*  
*CE\_IPV6\_DISABLED*

**5.8.3.15 enum telux::data::HandoffReasonCode [strong]**

Data call end/termination reason code for [EndReasonType::CE\\_HANDOFF](#)

**Enumerator**

**CE\_VCER\_HANDOFF\_PREF\_SYS\_BACK\_TO\_SRAT**

**5.8.3.16 enum telux::data::ProfileChangeEvent [strong]**

Event due to which change in profile happened.

**Enumerator**

**CREATE\_PROFILE\_EVENT** Profile was created

**DELETE\_PROFILE\_EVENT** Profile was deleted

**MODIFY\_PROFILE\_EVENT** Profile was modified

**5.8.3.17 enum telux::data::OperationType [strong]**

This applies in architectures where the modem is attached to an External Application Processor(EAP). An API, like start/stop data call, INatManager, IFirewallManager can be invoked from the EAP or from the modems Internal Application Processor (IAP). This type specifies where the operation should be carried out.

**Enumerator**

**DATA\_LOCAL** Perform the operation on the processor where the API is invoked.

**DATA\_REMOTE** Perform the operation on the application processor other than where the API is invoked.

**5.8.3.18 enum telux::data::Direction [strong]**

Direction of firewall rule

**Enumerator**

**UPLINK** Uplink Direction

**DOWNLINK** Downlink Direction

**5.8.3.19 enum telux::data::InterfaceType [strong]**

Peripheral Interface type

**Enumerator**

**UNKNOWN** UNKNOWN interface

**WLAN** Wireless Local Area Network (WLAN)

**ETH** Ethernet (ETH)

**ECM** Ethernet Control Model (ECM)

**RNDIS** Remote Network Driver Interface Specification (RNDIS)

**MHI** Modem Host Interface (MHI)

### 5.8.3.20 enum telux::data::BackhaulType [strong]

Specifies backhaul types

#### Enumerator

**ETH**  
**USB** Ethernet Backhaul  
**WLAN** USB Backhaul  
**WWAN** WLAN Backhaul  
**BLE** WWAN Backhaul with default profile ID set by  
telux::data::DataConnectionManager::setDefaultProfile  
**MAX\_SUPPORTED** Bluetooth Backhaul

### 5.8.3.21 enum telux::data::LatencyLevel [strong]

Latency Level Latency level of configured IP packets. [LatencyLevel::NORMAL](#) is a default setting that has no additional priority.

#### Enumerator

**INVALID**  
**NORMAL** Not configured  
**LOW** Default, no priority

### 5.8.3.22 enum telux::data::ServiceState [strong]

State of Service

#### Enumerator

**INACTIVE** Service is inactive  
**ACTIVE** Service is Active

### 5.8.3.23 enum telux::data::QosFlowStateChangeEvent [strong]

QOS flow state change type

#### Enumerator

**UNKNOWN** UNKNOWN state  
**ACTIVATED** Flow activated  
**MODIFIED** Flow modified  
**DELETED** Flow deleted

### 5.8.3.24 enum telux::data::IpTrafficClassType [strong]

QOS flow IP traffic class type

#### Enumerator

**UNKNOWN** UNKNOWN type

**CONVERSATIONAL** Conversational IP Traffic class  
**STREAMING** Streaming IP Traffic class  
**INTERACTIVE** Interactive IP Traffic class  
**BACKGROUND** Background IP Traffic class

### 5.8.3.25 enum telux::data::QosIPFlowMaskType

Specifies QOS IP Flow parameter mask

#### Enumerator

**MASK\_IP\_FLOW\_NONE**  
**MASK\_IP\_FLOW\_TRF\_CLASS** No parameters set  
**MASK\_IP\_FLOW\_DATA\_RATE\_MIN\_MAX** Traffic class

### 5.8.3.26 enum telux::data::QosFlowMaskType

Specifies QOS Flow parameter mask

#### Enumerator

**MASK\_FLOW\_NONE**  
**MASK\_FLOW\_TX\_GRANTED** No parameters set  
**MASK\_FLOW\_RX\_GRANTED** TX Granted flow set  
**MASK\_FLOW\_TX\_FILTERS** RX Granted flow set  
**MASK\_FLOW\_RX\_FILTERS** TX filters set

### 5.8.3.27 enum telux::data::RoamingType [strong]

Roaming Type.

#### Enumerator

**UNKNOWN** Device roaming mode is unknown  
**DOMESTIC** Device is in Domestic roaming network  
**INTERNATIONAL** Device is in International roaming network

### 5.8.3.28 enum telux::data::DataServiceState [strong]

Data Service State. Indicates whether data service is ready to setup a data call or not.

#### Enumerator

**UNKNOWN** Service State not available  
**IN\_SERVICE** Service Available  
**OUT\_OF\_SERVICE** Service Not Available

### 5.8.3.29 enum telux::data::NetworkRat [strong]

Data Network RATs.

**Enumerator**

**UNKNOWN** UNKNOWN  
**CDMA\_1X** CDMA\_1X  
**CDMA\_EVDO** CDMA\_EVDO  
**GSM** GSM  
**WCDMA** WCDMA  
**LTE** LTE  
**TDSCDMA** TDSCDMA  
**NR5G** NR5G



## 5.9 Subscription Management

This section contains APIs related to Subscription Management.

### 5.9.1 Data Structure Documentation

#### 5.9.1.1 class telux::tel::ISubscription

Subscription returns information about network operator subscription details pertaining to a SIM card.

##### Public member functions

- virtual std::string [getCarrierName](#) ()=0
- virtual std::string [getIccId](#) ()=0
- virtual int [getMcc](#) ()=0
- virtual int [getMnc](#) ()=0
- virtual std::string [getMobileCountryCode](#) ()=0
- virtual std::string [getMobileNetworkCode](#) ()=0
- virtual std::string [getPhoneNumber](#) ()=0
- virtual int [getSlotId](#) ()=0
- virtual std::string [getImsi](#) ()=0
- virtual std::string [getGID1](#) ()=0
- virtual std::string [getGID2](#) ()=0
- virtual [~ISubscription](#) ()

##### 5.9.1.1.1 Constructors and Destructors

5.9.1.1.1.1 virtual telux::tel::ISubscription::~ISubscription ( ) [virtual]

##### 5.9.1.1.2 Member Function Documentation

5.9.1.1.2.1 virtual std::string telux::tel::ISubscription::getCarrierName ( ) [pure virtual]

Retrieves the name of the carrier on which this subscription is made.

##### Returns

Name of the carrier.

**5.9.1.1.2.2 virtual std::string telux::tel::ISubscription::getIccid ( ) [pure virtual]**

Retrieves the SIM's ICCID (Integrated Chip ID) - i.e SIM Serial Number.

**Returns**

Integrated Chip Id.

**5.9.1.1.2.3 virtual int telux::tel::ISubscription::getMcc ( ) [pure virtual]**

Retrieves the mobile country code of the carrier to which the phone is connected.

**Returns**

Mobile Country Code.

**Deprecated** Use [telux::tel::ISubscription::getMobileCountryCode\(\)](#) API instead

**5.9.1.1.2.4 virtual int telux::tel::ISubscription::getMnc ( ) [pure virtual]**

Retrieves the mobile network code of the carrier to which phone is connected.

**Returns**

Mobile Network Code.

**Deprecated** Use [telux::tel::ISubscription::getMobileNetworkCode\(\)](#) API instead

**5.9.1.1.2.5 virtual std::string telux::tel::ISubscription::getMobileCountryCode ( ) [pure virtual]**

Retrieves the mobile country code(MCC) of the carrier to which the phone is connected.

**Returns**

mcc.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.9.1.1.2.6 virtual std::string telux::tel::ISubscription::getMobileNetworkCode ( ) [pure virtual]**

Retrieves the mobile network code(MNC) of the carrier to which the phone is connected.

**Returns**

mnc.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.9.1.1.2.7 virtual std::string telx::tel::ISubscription::getPhoneNumber ( ) [pure virtual]**

Retrieves the phone number for the SIM subscription.

**Returns**

PhoneNumber.

**5.9.1.1.2.8 virtual int telx::tel::ISubscription::getSlotId ( ) [pure virtual]**

Retrieves SIM Slot index for the SIM pertaining to this subscription object.

**Returns**

SIM slotId.

**5.9.1.1.2.9 virtual std::string telx::tel::ISubscription::getImsi ( ) [pure virtual]**

Retrieves IMSI (International Mobile Subscriber Identity) for the SIM. This will have home network MCC and MNC values.

**Returns**

imsi.

**5.9.1.1.2.10 virtual std::string telx::tel::ISubscription::getGID1 ( ) [pure virtual]**

Retrieves the GID1(group identifier level1) on the SIM. It represents identifier for particular SIM and ME associations. It can be used to identify a group of SIMs for a particular application. Defined in 3GPP Spec 131.102 section 4.2.10

**Returns**

GID1 content in hex format.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.9.1.1.2.11 virtual std::string telux::tel::ISubscription::getGID2 ( ) [pure virtual]**

Retrieves the GID2(group identifier level2) content on the SIM. It represents identifier for particular SIM and ME associations. It can be used to identify a group of SIMs for a particular application. Defined in 3GPP Spec 131.102 section 4.2.11

**Returns**

GID2 content in hex format.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.9.1.2 class telux::tel::ISubscriptionListener**

A listener class for receiving device subscription information. The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

**Public member functions**

- virtual void [onSubscriptionInfoChanged](#) (std::shared\_ptr< [ISubscription](#) > subscription)
- virtual void [onNumberOfSubscriptionsChanged](#) (int count)
- virtual [~ISubscriptionListener](#) ()

**5.9.1.2.1 Constructors and Destructors****5.9.1.2.1.1 virtual telux::tel::ISubscriptionListener::~ISubscriptionListener ( ) [virtual]****5.9.1.2.2 Member Function Documentation****5.9.1.2.2.1 virtual void telux::tel::ISubscriptionListener::onSubscriptionInfoChanged ( std::shared\_ptr< [ISubscription](#) > *subscription* ) [virtual]**

This function is called whenever there is a change in Subscription details.

**Parameters**

in	<i>subscription</i>	Pointer to <a href="#">ISubscription</a> Object.
----	---------------------	--

**5.9.1.2.2.2 virtual void telux::tel::ISubscriptionListener::onNumberOfSubscriptionsChanged ( int *count* ) [virtual]**

This function called whenever there is a change in the subscription count. for example when a new subscription is discovered or an existing subscription goes away when SIM is inserted or removed respectively.

in	count	count of subscription
----	-------	-----------------------

### 5.9.1.3 class telux::tel::ISubscriptionManager

#### Public member functions

- virtual bool `isSubsystemReady ()=0`
- virtual `std::future< bool > onSubsystemReady ()=0`
- virtual `std::shared_ptr< ISubscription > getSubscription (int slotId=DEFAULT_SLOT_ID, telux::common::Status *status=nullptr)=0`
- virtual `std::vector< std::shared_ptr< ISubscription > > getAllSubscriptions (telux::common::Status *status=nullptr)=0`
- virtual `telux::common::Status registerListener (std::weak_ptr< ISubscriptionListener > listener)=0`
- virtual `telux::common::Status removeListener (std::weak_ptr< ISubscriptionListener > listener)=0`
- virtual `~ISubscriptionManager ()`

#### 5.9.1.3.1 Constructors and Destructors

5.9.1.3.1.1 virtual `telux::tel::ISubscriptionManager::~ISubscriptionManager ( ) [virtual]`

#### 5.9.1.3.2 Member Function Documentation

5.9.1.3.2.1 virtual bool `telux::tel::ISubscriptionManager::isSubsystemReady ( ) [pure virtual]`

Checks the status of SubscriptionManager and returns the result.

#### Returns

If true then SubscriptionManager is ready for service.

5.9.1.3.2.2 virtual `std::future<bool> telux::tel::ISubscriptionManager::onSubsystemReady ( ) [pure virtual]`

Wait for Subscription subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when SubscriptionManager is ready.

5.9.1.3.2.3 virtual `std::shared_ptr<ISubscription> telux::tel::ISubscriptionManager::getSubscription ( int slotId = DEFAULT_SLOT_ID, telux::common::Status * status = nullptr ) [pure virtual]`

Get Subscription details of the SIM in the given SIM slot.

in	<i>slotId</i>	Slot id corresponding to the subscription.
out	<i>status</i>	Status of getSubscription i.e. success or suitable status code.

**Returns**

Pointer to [ISubscription](#) object.

**5.9.1.3.2.4** `virtual std::vector<std::shared_ptr<ISubscription> > telux::tel::ISubscriptionManager::getAllSubscriptions ( telux::common::Status * status = nullptr ) [pure virtual]`

Get all the subscription details of the device.

**Parameters**

out	<i>status</i>	Status of getAllSubscriptions i.e. success or suitable status code.
-----	---------------	---

**Returns**

list of [ISubscription](#) objects.

**5.9.1.3.2.5** `virtual telux::common::Status telux::tel::ISubscriptionManager::registerListener ( std::weak_ptr< ISubscriptionListener > listener ) [pure virtual]`

Register a listener for Subscription events.

**Parameters**

in	<i>listener</i>	Pointer to <a href="#">ISubscriptionListener</a> object that processes the notification.
----	-----------------	--

**Returns**

Status of registerListener i.e. success or suitable status code.

**5.9.1.3.2.6** `virtual telux::common::Status telux::tel::ISubscriptionManager::removeListener ( std::weak_ptr< ISubscriptionListener > listener ) [pure virtual]`

Remove a previously added listener.

**Parameters**

in	<i>listener</i>	Pointer to <a href="#">ISubscriptionListener</a> object that needs to be removed.
----	-----------------	---

**Returns**

Status of removeListener i.e. success or suitable status code.

## 5.10 Network Selection

Network Selection Manager provides the interface to get and set network selection mode (Manual or Automatic), scan available networks and set and get preferred networks list.

### 5.10.1 Data Structure Documentation

#### 5.10.1.1 struct telux::tel::PreferredNetworkInfo

Defines the preferred network information

##### Data fields

Type	Field	Description
uint16_t	mcc	mobile country code
uint16_t	mnc	mobile network code
<a href="#">RatMask</a>	ratMask	bit mask denotes which of the radio access technologies are set

#### 5.10.1.2 struct telux::tel::OperatorStatus

Defines status of network operator

##### Data fields

Type	Field	Description
<a href="#">InUseStatus</a>	inUse	In-use status of network operator
<a href="#">RoamingStatus</a>	roaming	Roaming status of network operator
<a href="#">Forbidden↔ Status</a>	forbidden	Forbidden status of network operator
<a href="#">PreferredStatus</a>	preferred	Preferred status of network operator

#### 5.10.1.3 class telux::tel::NetworkSelectionManager

Network Selection Manager class provides the interface to get and set network selection mode, preferred network list and scan available networks.

##### Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status requestNetworkSelectionMode](#) ([SelectionModeResponseCallback](#) callback)=0
- virtual [telux::common::Status setNetworkSelectionMode](#) ([NetworkSelectionMode](#) selectMode, std::string mcc, std::string mnc, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status requestPreferredNetworks](#) ([PreferredNetworksCallback](#) callback)=0
- virtual [telux::common::Status setPreferredNetworks](#) (std::vector< [PreferredNetworkInfo](#) > preferredNetworksInfo, bool clearPrevious, [common::ResponseCallback](#) callback=nullptr)=0

- virtual `telux::common::Status performNetworkScan (NetworkScanCallback callback)=0`
- virtual `telux::common::Status registerListener (std::weak_ptr< INetworkSelectionListener > listener)=0`
- virtual `telux::common::Status deregisterListener (std::weak_ptr< INetworkSelectionListener > listener)=0`
- virtual `~INetworkSelectionManager ()`

### 5.10.1.3.1 Constructors and Destructors

5.10.1.3.1.1 `virtual telux::tel::INetworkSelectionManager::~~INetworkSelectionManager ( ) [virtual]`

### 5.10.1.3.2 Member Function Documentation

5.10.1.3.2.1 `virtual bool telux::tel::INetworkSelectionManager::isSubsystemReady ( ) [pure virtual]`

Checks the status of network subsystem and returns the result.

#### Returns

True if network subsystem is ready for service otherwise false.

5.10.1.3.2.2 `virtual std::future<bool> telux::tel::INetworkSelectionManager::onSubsystemReady ( ) [pure virtual]`

Wait for network subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when network subsystem is ready.

5.10.1.3.2.3 `virtual telux::common::Status telux::tel::INetworkSelectionManager::request↔ NetworkSelectionMode ( SelectionModeResponseCallback callback ) [pure virtual]`

Get current network selection mode (i.e Manual or Automatic) asynchronously.

#### Parameters

<code>in</code>	<code>callback</code>	Callback function to get the response of get network selection mode request.
-----------------	-----------------------	--

#### Returns

Status of requestNetworkSelectionMode i.e. success or suitable error code.



**5.10.1.3.2.4** `virtual telux::common::Status telux::tel::INetworkSelectionManager::setNetworkSelectionMode ( NetworkSelectionMode selectMode, std::string mcc, std::string mnc, common::ResponseCallback callback = nullptr ) [pure virtual]`

Set current network selection mode and receive the response asynchronously.

#### Parameters

in	<i>selectMode</i>	Selection mode for a network i.e. automatic or manual. If selection mode is automatic then MCC and MNC are ignored. If it is manual, client has to explicitly pass MCC and MNC as arguments.
in	<i>callback</i>	Optional callback function to get the response of set network selection mode request.
in	<i>mcc</i>	Mobile Country Code (Applicable only for MANUAL selection mode).
in	<i>mnc</i>	Mobile Network Code (Applicable only for MANUAL selection mode).

#### Returns

Status of setNetworkSelectionMode i.e. success or suitable error code.

**5.10.1.3.2.5** `virtual telux::common::Status telux::tel::INetworkSelectionManager::requestPreferredNetworks ( PreferredNetworksCallback callback ) [pure virtual]`

Get 3GPP preferred network list and static 3GPP preferred network list asynchronously. Higher priority networks appear first in the list. The networks that appear in the 3GPP Preferred Networks list get higher priority than the networks in the static 3GPP preferred networks list.

#### Parameters

in	<i>callback</i>	Callback function to get the response of get preferred networks request.
----	-----------------	--

#### Returns

Status of requestPreferredNetworks i.e. success or suitable error code.

**5.10.1.3.2.6** `virtual telux::common::Status telux::tel::INetworkSelectionManager::setPreferredNetworks ( std::vector< PreferredNetworkInfo > preferredNetworksInfo, bool clearPrevious, common::ResponseCallback callback = nullptr ) [pure virtual]`

Set 3GPP preferred network list and receive the response asynchronously. It overrides the existing preferred network list. The preferred network list affects network selection selection when automatic registration is performed by the device. Higher priority networks should appear first in the list.

in	<i>preferredNetworks</i> ↔ <i>Info</i>	List of 3GPP preferred networks.
in	<i>clearPrevious</i>	If flag is false then new 3GPP preferred network list is appended to existing preferred network list. If flag is true then old list is flushed and new 3GPP preferred network list is added.
in	<i>callback</i>	Callback function to get the response of set preferred network list request.

**Returns**

Status of setPreferredNetworks i.e. success or suitable error code.

#### 5.10.1.3.2.7 virtual telux::common::Status telux::tel::INetworkSelectionManager::performNetworkScan ( NetworkScanCallback *callback* ) [pure virtual]

Perform the network scan and returns a list of available networks.

**Parameters**

in	<i>callback</i>	Callback function to get the response of perform network scan request
----	-----------------	---

**Returns**

Status of performNetworkScan i.e. success or suitable error code.

#### 5.10.1.3.2.8 virtual telux::common::Status telux::tel::INetworkSelectionManager::registerListener ( std::weak\_ptr< INetworkSelectionListener > *listener* ) [pure virtual]

Register a listener for specific updates from network access service.

**Parameters**

in	<i>listener</i>	Pointer of <a href="#">INetworkSelectionListener</a> object that processes the notification
----	-----------------	---

**Returns**

Status of registerListener i.e success or suitable status code.

#### 5.10.1.3.2.9 virtual telux::common::Status telux::tel::INetworkSelectionManager::deregisterListener ( std::weak\_ptr< INetworkSelectionListener > *listener* ) [pure virtual]

Deregister the previously added listener.

**Parameters**

<code>in</code>	<i>listener</i>	Previously registered <a href="#">INetworkSelectionListener</a> that needs to be removed
-----------------	-----------------	--

**Returns**

Status of removeListener success or suitable status code

**5.10.1.4 class telux::tel::OperatorInfo**

Operator Info class provides operator name, MCC, MNC and network status.

**Public member functions**

- [OperatorInfo](#) (std::string networkName, std::string mcc, std::string mnc, [OperatorStatus](#) operatorStatus)
- std::string [getName](#) ()
- std::string [getMcc](#) ()
- std::string [getMnc](#) ()
- [OperatorStatus](#) [getStatus](#) ()

**5.10.1.4.1 Constructors and Destructors**

**5.10.1.4.1.1** `telux::tel::OperatorInfo::OperatorInfo ( std::string networkName, std::string mcc, std::string mnc, OperatorStatus operatorStatus )`

**5.10.1.4.2 Member Function Documentation**

**5.10.1.4.2.1** `std::string telux::tel::OperatorInfo::getName ( )`

Get Operator name or description

**Returns**

Operator name.

**5.10.1.4.2.2** `std::string telux::tel::OperatorInfo::getMcc ( )`

Get mcc from the operator numeric.

**Returns**

MCC.

**5.10.1.4.2.3 std::string telx::tel::OperatorInfo::getMnc ( )**

Get mnc from operator numeric.

**Returns**

MNC.

**5.10.1.4.2.4 OperatorStatus telx::tel::OperatorInfo::getStatus ( )**

Get status of operator.

**Returns**

status of the operator [OperatorStatus](#).

**5.10.1.5 class telx::tel::INetworkSelectionListener**

Listener class for getting network selection mode change notification.

The methods in listener can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe.

**Public member functions**

- virtual void [onSelectionModeChanged](#) ([NetworkSelectionMode](#) mode)
- virtual [~INetworkSelectionListener](#) ()

**5.10.1.5.1 Constructors and Destructors****5.10.1.5.1.1 virtual telx::tel::INetworkSelectionListener::~~INetworkSelectionListener ( ) [virtual]**

Destructor of [INetworkSelectionListener](#)

**5.10.1.5.2 Member Function Documentation****5.10.1.5.2.1 virtual void telx::tel::INetworkSelectionListener::onSelectionModeChanged ( [NetworkSelectionMode](#) mode ) [virtual]**

This function is called whenever network selection mode is changed.

**Parameters**

in	<i>mode</i>	Network selection mode <a href="#">NetworkSelectionMode</a>
----	-------------	---

## 5.10.2 Enumeration Type Documentation

### 5.10.2.1 enum telux::tel::RatType

Defines network RAT type for preferred networks. Each value represents corresponding bit for RatMask bitset.

#### Enumerator

**UMTS** UMTS  
**LTE** LTE  
**LTE**  
**GSM** GSM  
**GSM**  
**NR5G** NR5G

### 5.10.2.2 enum telux::tel::NetworkSelectionMode [strong]

Defines network selection mode

#### Enumerator

**UNKNOWN** Unknown  
**AUTOMATIC** Device registers according to provisioned mcc and mnc  
**MANUAL** Device registers to specified network as per provided mcc and mnc

### 5.10.2.3 enum telux::tel::InUseStatus [strong]

Defines in-use status of network operator

#### Enumerator

**UNKNOWN** Unknown  
**CURRENT\_SERVING** Current serving  
**AVAILABLE** Available

### 5.10.2.4 enum telux::tel::RoamingStatus [strong]

Defines roaming status of network operator

#### Enumerator

**UNKNOWN** Unknown  
**HOME** Home  
**ROAM** Roaming

### 5.10.2.5 enum telux::tel::ForbiddenStatus [strong]

Defines forbidden status of network operator

**Enumerator**

**UNKNOWN** Unknown  
**FORBIDDEN** Forbidden  
**NOT\_FORBIDDEN** Not forbidden

**5.10.2.6 enum telx::tel::PreferredStatus [strong]**

Defines preferred status of network operator

**Enumerator**

**UNKNOWN** Unknown  
**PREFERRED** Preferred  
**NOT\_PREFERRED** Not preferred

## 5.11 Serving System

Serving System Manager class provides the interface to request and set service domain preference and radio access technology mode preference for searching and registering (CS/PS domain, RAT and operation mode)

### 5.11.1 Data Structure Documentation

#### 5.11.1.1 struct telux::tel::DcStatus

Defines Dual Connectivity status

##### Data fields

Type	Field	Description
<a href="#">Endc↔Availability</a>	endc↔Availability	ENDC availability
<a href="#">DcnrRestriction</a>	dcnrRestriction	DCNR restriction

#### 5.11.1.2 class telux::tel::IServingSystemManager

##### Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status setRatPreference](#) ([RatPreference](#) ratPref, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status requestRatPreference](#) ([RatPreferenceCallback](#) callback)=0
- virtual [telux::common::Status setServiceDomainPreference](#) ([ServiceDomainPreference](#) serviceDomain, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status requestServiceDomainPreference](#) ([ServiceDomainPreferenceCallback](#) callback)=0
- virtual [telux::tel::DcStatus getDcStatus](#) ()=0
- virtual [telux::common::Status requestNetworkTime](#) ([NetworkTimeResponseCallback](#) callback)=0
- virtual [telux::common::Status registerListener](#) (std::weak\_ptr< [IServingSystemListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak\_ptr< [IServingSystemListener](#) > listener)=0
- virtual [~IServingSystemManager](#) ()

##### 5.11.1.2.1 Constructors and Destructors

**5.11.1.2.1.1 virtual telux::tel::IServingSystemManager::~~IServingSystemManager ( ) [virtual]**

Destructor of [IServingSystemManager](#)

### 5.11.1.2.2 Member Function Documentation

**5.11.1.2.2.1 virtual bool telux::tel::IServingSystemManager::isSubsystemReady ( ) [pure virtual]**

Checks the status of serving subsystem and returns the result.

#### Returns

True if serving subsystem is ready for service otherwise false.

**5.11.1.2.2.2 virtual std::future<bool> telux::tel::IServingSystemManager::onSubsystemReady ( ) [pure virtual]**

Wait for serving subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when serving subsystem is ready.

**5.11.1.2.2.3 virtual telux::common::Status telux::tel::IServingSystemManager::setRatPreference ( RatPreference *ratPref*, common::ResponseCallback *callback* = *nullptr* ) [pure virtual]**

Set the preferred radio access technology mode that the device should use to acquire service.

#### Parameters

in	<i>ratPref</i>	Radio access technology mode preference.
in	<i>callback</i>	Callback function to get the response of set RAT mode preference.

#### Returns

Status of setRatPreference i.e. success or suitable error code.

**5.11.1.2.2.4 virtual telux::common::Status telux::tel::IServingSystemManager::requestRatPreference ( RatPreferenceCallback *callback* ) [pure virtual]**

Request for preferred radio access technology mode.



**Parameters**

in	<i>callback</i>	Callback function to get the response of request preferred RAT mode.
----	-----------------	--

**Returns**

Status of requestRatPreference i.e. success or suitable error code.

**5.11.1.2.2.5** `virtual telux::common::Status telux::tel::IServingSystemManager::setServiceDomainPreference ( ServiceDomainPreference serviceDomain, common::ResponseCallback callback = nullptr ) [pure virtual]`

Initiate service domain preference like CS, PS or CS\_PS and receive the response asynchronously.

**Parameters**

in	<i>serviceDomain</i>	<a href="#">ServiceDomainPreference</a> .
in	<i>callback</i>	Callback function to get the response of set service domain preference request.

**Returns**

Status of setServiceDomainPreference i.e. success or suitable error code.

**5.11.1.2.2.6** `virtual telux::common::Status telux::tel::IServingSystemManager::requestServiceDomainPreference ( ServiceDomainPreferenceCallback callback ) [pure virtual]`

Request for Service Domain Preference asynchronously.

**Parameters**

in	<i>callback</i>	Callback function to get the response of request service domain preference.
----	-----------------	---

**Returns**

Status of requestServiceDomainPreference i.e. success or suitable error code.

**5.11.1.2.2.7** `virtual telux::tel::DcStatus telux::tel::IServingSystemManager::getDcStatus ( ) [pure virtual]`

Request for Dual Connectivity status on 5G NR.

**Returns**

[DcStatus](#)

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.11.1.2.2.8 virtual telux::common::Status telux::tel::IServingSystemManager::requestNetworkTime ( NetworkTimeResponseCallback *callback* ) [pure virtual]

Get network time information asynchronously.

**Parameters**

in	<i>callback</i>	Callback function to get the response of get network time information request.
----	-----------------	--

**Returns**

Status of requestNetworkTime i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.11.1.2.2.9 virtual telux::common::Status telux::tel::IServingSystemManager::registerListener ( std::weak\_ptr< IServingSystemListener > *listener* ) [pure virtual]

Register a listener for specific updates from serving system.

**Parameters**

in	<i>listener</i>	Pointer of <a href="#">IServingSystemListener</a> object that processes the notification
----	-----------------	--

**Returns**

Status of registerListener i.e success or suitable status code.

#### 5.11.1.2.2.10 virtual telux::common::Status telux::tel::IServingSystemManager::deregisterListener ( std::weak\_ptr< IServingSystemListener > *listener* ) [pure virtual]

Deregister the previously added listener.

**Parameters**

in	<i>listener</i>	Previously registered <a href="#">IServingSystemListener</a> that needs to be removed
----	-----------------	---

## Returns

Status of removeListener i.e. success or suitable status code

### 5.11.1.3 class telux::tel::IServingSystemListener

Listener class for getting radio access technology mode preference change notification.

The listener method can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe.

#### Public member functions

- virtual void [onRatPreferenceChanged](#) ([RatPreference](#) preference)
- virtual void [onServiceDomainPreferenceChanged](#) ([ServiceDomainPreference](#) preference)
- virtual void [onDcStatusChanged](#) ([DcStatus](#) dcStatus)
- virtual void [onNetworkTimeChanged](#) ([NetworkTimeInfo](#) info)
- virtual [~IServingSystemListener](#) ()

#### 5.11.1.3.1 Constructors and Destructors

5.11.1.3.1.1 virtual telux::tel::IServingSystemListener::~~IServingSystemListener ( ) [virtual]

Destructor of [IServingSystemListener](#)

#### 5.11.1.3.2 Member Function Documentation

5.11.1.3.2.1 virtual void telux::tel::IServingSystemListener::onRatPreferenceChanged ( [RatPreference](#) *preference* ) [virtual]

This function is called whenever RAT mode preference is changed.

#### Parameters

in	<i>preference</i>	<a href="#">RatPreference</a>
----	-------------------	-------------------------------

5.11.1.3.2.2 virtual void telux::tel::IServingSystemListener::onServiceDomainPreferenceChanged ( [ServiceDomainPreference](#) *preference* ) [virtual]

This function is called whenever service domain preference is changed.

#### Parameters

in	<i>preference</i>	<a href="#">ServiceDomainPreference</a>
----	-------------------	---

### 5.11.1.3.2.3 virtual void telux::tel::IServingSystemListener::onDcStatusChanged ( DcStatus *dcStatus* ) [virtual]

This function is called whenever the Dual Connectivity status is changed on 5G NR.

#### Parameters

in	<i>dcStatus</i>	DcStatus
----	-----------------	----------

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.11.1.3.2.4 virtual void telux::tel::IServingSystemListener::onNetworkTimeChanged ( NetworkTimeInfo *info* ) [virtual]

This function is called whenever network time information is changed.

#### Parameters

in	<i>info</i>	Network time information <a href="#">NetworkTimeInfo</a>
----	-------------	--

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

## 5.11.2 Enumeration Type Documentation

### 5.11.2.1 enum telux::tel::ServiceDomainPreference [strong]

Defines service domain preference

#### Enumerator

**UNKNOWN**  
**CS\_ONLY** Circuit-switched only  
**PS\_ONLY** Packet-switched only  
**CS\_PS** Circuit-switched and packet-switched

### 5.11.2.2 enum telux::tel::RatPrefType

Defines the radio access technology mode preference.

#### Enumerator

**PREF\_CDMA\_1X** CDMA\_1X  
**PREF\_CDMA\_EVDO** CDMA\_EVDO  
**PREF\_GSM** GSM  
**PREF\_WCDMA** WCDMA

**PREF\_LTE** LTE  
**PREF\_TDSCDMA** TDSCDMA  
**PREF\_NR5G** NR5G

### 5.11.2.3 enum telux::tel::EndcAvailability [strong]

Defines ENDC(E-UTRAN New Radio-Dual Connectivity) Availability status on 5G NR

#### Enumerator

**UNKNOWN** Status unknown  
**AVAILABLE** ENDC is Available  
**UNAVAILABLE** ENDC is not Available

### 5.11.2.4 enum telux::tel::DcnrRestriction [strong]

Defines DCNR(Dual Connectivity with NR) Restriction status on 5G NR

#### Enumerator

**UNKNOWN** Status unknown  
**RESTRICTED** DCNR is Rescticted  
**UNRESTRICTED** DCNR is not Restricted

## 5.12 Common

This section contains APIs related to Command Callbacks, Error Codes and [Version](#) information.

### 5.12.1 Data Structure Documentation

#### 5.12.1.1 class `telux::common::ICommandCallback`

Base command callback class is responsible for single shot asynchronous callback. This callback will be invoked only once when the operation succeeds or fails.

##### Public member functions

- virtual `~ICommandCallback ()`

##### 5.12.1.1.1 Constructors and Destructors

5.12.1.1.1 virtual `telux::common::ICommandCallback::~ICommandCallback ( ) [virtual]`

#### 5.12.1.2 class `telux::common::ICommandResponseCallback`

General command response callback for most of the requests, client needs to implement this interface to get single shot response.

The methods in callback can be invoked from multiple different threads. The implementation should be thread safe.

##### Public member functions

- virtual void `commandResponse (ErrorCode error)=0`
- virtual `~ICommandResponseCallback ()`

##### 5.12.1.2.1 Constructors and Destructors

5.12.1.2.1.1 virtual `telux::common::ICommandResponseCallback::~ICommandResponseCallback ( ) [virtual]`

##### 5.12.1.2.2 Member Function Documentation

5.12.1.2.2.1 virtual void `telux::common::ICommandResponseCallback::commandResponse ( ErrorCode error ) [pure virtual]`

This function is called with the response to the command operation.

##### Parameters

<i>in</i>	<i>error</i>	- <a href="#">ErrorCode</a>
-----------	--------------	-----------------------------

### 5.12.1.3 class telux::common::DeviceConfig

#### Static Public Member Functions

- static bool [isMultiSimSupported](#) ()

#### 5.12.1.3.1 Member Function Documentation

##### 5.12.1.3.1.1 static bool telux::common::DeviceConfig::isMultiSimSupported ( ) [static]

Check whether multi SIM support available.

#### Returns

bool to determine multi SIM support

### 5.12.1.4 class telux::common::Log

#### Static Public Member Functions

- template<typename... MessageArgs>  
static void [logMessage](#) ([LogLevel](#) logLevel, const std::string &fileName, const std::string &lineNo, const int &component, MessageArgs...params)

#### 5.12.1.4.1 Member Function Documentation

##### 5.12.1.4.1.1 template<typename... MessageArgs> static void telux::common::Log::logMessage ( [LogLevel](#) *logLevel*, const std::string & *fileName*, const std::string & *lineNo*, const int & *component*, MessageArgs... *params* ) [static]

### 5.12.1.5 struct telux::common::SdkVersion

Structure of major, minor and patch version

#### Data fields

Type	Field	Description
int	major	Major <a href="#">Version</a> : This number will be incremented whenever significant changes or features are introduced
int	minor	Minor <a href="#">Version</a> : This number will be incremented when smaller features with some new APIs are introduced.
int	patch	Patch <a href="#">Version</a> : If the release only contains bug fixes, but no API change then the patch version would be incremented.

### 5.12.1.6 class telux::common::Version

Provides version of SDK.

## Static Public Member Functions

- static std::string [getReleaseName](#) ()
- static [SdkVersion](#) [getSdkVersion](#) ()

### 5.12.1.6.1 Member Function Documentation

#### 5.12.1.6.1.1 static std::string telux::common::Version::getReleaseName ( ) [static]

Get the release name.

#### Returns

String contains release name

#### 5.12.1.6.1.2 static SdkVersion telux::common::Version::getSdkVersion ( ) [static]

Get the Telematics SDK version, for example: 01.00.00

#### Returns

[SdkVersion](#) structure of major, minor and patch version

## 5.12.2 Enumeration Type Documentation

### 5.12.2.1 enum telux::common::Status [strong]

Defines all the status codes that all Telematics SDK APIs can return

#### Enumerator

**SUCCESS** API processing is successful, returned parameters are valid

**FAILED** API processing failure.

**NOCONNECTION** Connection to Socket server has not been established

**NOSUBSCRIPTION** Subscription not available

**INVALIDPARAM** Input parameters are invalid

**INVALIDSTATE** Invalid State

**NOTREADY** Subsystem is not ready

**NOTALLOWED** Operation not allowed

**NOTIMPLEMENTED** Functionality not implemented

**CONNECTIONLOST** Connection to Socket server lost

**EXPIRED** Expired

**ALREADY** Already registered handler

**NOSUCH** No such object

**NOTSUPPORTED** Not supported on target platform

**NOMEMORY** Not sufficient memory to process the request



### 5.12.2.2 enum telux::common::ErrorCode [strong]

Generic Error code for each API responses

#### Enumerator

**SUCCESS** No error

**RADIO\_NOT\_AVAILABLE** If radio did not start or is resetting

**GENERIC\_FAILURE** Generic Failure

**PASSWORD\_INCORRECT** For PIN/PIN2 methods only

**SIM\_PIN2** Operation requires SIM PIN2 to be entered

**SIM\_PUK2** Operation requires SIM PIN2 to be entered

**REQUEST\_NOT\_SUPPORTED** Not Supported request

**CANCELLED** Cancelled

**OP\_NOT\_ALLOWED\_DURING\_VOICE\_CALL** Data operation are not allowed during voice call on a Class C GPRS device

**OP\_NOT\_ALLOWED\_BEFORE\_REG\_TO\_NW** Data operation are not allowed before device registers in network

**SMS\_SEND\_FAIL\_RETRY** Fail to send SMS and need retry

**SIM\_ABSENT** Fail to set the location where CDMA subscription shall be retrieved because of SIM or RUIM are absent

**SUBSCRIPTION\_NOT\_AVAILABLE** Fail to find CDMA subscription from specified location

**MODE\_NOT\_SUPPORTED** Hardware does not support preferred network type

**FDN\_CHECK\_FAILURE** Command failed because recipient is not on FDN list

**ILLEGAL\_SIM\_OR\_ME** Network selection failed due to illegal SIM or ME

**MISSING\_RESOURCE** No logical channel available

**NO\_SUCH\_ELEMENT** Application not found on SIM

**DIAL\_MODIFIED\_TO USSD** DIAL request modified to USSD

**DIAL\_MODIFIED\_TO\_SS** DIAL request modified to SS

**DIAL\_MODIFIED\_TO\_DIAL** DIAL request modified to DIAL with different data

**USSD\_MODIFIED\_TO\_DIAL** USSD request modified to DIAL

**USSD\_MODIFIED\_TO\_SS** USSD request modified to SS

**USSD\_MODIFIED\_TO USSD** USSD request modified to different USSD request

**SS\_MODIFIED\_TO\_DIAL** SS request modified to DIAL

**SS\_MODIFIED\_TO USSD** SS request modified to USSD

**SUBSCRIPTION\_NOT\_SUPPORTED** Subscription not supported

**SS\_MODIFIED\_TO\_SS** SS request modified to different SS request

**LCE\_NOT\_SUPPORTED** LCE service not supported

**NO\_MEMORY** Not sufficient memory to process the request

**INTERNAL\_ERR** Hit unexpected vendor internal error scenario

**SYSTEM\_ERR** Hit platform or system error

**MODEM\_ERR** Hit unexpected modem error

**INVALID\_STATE** Unexpected request for the current state

**NO\_RESOURCES** Not sufficient resource to process the request

**SIM\_ERR** Received error from SIM card

**INVALID\_ARGUMENTS** Received invalid arguments in request

**INVALID\_SIM\_STATE** Cannot process the request in current SIM state

**INVALID\_MODEM\_STATE** Cannot process the request in current Modem state

**INVALID\_CALL\_ID** Received invalid call id in request

**NO\_SMS\_TO\_ACK** ACK received when there is no SMS to ack

**NETWORK\_ERR** Received error from network

**REQUEST\_RATE\_LIMITED** Operation denied due to overly-frequent requests

**SIM\_BUSY** SIM is busy

**SIM\_FULL** The target EF is full

**NETWORK\_REJECT** Request is rejected by network

**OPERATION\_NOT\_ALLOWED** Not allowed the request now

**EMPTY\_RECORD** The request record is empty

**INVALID\_SMS\_FORMAT** Invalid SMS format

**ENCODING\_ERR** Message not encoded properly

**INVALID\_SMSC\_ADDRESS** SMSC address specified is invalid

**NO\_SUCH\_ENTRY** No such entry present to perform the request

**NETWORK\_NOT\_READY** Network is not ready to perform the request

**NOT\_PROVISIONED** Device does not have this value provisioned

**NO\_SUBSCRIPTION** Device does not have subscription

**NO\_NETWORK\_FOUND** Network cannot be found

**DEVICE\_IN\_USE** Operation cannot be performed because the device is currently in use

**ABORTED** Operation aborted

**INCOMPATIBLE\_STATE** Operation cannot be performed because the device is in incompatible state

**NO\_EFFECT** Given request had to no effect

**DEVICE\_NOT\_READY** Device not ready

**MISSING\_ARGUMENTS** Missing one or more arguments

**PIN\_PERM\_BLOCKED** PIN is permanently blocked. The SIM is unusable.

**PIN\_BLOCKED** PIN is blocked. Unblock operation must be issued.

**MALFORMED\_MSG** Message was not formulated correctly by the control point or the message was corrupted during transmission

**INTERNAL** Internal error

**CLIENT\_IDS\_EXHAUSTED** Client IDs exhausted

**UNABORTABLE\_TRANSACTION** The specified transaction could not be aborted

**INVALID\_CLIENT\_ID** Could not find client's request

**NO\_THRESHOLDS** No thresholds specified in enable signal strength

**INVALID\_HANDLE** Invalid client handle was received

**INVALID\_PROFILE** Invalid profile index specified

**INVALID\_PINID** PIN in the request is invalid.

**INCORRECT\_PIN** PIN in the request is incorrect.

**CALL\_FAILED** Call origination failed in the lower layers

**OUT\_OF\_CALL** Request issued when packet data session disconnected

**MISSING\_ARG** TLV was missing in the request.

**ARG\_TOO\_LONG** Path in the request was too long.

**INVALID\_TX\_ID** The transaction ID supplied in the request does not match any pending transaction  
i.e. either the transaction was not received or it is already executed by the device

**OP\_NETWORK\_UNSUPPORTED** Selected operation is not supported by the network

**OP\_DEVICE\_UNSUPPORTED** Operation is not supported by device or SIM card

**NO\_FREE\_PROFILE** Maximum number of profiles are stored in the device and there is no more storage available to create a new profile

**INVALID\_PDP\_TYPE** PDP type specified is not supported

**INVALID\_TECH\_PREF** Invalid technology preference

**INVALID\_PROFILE\_TYPE** Invalid profile type is specified

**INVALID\_SERVICE\_TYPE** Invalid service type

**INVALID\_REGISTER\_ACTION** Invalid register action value specified in request

**INVALID\_PS\_ATTACH\_ACTION** Invalid PS attach action value specified in request

**AUTHENTICATION\_FAILED** Authentication error.

**SIM\_NOT\_INITIALIZED** PIN is not yet initialized because the SIM initialization has not finished. Try the PIN operation later.

**MAX\_QOS\_REQUESTS\_IN\_USE** Maximum QoS requests in use

**INCORRECT\_FLOW\_FILTER** Incorrect flow filter

**NETWORK\_QOS\_UNAWARE** Network QoS unaware

**INVALID\_ID** Invalid call ID was sent in the request

**REQUESTED\_NUM\_UNSUPPORTED** Requested message ID is not supported by the currently running software

**INTERFACE\_NOT\_FOUND** Cannot retrieve the FMC interface

**FLOW\_SUSPENDED** Flow suspended

**INVALID\_DATA\_FORMAT** Invalid data format

**GENERAL** General error

**UNKNOWN** Unknown error

**INVALID\_ARG** Parameters passed as input were invalid

**INVALID\_INDEX** MIP profile index is not within the valid range

**NO\_ENTRY** No message exists at the specified memory storage designation

**DEVICE\_STORAGE\_FULL** Memory storage specified in the request is full

**CAUSE\_CODE** There was an error in the request

**MESSAGE\_NOT\_SENT** Message could not be sent

**MESSAGE\_DELIVERY\_FAILURE** Message could not be delivered

**INVALID\_MESSAGE\_ID** Message ID specified for the message is invalid

**ENCODING** Message is not encoded properly

**AUTHENTICATION\_LOCK** Maximum number of authentication failures has been reached

**INVALID\_TRANSITION** Selected operating mode transition from the current operating mode is invalid

**NOT\_A\_MCAST\_IFACE** Not a MCAST interface

**MAX\_MCAST\_REQUESTS\_IN\_USE** MCAST request in use

**INVALID\_MCAST\_HANDLE** An invalid MCAST handle

**INVALID\_IP\_FAMILY\_PREF** IP family preference is invalid

**SESSION\_INACTIVE** Session inactive

**SESSION\_INVALID** Session not valid

**SESSION\_OWNERSHIP** Session ownership error

**INSUFFICIENT\_RESOURCES** Response is longer than the maximum supported size

**DISABLED** Disabled

**INVALID\_OPERATION** Device is not expecting the request.

**INVALID\_QMI\_CMD** Invalid QMI command

**TPDU\_TYPE** Message in memory contains a TPDU type that cannot be read

**SMSC\_ADDR** SMSC address specified is invalid

**INFO\_UNAVAILABLE** Information is not available

**SEGMENT\_TOO\_LONG** PRL segment size is too large

**SEGMENT\_ORDER** PRL segment order is incorrect

**BUNDLING\_NOT\_SUPPORTED** Bundling not supported

**OP\_PARTIAL\_FAILURE** Some personalization codes were set but an error prevented

**POLICY\_MISMATCH** Network policy does not match a valid NAT

**SIM\_FILE\_NOT\_FOUND** File is not present on the card.

**EXTENDED\_INTERNAL** Error from the the DS profile module, the extended error

**ACCESS\_DENIED** Access to the requested file is denied. This can occur when there is an attempt to access a PIN-protected file.

**HARDWARE\_RESTRICTED** Selected operating mode is invalid with the current wireless disable setting

**ACK\_NOT\_SENT** ACK could not be sent  
**INJECT\_TIMEOUT** Inject timeout  
**FDN\_RESTRICT** FDN restriction  
**SUPS\_FAILURE\_CAUSE** Indicates supplementary services failure information;  
**NO\_RADIO** Radio is not available  
**NOT\_SUPPORTED** Operation is not supported  
**CARD\_CALL\_CONTROL\_FAILED** SIM/R-UIM call control failed  
**NETWORK\_ABORTED** Operation was released abruptly by the network  
**MSG\_BLOCKED** Message blocked  
**INVALID\_SESSION\_TYPE** Invalid session type  
**INVALID\_PB\_TYPE** Invalid Phone Book type  
**NO\_SIM** Action is being performed on a SIM that is not initialized.  
**PB\_NOT\_READY** Phone Book not ready  
**PIN\_RESTRICTION** PIN restriction  
**PIN2\_RESTRICTION** PIN2 restriction  
**PUK\_RESTRICTION** PUK restriction  
**PUK2\_RESTRICTION** PUK2 restriction  
**PB\_ACCESS\_RESTRICTED** Phone Book access restricted  
**PB\_DELETE\_IN\_PROG** Phone Book delete in progress  
**PB\_TEXT\_TOO\_LONG** Phone Book text too long  
**PB\_NUMBER\_TOO\_LONG** Phone Book number too long  
**PB\_HIDDEN\_KEY\_RESTRICTION** Phone Book hidden key restriction  
**PB\_NOT\_AVAILABLE** Phone Book not available  
**DEVICE\_MEMORY\_ERROR** Device memory error  
**NO\_PERMISSION** No permission  
**TOO\_SOON** Too soon  
**TIME\_NOT\_ACQUIRED** Time not acquired  
**OP\_IN\_PROGRESS** Operation is in progress  
**DS\_PROFILE\_REG\_RESULT\_FAIL** General failure  
**DS\_PROFILE\_REG\_RESULT\_ERR\_INVALID\_HNDL** Request contains an invalid profile handle  
**DS\_PROFILE\_REG\_RESULT\_ERR\_INVALID\_OP** Invalid operation was requested  
**DS\_PROFILE\_REG\_RESULT\_ERR\_INVALID\_PROFILE\_TYPE** Request contains an invalid technology  
type  
**DS\_PROFILE\_REG\_RESULT\_ERR\_INVALID\_PROFILE\_NUM** Request contains an invalid profile  
number  
**DS\_PROFILE\_REG\_RESULT\_ERR\_INVALID\_IDENT** Request contains an invalid profile identifier  
**DS\_PROFILE\_REG\_RESULT\_ERR\_INVALID** Request contains an invalid argument other than profile  
number and profile identifier received  
**DS\_PROFILE\_REG\_RESULT\_ERR\_LIB\_NOT\_INITED** Profile registry has not been initialized yet  
**DS\_PROFILE\_REG\_RESULT\_ERR\_LEN\_INVALID** Request contains a parameter with invalid length  
**DS\_PROFILE\_REG\_RESULT\_LIST\_END** End of the profile list was reached while searching for the  
requested profile  
**DS\_PROFILE\_REG\_RESULT\_ERR\_INVALID\_SUBS\_ID** Request contains an invalid subscription  
identifier  
**DS\_PROFILE\_REG\_INVALID\_PROFILE\_FAMILY** Request contains an invalid profile family  
**DS\_PROFILE\_REG\_PROFILE\_VERSION\_MISMATCH** [Version](#) mismatch  
**REG\_RESULT\_ERR\_OUT\_OF\_MEMORY** Out of memory  
**DS\_PROFILE\_REG\_RESULT\_ERR\_FILE\_ACCESS** File access error  
**DS\_PROFILE\_REG\_RESULT\_ERR\_EOF** End of field  
**REG\_RESULT\_ERR\_VALID\_FLAG\_NOT\_SET** A valid flag is not set

**REG\_RESULT\_ERR\_OUT\_OF\_PROFILES** Out of profiles

**REG\_RESULT\_NO\_EMERGENCY\_PDN\_SUPPORT** No emergency PDN support

**DS\_PROFILE\_3GPP\_INVALID\_PROFILE\_FAMILY** Request contains an invalid 3GPP profile family

**DS\_PROFILE\_3GPP\_ACCESS\_ERR** Error was encountered while accessing the 3GPP profiles

**DS\_PROFILE\_3GPP\_CONTEXT\_NOT\_DEFINED** Specified 3GPP profile does not have a valid context

**DS\_PROFILE\_3GPP\_VALID\_FLAG\_NOT\_SET** Specified 3GPP profile is marked invalid

**DS\_PROFILE\_3GPP\_READ\_ONLY\_FLAG\_SET** Specified 3GPP profile is marked read-only

**DS\_PROFILE\_3GPP\_ERR\_OUT\_OF\_PROFILES** Creation of a new 3GPP profile failed because the limit of 16 profiles has already been reached

**DS\_PROFILE\_3GPP2\_ERR\_INVALID\_IDENT\_FOR\_PROFILE** Invalid profile identifier was received as part of the 3GPP2 profile modification request

**DS\_PROFILE\_3GPP2\_ERR\_OUT\_OF\_PROFILE** Creation of a new 3GPP2 profile failed because the limit has already been reached

**INTERNAL\_ERROR** Internal error

**SERVICE\_ERROR** Service error

**TIMEOUT\_ERROR** Timeout error

**EXTENDED\_ERROR** Extended error

**PORT\_NOT\_OPEN\_ERROR** Port not open

**MEMCOPY\_ERROR** Memory copy error

**INVALID\_TRANSACTION** Invalid transaction

**ALLOCATION\_FAILURE** Allocation failure

**TRANSPORT\_ERROR** Transport error

**PARAM\_ERROR** Parameter error

**INVALID\_CLIENT** Invalid client

**FRAMEWORK\_NOT\_READY** Framework not ready

**INVALID\_SIGNAL** Invalid signal

**TRANSPORT\_BUSY\_ERROR** Transport busy error

**SUBSYSTEM\_UNAVAILABLE** Underlying service currently unavailable

**OPERATION\_TIMEOUT** Timeout error

### 5.12.2.3 enum telux::common::ServiceStatus [strong]

Service status.

#### Enumerator

**SERVICE\_UNAVAILABLE**  
**SERVICE\_AVAILABLE**  
**SERVICE\_FAILED**

### 5.12.2.4 enum telux::common::LogLevel [strong]

Indicates supported logging levels.

#### Enumerator

**LEVEL\_NONE**  
**LEVEL\_PERF** Prints messages with nanoseconds precision timestamp  
**LEVEL\_ERROR** Prints perf and error messages only  
**LEVEL\_WARNING** Prints perf, error and warning messages  
**LEVEL\_INFO** Prints perf, errors, warning and information messages

**LEVEL\_DEBUG** Full logging including debug messages

## 5.13 C-V2X

This section contains APIs related to Cellular-V2X operation.

### 5.13.1 Data Structure Documentation

#### 5.13.1.1 class `telux::cv2x::ICv2xConfigListener`

Listeners for `ICv2xConfig` must implement this interface.

##### Public member functions

- virtual void `onConfigChanged` (const `ConfigEventInfo` &info)
- virtual `~ICv2xConfigListener` ()

##### 5.13.1.1.1 Constructors and Destructors

5.13.1.1.1 virtual `telux::cv2x::ICv2xConfigListener::~~ICv2xConfigListener` ( ) [virtual]

Destructor for `ICv2xConfigListener`

##### 5.13.1.1.2 Member Function Documentation

5.13.1.1.2 virtual void `telux::cv2x::ICv2xConfigListener::onConfigChanged` ( const `ConfigEventInfo` &  
*info* ) [virtual]

Called when CV2X configuration has changed in the below scenarios:

1. The specified configuration source has expired.
2. The active configuration source has changed to the specified configuration source type due to the expiration of the configuration source being used.
3. The specified configuration source has been updated.

##### Parameters

<i>in</i>	<i>info</i>	- Information of CV2X configuration event.
-----------	-------------	--

#### 5.13.1.2 class `telux::cv2x::ICv2xConfig`

`Cv2xConfig` provide operations to update or request `cv2x` configuration.

##### Public member functions

- virtual `~ICv2xConfig` ()
- virtual bool `isReady` ()=0
- virtual `std::future< bool >` `onReady` ()=0

- virtual [telux::common::Status updateConfiguration](#) (const std::string &configFilePath, [telux::common::ResponseCallback](#) cb)=0
- virtual [telux::common::Status retrieveConfiguration](#) (const std::string &configFilePath, [telux::common::ResponseCallback](#) cb)=0
- virtual [telux::common::Status registerListener](#) (std::weak\_ptr< [ICv2xConfigListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak\_ptr< [ICv2xConfigListener](#) > listener)=0

### 5.13.1.2.1 Constructors and Destructors

5.13.1.2.1.1 virtual [telux::cv2x::ICv2xConfig::~ICv2xConfig](#) ( ) [[virtual](#)]

### 5.13.1.2.2 Member Function Documentation

5.13.1.2.2.1 virtual bool [telux::cv2x::ICv2xConfig::isReady](#) ( ) [[pure virtual](#)]

Checks if the Cv2x Config Manager is ready.

#### Returns

True if Cv2x Config is ready for service, otherwise returns false.

5.13.1.2.2.2 virtual [std::future<bool>](#) [telux::cv2x::ICv2xConfig::onReady](#) ( ) [[pure virtual](#)]

Wait for Cv2x Config to be ready.

#### Returns

A future that caller can wait on to be notified when Cv2x Radio Manager is ready.

5.13.1.2.2.3 virtual [telux::common::Status telux::cv2x::ICv2xConfig::updateConfiguration](#) ( const [std::string](#) & *configFilePath*, [telux::common::ResponseCallback](#) *cb* ) [[pure virtual](#)]

Updates CV2X configuration. Requires CV2X TX/RX radio status be Inactive. If CV2X radio status is Active or Suspended, call [ICv2xRadioManager::stopCv2x](#) before trying to update configuration. The functionality of V2X configuration expiration is supported by adding an expiration leaf to the V2X configuration file passed in. When the active configuration expires, the system fallbacks to a lower priority V2X configuration [ConfigSourceType](#) if existed. If the V2X status is active, it changes to suspended when the active V2X configuration expires and then changes to active after the system fallbacks to a lower priority V2X configuration or changes to inactive if no V2X configuration is available.

#### Parameters

in	<i>configFilePath</i>	- Path to config file. This is the fully qualified file path including the name of the file.
in	<i>cb</i>	- Callback that is invoked when the send is complete. This may be null.



#### 5.13.1.2.2.4 virtual telux::common::Status telux::cv2x::ICv2xConfig::retrieveConfiguration ( const std::string & *configFilePath*, telux::common::ResponseCallback *cb* ) [pure virtual]

Retrieve active CV2X configuration. The calling application should have write access to the path specified by *configFilePath*. And if the v2x configuration retrieval request succeed, the file specified by *configFilePath* will be created and filled with the configuration contents. Otherwise, no file will be created.

##### Parameters

in	<i>configFilePath</i>	- Path to config file. This is the fully qualified file path including the name of the file.
in	<i>cb</i>	- Callback that is invoked when the configuration retrieval is complete. This may be null.

#### 5.13.1.2.2.5 virtual telux::common::Status telux::cv2x::ICv2xConfig::registerListener ( std::weak\_ptr< ICv2xConfigListener > *listener* ) [pure virtual]

Registers a listener for this [ICv2xConfig](#).

##### Parameters

in	<i>listener</i>	- Listener that implements <a href="#">ICv2xConfigListener</a> interface.
----	-----------------	---

#### 5.13.1.2.2.6 virtual telux::common::Status telux::cv2x::ICv2xConfig::deregisterListener ( std::weak\_ptr< ICv2xConfigListener > *listener* ) [pure virtual]

Deregisters a listener for this [ICv2xConfig](#).

##### Parameters

in	<i>listener</i>	- Previously registered <a href="#">ICv2xConfigListener</a> that is to be deregistered.
----	-----------------	---

### 5.13.1.3 class telux::cv2x::Cv2xFactory

[Cv2xFactory](#) is the factory that creates the Cv2x Radio.

##### Public member functions

- std::shared\_ptr< [ICv2xRadioManager](#) > [getCv2xRadioManager](#) ()
- std::shared\_ptr< [ICv2xConfig](#) > [getCv2xConfig](#) ()
- std::shared\_ptr< [ICv2xThrottleManager](#) > [getCv2xThrottleManager](#) ()

##### Static Public Member Functions

- static [Cv2xFactory](#) & [getInstance](#) ()

### 5.13.1.3.1 Member Function Documentation

#### 5.13.1.3.1.1 `static Cv2xFactory& telux::cv2x::Cv2xFactory::getInstance ( ) [static]`

Get [Cv2xFactory](#) instance

#### Returns

Reference to [Cv2xFactory](#) singleton.

#### 5.13.1.3.1.2 `std::shared_ptr<ICv2xRadioManager> telux::cv2x::Cv2xFactory::getCv2xRadioManager ( )`

Get [Cv2xRadioManager](#) instance.

#### Returns

shared pointer to [Cv2x Radio Manager](#) upon success. nullptr otherwise.

#### 5.13.1.3.1.3 `std::shared_ptr<ICv2xConfig> telux::cv2x::Cv2xFactory::getCv2xConfig ( )`

Get [Cv2xConfig](#) instance.

#### Returns

shared pointer to [Cv2x Config](#) upon success. nullptr otherwise.

#### 5.13.1.3.1.4 `std::shared_ptr<ICv2xThrottleManager> telux::cv2x::Cv2xFactory::getCv2xThrottleManager ( )`

Get [Cv2xThrottleManager](#) instance.

#### Returns

shared pointer to [Cv2x ThrottleManager](#) upon success. nullptr otherwise.

### 5.13.1.4 class `telux::cv2x::ICv2xRadio`

This is class encapsulates a [Cv2xRadio](#) interface.

Returned from `getCv2xRadio` in [Cv2xFactory](#)

**Public member functions**

- virtual [Cv2xRadioCapabilities](#) [getCapabilities](#) () const =0
- virtual bool [isReady](#) () const =0
- virtual bool [isInitialized](#) () const =0
- virtual std::future< [telux::common::Status](#) > [onReady](#) ()=0
- virtual [telux::common::Status](#) [registerListener](#) (std::weak\_ptr< [ICv2xRadioListener](#) > listener)=0
- virtual [telux::common::Status](#) [deregisterListener](#) (std::weak\_ptr< [ICv2xRadioListener](#) > listener)=0
- virtual [telux::common::Status](#) [createRxSubscription](#) ([TrafficIpType](#) ipType, uint16\_t port, [CreateRxSubscriptionCallback](#) cb, std::shared\_ptr< std::vector< uint32\_t >> idList=nullptr)=0
- virtual [telux::common::Status](#) [createTxSpsFlow](#) ([TrafficIpType](#) ipType, uint32\_t serviceId, const [SpsFlowInfo](#) &spsInfo, uint16\_t spsSrcPort, bool eventSrcPortValid, uint16\_t eventSrcPort, [CreateTxSpsFlowCallback](#) cb)=0
- virtual [telux::common::Status](#) [createTxEventFlow](#) ([TrafficIpType](#) ipType, uint32\_t serviceId, uint16\_t eventSrcPort, [CreateTxEventFlowCallback](#) cb)=0
- virtual [telux::common::Status](#) [createTxEventFlow](#) ([TrafficIpType](#) ipType, uint32\_t serviceId, const [EventFlowInfo](#) &flowInfo, uint16\_t eventSrcPort, [CreateTxEventFlowCallback](#) cb)=0
- virtual [telux::common::Status](#) [closeRxSubscription](#) (std::shared\_ptr< [ICv2xRxSubscription](#) > rxSub, [CloseRxSubscriptionCallback](#) cb)=0
- virtual [telux::common::Status](#) [closeTxFlow](#) (std::shared\_ptr< [ICv2xTxFlow](#) > txFlow, [CloseTxFlowCallback](#) cb)=0
- virtual [telux::common::Status](#) [changeSpsFlowInfo](#) (std::shared\_ptr< [ICv2xTxFlow](#) > txFlow, const [SpsFlowInfo](#) &spsInfo, [ChangeSpsFlowInfoCallback](#) cb)=0
- virtual [telux::common::Status](#) [requestSpsFlowInfo](#) (std::shared\_ptr< [ICv2xTxFlow](#) > txFlow, [RequestSpsFlowInfoCallback](#) cb)=0
- virtual [telux::common::Status](#) [changeEventFlowInfo](#) (std::shared\_ptr< [ICv2xTxFlow](#) > txFlow, const [EventFlowInfo](#) &flowInfo, [ChangeEventFlowInfoCallback](#) cb)=0
- virtual [telux::common::Status](#) [requestCapabilities](#) ([RequestCapabilitiesCallback](#) cb)=0
- virtual [telux::common::Status](#) [requestDataSessionSettings](#) ([RequestDataSessionSettingsCallback](#) cb)=0
- virtual [telux::common::Status](#) [updateSrcL2Info](#) ([UpdateSrcL2InfoCallback](#) cb)=0
- virtual [telux::common::Status](#) [updateTrustedUEList](#) (const [TrustedUEInfoList](#) &infoList, [UpdateTrustedUEListCallback](#) cb)=0
- virtual [~ICv2xRadio](#) ()
- virtual std::string [getIfaceNameFromIpType](#) ([TrafficIpType](#) ipType)=0
- virtual [telux::common::Status](#) [createCv2xTcpSocket](#) (const [EventFlowInfo](#) &eventInfo, const [SocketInfo](#) &sockInfo, [CreateTcpSocketCallback](#) cb)=0
- virtual [telux::common::Status](#) [closeCv2xTcpSocket](#) (std::shared\_ptr< [ICv2xTxRxSocket](#) > sock, [CloseTcpSocketCallback](#) cb)=0

- virtual [telux::common::Status registerTxStatusReportListener](#) (uint16\_t port, std::weak\_ptr<[ICv2xTxStatusReportListener](#) > listener, [telux::common::ResponseCallback](#) cb)=0
- virtual [telux::common::Status deregisterTxStatusReportListener](#) (uint16\_t port, [telux::common::ResponseCallback](#) cb)=0

#### 5.13.1.4.1 Constructors and Destructors

##### 5.13.1.4.1.1 virtual [telux::cv2x::ICv2xRadio::~~ICv2xRadio](#) ( ) [[virtual](#)]

Destructor for [ICv2xRadio](#)

#### 5.13.1.4.2 Member Function Documentation

##### 5.13.1.4.2.1 virtual [Cv2xRadioCapabilities](#) [telux::cv2x::ICv2xRadio::getCapabilities](#) ( ) const [[pure virtual](#)]

Get the capabilities of this [Cv2xRadio](#).

#### Returns

[Cv2xRadioCapabilities](#) - Contains capabilities of this [Cv2xRadio](#).

**Deprecated** Use [requestCapabilities\(\)](#) API

##### 5.13.1.4.2.2 virtual bool [telux::cv2x::ICv2xRadio::isReady](#) ( ) const [[pure virtual](#)]

Returns true if the radio interface was successfully initialized.

#### Returns

True if ready. False otherwise.

##### 5.13.1.4.2.3 virtual bool [telux::cv2x::ICv2xRadio::isInitialized](#) ( ) const [[pure virtual](#)]

Returns true if the radio interface has completed initialization.

#### Returns

True if initialized. False otherwise.

**5.13.1.4.2.4** `virtual std::future<telux::common::Status> telux::cv2x::ICv2xRadio::onReady ( ) [pure virtual]`

Returns a future that indicated if the radio interface is ready or if radio failed to initialize.

#### Returns

SUCCESS if Cv2xRadio initialization was successful. Otherwise it returns an Error Code.

**5.13.1.4.2.5** `virtual telux::common::Status telux::cv2x::ICv2xRadio::registerListener ( std::weak_ptr<ICv2xRadioListener > listener ) [pure virtual]`

Registers a listener for this Cv2xRadio.

#### Parameters

in	<i>listener</i>	- Listener that implements Cv2xRadioListener interface.
----	-----------------	---

**5.13.1.4.2.6** `virtual telux::common::Status telux::cv2x::ICv2xRadio::deregisterListener ( std::weak_ptr<ICv2xRadioListener > listener ) [pure virtual]`

Deregisters a listener from this Cv2xRadio.

#### Parameters

in	<i>listener</i>	- Previously registered Cv2xRadioListener that is to be deregistered.
----	-----------------	---

**5.13.1.4.2.7** `virtual telux::common::Status telux::cv2x::ICv2xRadio::createRxSubscription ( TrafficType ipType, uint16_t port, CreateRxSubscriptionCallback cb, std::shared_ptr<std::vector< uint32_t >> idList = nullptr ) [pure virtual]`

Creates and initializes a new Rx subscription which will be returned in the user-supplied callback.

#### Parameters

in	<i>ipType</i>	- IP traffic type (IP or NON-IP)
in	<i>port</i>	- Rx port number
in	<i>cb</i>	- Callback function that is invoked when socket creation is complete.
in	<i>idList</i>	- Service ID list to subscribe, optional parameter using nullptr by default. Subscribe wildcard if this parameter is set to nullptr.

#### Returns

SUCCESS on success. Error status otherwise.

**Dependencies** The interface must be pre-initialized with `init()`.

**5.13.1.4.2.8** `virtual telux::common::Status telux::cv2x::ICv2xRadio::createTxSpsFlow ( TrafficIpType ipType, uint32_t serviceId, const SpsFlowInfo & spsInfo, uint16_t spsSrcPort, bool eventSrcPortValid, uint16_t eventSrcPort, CreateTxSpsFlowCallback cb ) [pure virtual]`

Creates a Tx SPS flow with the specified IP type, serviceId, and other parameters specified in reservation. Additionally, an option event flow will be created with the same IP type and serviceId. A Tx socket will be created and initialized for the SPS flow. A Tx socket will be created and initialized for the event flow if the optional event flow is specified.

#### Parameters

in	<i>ipType</i>	- IP traffic type (IP or NON-IP)
in	<i>serviceId</i>	- ID used for transmissions that will be mapped to an L2 destination address. Variable length 4-byte PSID or ITS_AID, or another service ID.
in	<i>spsInfo</i>	- SPS reservation parameters.
in	<i>spsPort</i>	- Requested source port number for the bandwidth reserved SPS transmissions.
in	<i>eventSrcPortValid</i>	- True if an optional event flow is desired. If this field is left false, the event flow will not be created.
in	<i>eventSrcPort</i>	- Requested source port number for the optional event flow.
in	<i>cb</i>	- Callback function that is invoked when socket creation is complete. This must not be null.

**This caller is expected to identify two unused local port numbers**

to use for binding: one for the event-driven flow and one for the SPS flow.

#### Returns

SUCCESS upon success. Error status otherwise.

**5.13.1.4.2.9** `virtual telux::common::Status telux::cv2x::ICv2xRadio::createTxEventFlow ( TrafficIpType ipType, uint32_t serviceId, uint16_t eventSrcPort, CreateTxEventFlowCallback cb ) [pure virtual]`

Creates an event flow. An associated Tx socket will be created and initialized.

#### Parameters

in	<i>ipType</i>	- IP traffic type (IP or NON-IP)
in	<i>serviceId</i>	- ID used for transmissions that will be mapped to an L2 destination address. Variable length 4-byte PSID or ITS_AID, or another service ID.

in	<i>eventSrcPort</i>	- Local port number to which the socket is bound. Used for transmissions of this ID.
in	<i>cb</i>	- Callback function that is invoked when socket creation is complete. This must not be null.

**Detailed description** This function is used only for TX when no periodicity is

available for the application type. If your transmit data periodicity is known, use [createTxSpsFlow\(\)](#) instead.

**These even-driven sockets pay attention to the QoS parameters in**

the IP socket.

**Returns**

SUCCESS upon success. Error status otherwise.

**5.13.1.4.2.10** `virtual telux::common::Status telux::cv2x::ICv2xRadio::createTxEventFlow ( TrafficIpType ipType, uint32_t serviceId, const EventFlowInfo & flowInfo, uint16_t eventSrcPort, CreateTxEventFlowCallback cb ) [pure virtual]`

Creates an event flow. An associated Tx socket will be created and initialized.

**Parameters**

in	<i>ipType</i>	- IP traffic type (IP or NON-IP)
in	<i>serviceId</i>	- ID used for transmissions that will be mapped to an L2 destination address. Variable length 4-byte PSID or ITS_AID, or another service ID.
in	<i>flowInfo</i>	- Flow configuration parameters
in	<i>eventSrcPort</i>	- Local port number to which the socket is bound. Used for transmissions of this ID.
in	<i>cb</i>	- Callback function that is invoked when socket creation is complete. This must not be null.

**Detailed description** This function is used only for TX when no periodicity is

available for the application type. If your transmit data periodicity is known, use [createTxSpsFlow\(\)](#) instead.

**These even-driven sockets pay attention to the QoS parameters in**

the IP socket.

**Returns**

SUCCESS upon success. Error status otherwise.

**5.13.1.4.2.11** `virtual telux::common::Status telux::cv2x::ICv2xRadio::closeRxSubscription ( std::shared_ptr< ICv2xRxSubscription > rxSub, CloseRxSubscriptionCallback cb ) [pure virtual]`

Closes the RxSubscription and frees resources (such as the Rx socket) associated with it.

#### Parameters

in	<i>rxSub</i>	- RxSubscription to close
in	<i>cb</i>	- Callback that is invoked when socket close is complete. This may be null.

#### Returns

SUCCESS if no error occurred.

**5.13.1.4.2.12** `virtual telux::common::Status telux::cv2x::ICv2xRadio::closeTxFlow ( std::shared_ptr< ICv2xTxFlow > txFlow, CloseTxFlowCallback cb ) [pure virtual]`

Closes the TxFlow and frees resources associated with it (such as reserved SPS bandwidth contracts and sockets). This function works on both SPS and event flows.

#### Parameters

in	<i>txFlow</i>	- Tx (SPS or event) flow to close.
in	<i>cb</i>	- Callback that is invoked when Tx flow close is complete. This may be null.

#### Returns

SUCCESS if no error occurred.

**5.13.1.4.2.13** `virtual telux::common::Status telux::cv2x::ICv2xRadio::changeSpsFlowInfo ( std::shared_ptr< ICv2xTxFlow > txFlow, const SpsFlowInfo & spsInfo, ChangeSpsFlowInfoCallback cb ) [pure virtual]`

Request to change TX SPS Flow reservation parameters.

#### Parameters

in	<i>txFlow</i>	- Tx SPS flow
in	<i>spsInfo</i>	- Desired SPS reservation parameters
in	<i>cb</i>	- Callback that is invoked upon reservation change. This may be null.

#### Detailed description

This function does not update reservation priority



**Returns**

SUCCESS if no error occurred.

**5.13.1.4.2.14** `virtual telux::common::Status telux::cv2x::ICv2xRadio::requestSpsFlowInfo ( std::shared_ptr< ICv2xTxFlow > txFlow, RequestSpsFlowInfoCallback cb ) [pure virtual]`

Request SPS flow info.

**Parameters**

in	<i>sock</i>	- Tx SPS flow
in	<i>cb</i>	- Callback that will be invoked and returns the SPS info. Must not be null.

**Returns**

SUCCESS if no error occurred.

**5.13.1.4.2.15** `virtual telux::common::Status telux::cv2x::ICv2xRadio::changeEventFlowInfo ( std::shared_ptr< ICv2xTxFlow > txFlow, const EventFlowInfo & flowInfo, ChangeEventFlowInfoCallback cb ) [pure virtual]`

Request to change TX Event Flow reservation parameters.

**Parameters**

in	<i>txFlow</i>	- Tx Event flow
in	<i>flowInfo</i>	- Desired Event flow parameters
in	<i>cb</i>	- Callback that is invoked upon parameter change. This may be null.

**Returns**

SUCCESS if no error occurred.

**5.13.1.4.2.16** `virtual telux::common::Status telux::cv2x::ICv2xRadio::requestCapabilities ( RequestCapabilitiesCallback cb ) [pure virtual]`

Request modem Cv2x capability information.

**Parameters**

in	<i>cb</i>	- Callback that will be invoked and returns the capability info. Must not be null.
----	-----------	--

**Returns**

SUCCESS if no error occurred.

#### 5.13.1.4.2.17 **virtual telux::common::Status telux::cv2x::ICv2xRadio::requestDataSessionSettings ( RequestDataSessionSettingsCallback *cb* ) [pure virtual]**

Request data session settings currently in use.

##### Parameters

in	<i>cb</i>	- Callback that will be invoked and returns the data session settings. Must not be null.
----	-----------	--

##### Returns

SUCCESS if no error occurred.

#### 5.13.1.4.2.18 **virtual telux::common::Status telux::cv2x::ICv2xRadio::updateSrcL2Info ( UpdateSrcL2InfoCallback *cb* ) [pure virtual]**

Requests modem to change L2 info.

##### Parameters

in	<i>cb</i>	- Callback that will be invoked and returns status. Must not be null.
----	-----------	---

##### Returns

SUCCESS if no error occurred.

#### 5.13.1.4.2.19 **virtual telux::common::Status telux::cv2x::ICv2xRadio::updateTrustedUEList ( const TrustedUEInfoList & *infoList*, UpdateTrustedUEListCallback *cb* ) [pure virtual]**

Send request to modem to update the list of malicious UE source IDs and trusted UE source IDs with corresponding confidence information.

##### Parameters

in	<i>infoList</i>	- Trusted and malicious UE information list
in	<i>cb</i>	- Callback that will be invoked and returns status. Must not be null.

##### Returns

SUCCESS if no error occurred.

#### 5.13.1.4.2.20 **virtual std::string telux::cv2x::ICv2xRadio::getInterfaceNameFromIpType ( TrafficIpType *ipType* ) [pure virtual]**

Get interface name based on ipType.

<i>ipType</i>	- IP traffic type (IP or NON-IP)
---------------	----------------------------------

## Returns

Interface name as a string

## Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.13.1.4.2.21 **virtual telux::common::Status telux::cv2x::ICv2xRadio::createCv2xTcpSocket ( const EventFlowInfo & *eventInfo*, const SocketInfo & *sockInfo*, CreateTcpSocketCallback *cb* ) [pure virtual]**

Creates a CV2X TCP socket with specified event flow information and TCP socket information. The TCP socket will be created and bound to the IPv6 address of local IP interface with specified source port. Additionally, this API also registers a Tx event flow and subscribes Rx with specified service ID. If the created socket is expected to work as TCP client mode, the caller must connect the created socket to a destination using `connect()` and then use the socket for `send()` and `recv()` on successful connection. If the created socket is expected to work as TCP server mode, the caller must mark this socket as a listening socket using `listen()` and accept connections received from this listening socket using `accept()`, and then use the accepted sockets returned from `accept()` for `send()` or `recv()`.

## Parameters

in	<i>eventInfo</i>	- Information for the Event flow.
in	<i>sockInfo</i>	- Information for the TCP socket.
in	<i>cb</i>	- Callback function that is invoked when socket creation is complete. This must not be null.

## The caller is expected to identify an unused local port number as the source

port number in structure [SocketInfo](#) to use for binding.

## The caller must release the created socket and associated resources with

[closeCv2xTcpSocket](#). Additionally, if the created socket is marked as a listening socket, the caller must close all the accepted sockets returned by `accept()` using `close()` first, and then release the listening socket and associated resources by calling [closeCv2xTcpSocket](#).

## Returns

SUCCESS upon success. Error status otherwise.

**5.13.1.4.2.22** `virtual telux::common::Status telux::cv2x::ICv2xRadio::closeCv2xTcpSocket ( std::shared_ptr< ICv2xTxRxSocket > sock, CloseTcpSocketCallback cb ) [pure virtual]`

Closes the CV2X TCP socket and frees resources associated with it (such as registered event Tx flow and subscribed Rx service ID and created TCP socket).

#### Parameters

in	<i>sock</i>	- CV2X TCP socket to close.
in	<i>cb</i>	- Callback that is invoked when CV2X TCP socket close is complete. This may be null.

#### Returns

SUCCESS if no error occurred.

**5.13.1.4.2.23** `virtual telux::common::Status telux::cv2x::ICv2xRadio::registerTxStatusReportListener ( uint16_t port, std::weak_ptr< ICv2xTxStatusReportListener > listener, telux::common::ResponseCallback cb ) [pure virtual]`

Registers a listener for Tx status report.

#### Parameters

in	<i>port</i>	- Set this value to the port number of registered Tx Flow if user wants to receive Tx status report associated with its own Tx flow. If user wants to receive Tx status report associated with all Tx flows in system, set this value to 0.
in	<i>listener</i>	- Listener that implements <a href="#">ICv2xTxStatusReportListener</a> interface.
in	<i>cb</i>	- Callback that is invoked when the registration of CV2X Tx status report is complete.

**5.13.1.4.2.24** `virtual telux::common::Status telux::cv2x::ICv2xRadio::deregisterTxStatusReportListener ( uint16_t port, telux::common::ResponseCallback cb ) [pure virtual]`

Deregisters a listener for Tx status report.

#### Parameters

in	<i>port</i>	- Port number of previously registered <a href="#">ICv2xTxStatusReportListener</a> that is to be deregistered. If the listener is registered with port number 0, set this value to 0 to deregister the listener.
in	<i>cb</i>	- Callback that is invoked when the deregistration of CV2X Tx status report is complete.

### 5.13.1.5 class telux::cv2x::ICv2xRadioListener

Listeners for Cv2xRadio must implement this interface.

#### Public member functions

- virtual void [onStatusChanged](#) (Cv2xStatus status)
- virtual void [onStatusChanged](#) (Cv2xStatusEx status)
- virtual void [onL2AddrChanged](#) (uint32\_t newL2Address)
- virtual void [onSpsOffsetChanged](#) (int spsId, [MacDetails](#) details)
- virtual void [onSpsSchedulingChanged](#) (const [SpsSchedulingInfo](#) &schedulingInfo)
- virtual void [onCapabilitiesChanged](#) (const [Cv2xRadioCapabilities](#) &capabilities)
- virtual [~ICv2xRadioListener](#) ()

#### 5.13.1.5.1 Constructors and Destructors

5.13.1.5.1.1 virtual telux::cv2x::ICv2xRadioListener::~~ICv2xRadioListener ( ) [virtual]

Destructor for [ICv2xRadioListener](#)

#### 5.13.1.5.2 Member Function Documentation

5.13.1.5.2.1 virtual void telux::cv2x::ICv2xRadioListener::onStatusChanged ( Cv2xStatus *status* ) [virtual]

Called when the status of the CV2X radio has changed.

#### Parameters

in	<i>status</i>	- CV2X radio status.
----	---------------	----------------------

**Deprecated** use [onStatusChanged](#) in [Cv2xListener](#)

5.13.1.5.2.2 virtual void telux::cv2x::ICv2xRadioListener::onStatusChanged ( Cv2xStatusEx *status* ) [virtual]

Called when the status of the CV2X radio has changed.

#### Parameters

in	<i>status</i>	- CV2X radio status.
----	---------------	----------------------

**Deprecated** use [onStatusChanged](#) in [Cv2xListener](#)

**5.13.1.5.2.3** virtual void telux::cv2x::ICv2xRadioListener::onL2AddrChanged ( uint32\_t *newL2Address* ) [virtual]

Called when the L2 Address has changed.

#### Parameters

in	<i>newL2Address</i>	- The new L2 address.
----	---------------------	-----------------------

**5.13.1.5.2.4** virtual void telux::cv2x::ICv2xRadioListener::onSpsOffsetChanged ( int *spsId*, MacDetails *details* ) [virtual]

Called when SPS offset has changed.

#### Parameters

in	<i>spsId</i>	- SPS Id of the SPS flow
in	<i>details</i>	- new SPS MAC PHY details.

**Deprecated** use onSpsSchedulingChanged

**5.13.1.5.2.5** virtual void telux::cv2x::ICv2xRadioListener::onSpsSchedulingChanged ( const SpsSchedulingInfo & *schedulingInfo* ) [virtual]

Called when SPS scheduling has changed.

#### Parameters

in	<i>schedulingInfo</i>	- SPS scheduling information .
----	-----------------------	--------------------------------

**5.13.1.5.2.6** virtual void telux::cv2x::ICv2xRadioListener::onCapabilitiesChanged ( const Cv2xRadioCapabilities & *capabilities* ) [virtual]

Called when Cv2x radio capabilities have changed.

#### Parameters

in	<i>capabilities</i>	- Capabilities of the CV2X radio .
----	---------------------	------------------------------------

### 5.13.1.6 class telux::cv2x::ICv2xRadioManager

Cv2xRadioManager manages instances of Cv2xRadio.

#### Public member functions

- virtual bool [isReady](#) ()=0
- virtual std::future< bool > [onReady](#) ()=0

- virtual `std::shared_ptr< ICv2xRadio > getCv2xRadio (TrafficCategory category)=0`
- virtual `telux::common::Status startCv2x (StartCv2xCallback cb)=0`
- virtual `telux::common::Status stopCv2x (StopCv2xCallback cb)=0`
- virtual `telux::common::Status requestCv2xStatus (RequestCv2xStatusCallback cb)=0`
- virtual `telux::common::Status requestCv2xStatus (RequestCv2xStatusCallbackEx cb)=0`
- virtual `telux::common::Status registerListener (std::weak_ptr< ICv2xListener > listener)=0`
- virtual `telux::common::Status deregisterListener (std::weak_ptr< ICv2xListener > listener)=0`
- virtual `telux::common::Status updateConfiguration (const std::string &configFilePath, UpdateConfigurationCallback cb)=0`
- virtual `~ICv2xRadioManager ()`

### 5.13.1.6.1 Constructors and Destructors

5.13.1.6.1.1 virtual `telux::cv2x::ICv2xRadioManager::~~ICv2xRadioManager ( ) [virtual]`

### 5.13.1.6.2 Member Function Documentation

5.13.1.6.2.1 virtual `bool telux::cv2x::ICv2xRadioManager::isReady ( ) [pure virtual]`

Checks if the Cv2x Radio Manager is ready.

#### Returns

True if Cv2x Radio Manager is ready for service, otherwise returns false.

5.13.1.6.2.2 virtual `std::future<bool> telux::cv2x::ICv2xRadioManager::onReady ( ) [pure virtual]`

Wait for Cv2x Radio Manager to be ready.

#### Returns

A future that caller can wait on to be notified when Cv2x Radio Manager is ready.

5.13.1.6.2.3 virtual `std::shared_ptr<ICv2xRadio> telux::cv2x::ICv2xRadioManager::getCv2xRadio ( TrafficCategory category ) [pure virtual]`

Get Cv2xRadio instance

#### Parameters

in	<i>category</i>	- Specifies the category of the client application. This field is currently unused.
----	-----------------	---

**Returns**

Reference to Cv2xRadio interface that corresponds to the Cv2x Traffic Category specified.

**5.13.1.6.2.4 virtual telux::common::Status telux::cv2x::ICv2xRadioManager::startCv2x ( StartCv2x↔ Callback *cb* ) [pure virtual]**

Put modem into CV2X mode.

**Parameters**

<i>in</i>	<i>cb</i>	- Callback that is invoked when Cv2x mode is started
-----------	-----------	--

**Returns**

SUCCESS on success. Error status otherwise.

**5.13.1.6.2.5 virtual telux::common::Status telux::cv2x::ICv2xRadioManager::stopCv2x ( StopCv2x↔ Callback *cb* ) [pure virtual]**

Take modem out of CV2X mode

**Parameters**

<i>in</i>	<i>cb</i>	- Callback that is invoked when Cv2x mode is stopped
-----------	-----------	--

**Returns**

SUCCESS on success. Error status otherwise.

**5.13.1.6.2.6 virtual telux::common::Status telux::cv2x::ICv2xRadioManager::requestCv2xStatus ( RequestCv2xStatusCallback *cb* ) [pure virtual]**

request CV2X status from modem

**Parameters**

<i>in</i>	<i>cb</i>	- Callback that is invoked when Cv2x status is retrieved
-----------	-----------	--

**Returns**

SUCCESS on success. Error status otherwise.

**Deprecated** use requestCv2xStatus(RequestCv2xCalbackEx)

**5.13.1.6.2.7 virtual telux::common::Status telux::cv2x::ICv2xRadioManager::requestCv2xStatus ( RequestCv2xStatusCallbackEx *cb* ) [pure virtual]**

request CV2X status from modem



in	<i>cb</i>	- Callback that is invoked when Cv2x status is retrieved
----	-----------	--

**Returns**

SUCCESS on success. Error status otherwise.

**5.13.1.6.2.8 virtual telux::common::Status telux::cv2x::ICv2xRadioManager::registerListener ( std::weak\_ptr< ICv2xListener > *listener* ) [pure virtual]**

Registers a listener for this manager.

**Parameters**

in	<i>listener</i>	- Listener that implements Cv2xListener interface.
----	-----------------	--

**5.13.1.6.2.9 virtual telux::common::Status telux::cv2x::ICv2xRadioManager::deregisterListener ( std::weak\_ptr< ICv2xListener > *listener* ) [pure virtual]**

Deregisters a Cv2xListener for this manager.

**Parameters**

in	<i>listener</i>	- Previously registered CvListener that is to be deregistered.
----	-----------------	--

**5.13.1.6.2.10 virtual telux::common::Status telux::cv2x::ICv2xRadioManager::updateConfiguration ( const std::string & *configFilePath*, UpdateConfigurationCallback *cb* ) [pure virtual]**

Updates CV2X configuration. Requires CV2X TX/RX radio status be Inactive. If CV2X radio status is Active or Suspended, call [stopCv2x](#) before updateConfiguration.

**Parameters**

in	<i>configFilePath</i>	- Path to config file.
in	<i>cb</i>	- Callback that is invoked when the send is complete. This may be null.

**Deprecated** Use [ICv2xConfig](#) instead

**5.13.1.7 struct telux::cv2x::SocketInfo**

Encapsulates parameters of a CV2X socket.

Used in createCv2xTcpSocket.

**Data fields**

Type	Field	Description
uint32_t	serviceId	V2X service ID bound to the socket.
uint16_t	localPort	Local port number of the socket used for binding.

### 5.13.1.8 struct telux::cv2x::Cv2xStatus

Encapsulates status of CV2X radio.

Used in Cv2xRadioManager:requestV2xStatus and Cv2xRadioListener.

#### Data fields

Type	Field	Description
<a href="#">Cv2xStatus</a> ↔ Type	rxStatus	RX status
<a href="#">Cv2xStatus</a> ↔ Type	txStatus	TX status
<a href="#">Cv2xCause</a> ↔ Type	rxCause	RX cause of failure
<a href="#">Cv2xCause</a> ↔ Type	txCause	TX cause of failure
uint8_t	cbrValue	Channel Busy Ratio
bool	cbrValueValid	CBR value is valid

### 5.13.1.9 struct telux::cv2x::Cv2xPoolStatus

Encapsulates status for single pool.

Used in [Cv2xStatusEx](#).

#### Data fields

Type	Field	Description
uint8_t	poolId	pool ID
<a href="#">Cv2xStatus</a>	status	status

### 5.13.1.10 struct telux::cv2x::Cv2xStatusEx

Encapsulates status of CV2X radio and per pool status.

Used in Cv2xRadioManager:requestV2xStatus and Cv2xRadioListener.

#### Data fields

Type	Field	Description
<a href="#">Cv2xStatus</a>	status	Overall Cv2x status
vector< <a href="#">Cv2xPool</a> ↔ <a href="#">Status</a> >	poolStatus	Multi pool status vector
bool	time↔ Uncertainty↔ Valid	Time uncertainty value is valid
float	time↔ Uncertainty	Time uncertainty value in milliseconds

### 5.13.1.11 struct telux::cv2x::TxPoolIdInfo

Contains minimum and maximum frequency for a given TX pool ID. Multiple TX Pools allow the same radio and overall frequency range to be shared for multiple types of traffic like V2V and V2X. Each pool ID and frequency range corresponds to a certain type of traffic.

Used in [Cv2xRadioCapabilities](#)

#### Data fields

Type	Field	Description
uint8_t	poolId	TX pool ID.
uint16_t	minFreq	Minimum frequency in MHz.
uint16_t	maxFreq	Maximum frequency in MHz.

### 5.13.1.12 struct telux::cv2x::EventFlowInfo

Contains event flow configuration parameters.

Used in [createTxEventFlow](#)

#### Data fields

Type	Field	Description
bool	autoRetrans↔ EnabledValid	Set to true if autoRetransEnabled field is specified. If false, the system will use the default setting.
bool	autoRetrans↔ Enabled	Used to enable automatic-retransmissions.
bool	peakTxPower↔ Valid	Set to true if peakTxPower is used. If false, the system will use the default setting.
int32_t	peakTxPower	Max Tx power setting in dBm.
bool	mcsIndexValid	Set to true if mcsIndex is used. If false, the system will use its default setting.
uint8_t	mcsIndex	Modulation and Coding Scheme Index to use.
bool	txPoolIdValid	Set to true if txPoolId is used. If false, the system will use its default setting.
uint8_t	txPoolId	Transmission Pool ID.

### 5.13.1.13 struct telux::cv2x::SpsFlowInfo

Used to request the QoS bandwidth contract, implemented in PC5 3GPP V2X radio as a *Semi Persistent Flow* (SPS).

The underlying radio providing the interface might support periodicities of various granularity in 100ms integer multiples (e.g. 200ms, 300ms).

Used in [txSpsCreateAndBindSock](#) and [changeSpsFlowInfo](#)

**Data fields**

Type	Field	Description
<a href="#">Priority</a>	priority	Specifies one of the 3GPP levels of Priority for the traffic that is pre-reserved on the SPS flow. Default is PRIORITY_2. Use getCapabilities() to discover the supported priority levels. : periodicity, Use new periodicityMs instead
<a href="#">Periodicity</a>	periodicity	
uint64_t	periodicityMs	This is the new interface to specify periodicity in milliseconds for <a href="#">SpsFlowInfo</a> . Enum Periodicity is deprecated and will be removed in future release. Bandwidth-reserved periodicity interval in interval in milliseconds. There are limits on which intervals the underlying radio supports. Use getCapabilities() to discover minPeriodicityMultiplierMs and maximumPeriodicityMs.
uint32_t	nbytesReserved	Number of bytes of TX bandwidth that are sent every periodicity interval.
bool	autoRetrans↔ EnabledValid	Set to true if autoRetransEnabled field is specified. If false, the system will use the default setting.
bool	autoRetrans↔ Enabled	Used to enable automatic-retransmissions.
bool	peakTxPower↔ Valid	Set to true if peakTxPower is used. If false, the system will use the default setting.
int32_t	peakTxPower	Max Tx power setting in dBm.
bool	mcsIndexValid	Set to true if mcsIndex is used. If false, the system will use its default setting.
uint8_t	mcsIndex	Modulation and Coding Scheme Index to use.
bool	txPoolIdValid	Set to true if txPoolId is used. If false, the system will use its default setting.
uint8_t	txPoolId	Transmission Pool ID.

**5.13.1.14 struct telux::cv2x::Cv2xRadioCapabilities**

Contains capabilities of the Cv2xRadio.

Used in requestCapabilities and onCapabilitiesChanged

**Data fields**

Type	Field	Description
uint32_t	linkIpMtuBytes	Maximum data payload length (in bytes) of a packet supported by the IP Radio interface.
uint32_t	linkNonIp↔ MtuBytes	Maximum data payload length (in bytes) of a packet supported by the non-IP Radio interface.
<a href="#">Radio↔ Concurrency↔ Mode</a>	max↔ Supported↔ Concurrency	Indicates whether this interface supports concurrent WWAN with V2X (PC5).
uint16_t	nonIpTx↔ Payload↔ OffsetBytes	Byte offset in a non-IP Tx packet before the actual payload begins.

Type	Field	Description
uint16_t	nonIpRx↔ Payload↔ OffsetBytes	Byte offset in a non-IP Rx packet before the actual payload begins. : periodicitiesSupported, Use new periodicities instead
bitset< 8 >	periodicities↔ Supported	
vector< uint64_t >	periodicities	Specifies the periodicities supported
uint8_t	maxNum↔ Auto↔ Retransmissions	Least frequent bandwidth periodicity that is supported. Above this value, use event-driven periodic messages of a period larger than this value.
uint8_t	layer2Mac↔ AddressSize	Size of the L2 MAC address. Different Radio Access Technologies have different-sized L2 MAC addresses: 802.11 has 6 bytes, whereas 3GPP PC5 has only 3 bytes. Because a randomized MAC address comes from an HSM with good pseudo random entropy, higher layers must know how many bytes of the MAC address to generate.
bitset< 8 >	priorities↔ Supported	Bit set of different priority levels supported by this Cv2xRadio. Refer to Priority
uint16_t	maxNumSps↔ Flows	Maximum number of supported SPS reservations.
uint16_t	maxNumNon↔ SpsFlows	Maximum number of supported event flows (non-SPS ports).
int32_t	maxTxPower	Maximum supported transmission power.
int32_t	minTxPower	Minimum supported transmission power.
vector< Tx↔ PoolIdInfo >	txPoolIds↔ Supported	Vector of supported transmission pool IDs.

### 5.13.1.15 struct telux::cv2x::MacDetails

Contains MAC information that is reported from the actual MAC SPS in the radio. The offsets can periodically change on any given transmission report.

#### Data fields

Type	Field	Description
uint32_t	periodicityIn↔ UseNs	Actual transmission interval period (in nanoseconds) scheduled relative to 1PP 0:00.00 time
uint16_t	currently↔ Reserved↔ PeriodicBytes	Actual number of bytes currently reserved at the MAC layer. This number can be slightly larger than original request.
uint32_t	txReservation↔ OffsetNs	Actual offset, from a 1PPS pulse and TX flow periodicity, that the MAC selected and is using for the transmit reservation. If the data goes to the radio with enough time, it can be transmitted on the medium in the next immediately scheduled slot.

### 5.13.1.16 struct telux::cv2x::SpsSchedulingInfo

Contains SPS packet scheduling information that is reported from the radio.

Used in onSpsSchedulingChanged

#### Data fields

Type	Field	Description
uint8_t	spsId	SPS ID
uint64_t	utcTime	Absolute UTC start time of next selected grant in nanoseconds.
uint32_t	periodicity	Periodicity of the grant in milliseconds.

### 5.13.1.17 struct telux::cv2x::TrustedUEInfo

Contains time confidence, position confidence, and propagation delay for a trusted UE.

Used in [TrustedUEInfo](#)

#### Data fields

Type	Field	Description
uint32_t	sourceL2Id	Trusted Source L2 ID
float	time↔ Uncertainty	Time uncertainty value in milliseconds.
uint16_t	time↔ Confidence↔ Level	<b>Deprecated</b> Use timeUncertainty Time confidence level. Range from 0 to 127 with 0 being invalid/unavailable and 127 being the most confident.
uint16_t	position↔ Confidence↔ Level	Position confidence level. Range from 0 to 127 with 0 being invalid/unavailable and 127 being the most confident.
uint32_t	propagation↔ Delay	Propagation delay in microseconds.

### 5.13.1.18 struct telux::cv2x::TrustedUEInfoList

Contains list of malicious UE source L2 IDs. Contains list of trusted UE source L2 IDs and associated confidence values.

Used in updateTrustedUEList

#### Data fields

Type	Field	Description
bool	maliciousIds↔ Valid	Malicious remote UE sources are valid.
vector< uint32_t >	maliciousIds	Malicious remote UE source L2 IDs.

Type	Field	Description
bool	trustedUEs↔ Valid	Trusted remote UE sources are valid.
vector< <a href="#">TrustedUE↔</a> <a href="#">Info</a> >	trustedUEs	Trusted remote UE sources.

### 5.13.1.19 struct telux::cv2x::IPv6Address

Contains IPv6 address.

Used in [DataSessionSettings](#)

#### Data fields

Type	Field	Description
uint8_t	addr[16]	

### 5.13.1.20 struct telux::cv2x::DataSessionSettings

Contains packet data session settings.

Used in [requestDataSessionSettings](#)

#### Data fields

Type	Field	Description
bool	mtuValid	Set to true if mtu is valid.
uint32_t	mtu	MTU size.
bool	ipv6AddrValid	Set to true if ipv6 address is valid.
<a href="#">IPv6Address</a>	ipv6Addr	IPv6 address.

### 5.13.1.21 struct telux::cv2x::ConfigEventInfo

Information about any update to a CV2X config file.

Used in [onConfigFileChanged](#)

#### Data fields

Type	Field	Description
<a href="#">ConfigSource↔</a> Type	source	The type of the V2X config file.
<a href="#">ConfigEvent</a>	event	V2X config event.

### 5.13.1.22 struct telux::cv2x::RFTxInfo

Tx status per Tx chain and Tx power per Tx antenna for a specific transport block.

Used in [TxStatusReport](#)

#### Data fields

Type	Field	Description
<a href="#">RFTxStatus</a>	status	Fault detection status for a specific Tx chain.
int32_t	power	The target Tx power after MPR/AMPR reduction for a specific Tx antenna in dBm*10 format. Invalid value is -700, it means the corresponding antenna is not being used for the transmission of this transport block.

### 5.13.1.23 struct telux::cv2x::TxStatusReport

Information on Tx status of a V2X transport block that is reported from low layer.

1. A V2X packet might trigger multiple reports because of the segmentaion and re-Tx in low layer.
2. If a transport block is dropped in low layer, no report will be triggered for that transport block.
3. The power in the array of rfInfo is the target Tx power value in dBm\*10 after MPR/AMPR reduction for a specific Tx antenna. The status in the array of rfInfo is the fault detection status for a specific Tx chain.
  - In CDD mode, two antennas are being used for a specific transport block, both rfInfo[0].power and rfInfo[1].power are valid (not -700), rfInfo[i].status is reflecting the status of Tx chain/Tx antenna i.
  - In TXD mode, data transmission swtiches between two antennas/chains and only one antenna/chain is being used for a specific transport block, the Tx antenna being used has valid power (not -700) in the array of rfInfo, rfInfo[i].status is reflecting the status of Tx chain i or the status of the Tx antenna i whose power is valid (not -700) in the array of rfInfo. Used in onTxStatusReport

#### Data fields

Type	Field	Description
<a href="#">RFTxInfo</a>	rfInfo[ <a href="#">MAX_↔ ANTENNAS_↔ _SUPPORT_↔ ED</a> ]	Tx status per Tx chain and Tx power per Tx antenna.
uint8_t	numRb	Number of resource blocks used for the transport block.
uint8_t	startRb	Start resource block index used for the transport block.
uint8_t	mcs	Modulation and coding scheme used for the transport block that is defined in 3GPP TS 36.213.
uint8_t	segNum	Total number of segments of a V2X packet.
<a href="#">SegmentType</a>	segType	Segment type of the transport block.
<a href="#">TxType</a>	txType	Indication of new Tx or re-Tx of the transport block.
uint16_t	otaTiming	OTA timing in format of system frame number*10 + subframe number.



Type	Field	Description
uint16_t	port	Port number that can be used to link the report to a specific Tx flow which has the same source port number.

### 5.13.1.24 class telux::cv2x::ICv2xRxSubscription

This class encapsulates a Cv2xRadio Rx Subscription. It contains the Rx socket associated with the subscription from which client applications can read data. This class is referenced in Cv2xRadio::createRxSubscription and Cv2xRadio::closeRxSubscription.

#### Public member functions

- virtual uint32\_t [getSubscriptionId](#) () const =0
- virtual [TrafficIpType](#) [getIpType](#) () const =0
- virtual int [getSock](#) () const =0
- virtual struct sockaddr\_in6 [getSockAddr](#) () const =0
- virtual uint16\_t [getPortNum](#) () const =0
- virtual std::shared\_ptr< std::vector< uint32\_t > > [getServiceIDList](#) () const =0
- virtual void [setServiceIDList](#) (const std::shared\_ptr< std::vector< uint32\_t >> idList)=0
- virtual [~ICv2xRxSubscription](#) ()

#### 5.13.1.24.1 Constructors and Destructors

5.13.1.24.1.1 virtual telux::cv2x::ICv2xRxSubscription::~~ICv2xRxSubscription ( ) [virtual]

#### 5.13.1.24.2 Member Function Documentation

5.13.1.24.2.1 virtual uint32\_t telux::cv2x::ICv2xRxSubscription::getSubscriptionId ( ) const [pure virtual]

Accessor for Rx subscription ID

#### Returns

subscription ID

5.13.1.24.2.2 virtual TrafficIpType telux::cv2x::ICv2xRxSubscription::getIpType ( ) const [pure virtual]

Accessor for IP traffic type

#### Returns

The Rx subscriptions's IP traffic type (IP or NON-IP)

**5.13.1.24.2.3** `virtual int telux::cv2x::ICv2xRxSubscription::getSock ( ) const [pure virtual]`

Accessor for the socket file descriptor

**Returns**

The Rx subscriptions's socket fd.

**5.13.1.24.2.4** `virtual struct sockaddr_in6 telux::cv2x::ICv2xRxSubscription::getSockAddr ( ) const [pure virtual]`

Accessor for the socket address description

**Returns**

The Rx subscriptions's socket address

**5.13.1.24.2.5** `virtual uint16_t telux::cv2x::ICv2xRxSubscription::getPortNum ( ) const [pure virtual]`

Accessor for the subscriptions's port number

**Returns**

The Rx subscriptions's port num

**5.13.1.24.2.6** `virtual std::shared_ptr<std::vector<uint32_t> > telux::cv2x::ICv2xRxSubscription::getServiceIDList ( ) const [pure virtual]`

Get subscriptions's service ID list

**Returns**

The Rx subscriptions's service ID list

**5.13.1.24.2.7** `virtual void telux::cv2x::ICv2xRxSubscription::setServiceIDList ( const std::shared_ptr<std::vector< uint32_t >> idList ) [pure virtual]`

Set subscriptions's service ID list

**Parameters**

in	<i>idList</i>	- the subscriptions's service ID list
----	---------------	---------------------------------------

### 5.13.1.25 class telux::cv2x::ICv2xTxFlow

This class encapsulates a Cv2xRadio Tx flows. It contains the Tx socket associated with the flow through which client applications can send data. This class is referenced in Cv2xRadio::createTxSpsFlow, Cv2xRadio::createTxEventFlow, and Cv2xRadio::closeTxFlow

#### Public member functions

- virtual uint32\_t [getFlowId](#) () const =0
- virtual [TrafficIpType](#) [getIpType](#) () const =0
- virtual uint32\_t [getServiceId](#) () const =0
- virtual int [getSock](#) () const =0
- virtual struct sockaddr\_in6 [getSockAddr](#) () const =0
- virtual uint16\_t [getPortNum](#) () const =0
- virtual [~ICv2xTxFlow](#) ()

#### 5.13.1.25.1 Constructors and Destructors

5.13.1.25.1.1 virtual telux::cv2x::ICv2xTxFlow::~~ICv2xTxFlow ( ) [virtual]

#### 5.13.1.25.2 Member Function Documentation

5.13.1.25.2.1 virtual uint32\_t telux::cv2x::ICv2xTxFlow::getFlowId ( ) const [pure virtual]

Accessor for flow ID. The flow ID should be unique within a process but will not be unique between processes.

#### Returns

flow ID

5.13.1.25.2.2 virtual TrafficIpType telux::cv2x::ICv2xTxFlow::getIpType ( ) const [pure virtual]

Accessor for IP traffic type

#### Returns

The flow's IP traffic type (IP or NON-IP)

5.13.1.25.2.3 virtual uint32\_t telux::cv2x::ICv2xTxFlow::getServiceId ( ) const [pure virtual]

Accessor for service ID

#### Returns

The flow's Service ID.

**5.13.1.25.2.4 virtual int telux::cv2x::ICv2xTxFlow::getSock ( ) const [pure virtual]**

Accessor for the socket file descriptor

**Returns**

The flow's socket fd.

**5.13.1.25.2.5 virtual struct sockaddr\_in6 telux::cv2x::ICv2xTxFlow::getSockAddr ( ) const [pure virtual]**

Accessor for the socket address description

**Returns**

The flow's socket address

**5.13.1.25.2.6 virtual uint16\_t telux::cv2x::ICv2xTxFlow::getPortNum ( ) const [pure virtual]**

Accessor for the flow's source port number

**Returns**

The flow's source port num

**5.13.1.26 class telux::cv2x::ICv2xTxRxSocket**

This class encapsulates a Cv2xRadio socket for both Tx and Rx. It contains the socket through which client applications can send and receive data. This class is referenced in Cv2xRadio::createCv2xTcpSocket and Cv2xRadio::closeCv2xTcpSocket.

**Public member functions**

- virtual uint32\_t [getId](#) ( ) const =0
- virtual uint32\_t [getServiceId](#) ( ) const =0
- virtual int [getSocket](#) ( ) const =0
- virtual struct sockaddr\_in6 [getSocketAddr](#) ( ) const =0
- virtual uint16\_t [getPortNum](#) ( ) const =0
- virtual [~ICv2xTxRxSocket](#) ( )

**5.13.1.26.1 Constructors and Destructors****5.13.1.26.1.1 virtual telux::cv2x::ICv2xTxRxSocket::~~ICv2xTxRxSocket ( ) [virtual]**

### 5.13.1.26.2 Member Function Documentation

#### 5.13.1.26.2.1 `virtual uint32_t telux::cv2x::ICv2xTxRxSocket::getId ( ) const [pure virtual]`

Accessor for Cv2xRadio socket ID. The socket ID should be unique within a process but will not be unique between processes.

##### Returns

Cv2xRadio socket ID

#### 5.13.1.26.2.2 `virtual uint32_t telux::cv2x::ICv2xTxRxSocket::getServiceId ( ) const [pure virtual]`

Accessor for service ID

##### Returns

The Service ID bound to the socket.

#### 5.13.1.26.2.3 `virtual int telux::cv2x::ICv2xTxRxSocket::getSocket ( ) const [pure virtual]`

Accessor for the socket file descriptor

##### Returns

The socket fd.

#### 5.13.1.26.2.4 `virtual struct sockaddr_in6 telux::cv2x::ICv2xTxRxSocket::getSocketAddr ( ) const [pure virtual]`

Accessor for the socket address description

##### Returns

The socket address

#### 5.13.1.26.2.5 `virtual uint16_t telux::cv2x::ICv2xTxRxSocket::getPortNum ( ) const [pure virtual]`

Accessor for the local port number bound to the socket

##### Returns

The local port number

## 5.13.2 Enumeration Type Documentation

**5.13.2.1 enum telux::cv2x::TrafficCategory [strong]**

Defines CV2X Traffic Types.

Used in Cv2xRadioManager::getCv2xRadio

**Enumerator**

**SAFETY\_TYPE** Safety message traffic category  
**NON\_SAFETY\_TYPE** Non-safety message traffic category

**5.13.2.2 enum telux::cv2x::Cv2xStatusType [strong]**

Defines possible values for CV2X radio RX/TX status.

Used in [Cv2xStatus](#)

**Enumerator**

**INACTIVE** RX/TX is inactive  
**ACTIVE** RX/TX is active  
**SUSPENDED** RX/TX is suspended  
**UNKNOWN** RX/TX status unknown

**5.13.2.3 enum telux::cv2x::Cv2xCauseType [strong]**

Defines possible values for cause of CV2X radio failure.

Used in [Cv2xStatus](#)

**Enumerator**

**TIMING** V2X timing is not valid  
**CONFIG** No valid V2X configuration  
**UE\_MODE** V2X is not supported in current UE mode  
**GEOPOLYGON** V2X is not supported in current UE location  
**THERMAL** Device's temperature is high and is in thermal mitigation mode  
**THERMAL\_ECALL** Device is in an emergency call and the device's temperature has crossed a threshold resulting in thermal mitigation  
**GEOPOLYGON\_SWITCH** V2X stack is suspended due to geopolygon switch  
**SENSING** V2X stack is suspended due to sensing  
**LPM** V2X is not supported under Low Power Mode  
**UNKNOWN** Cause is unknown

**5.13.2.4 enum telux::cv2x::TrafficIpType [strong]**

Defines CV2X traffic type in terms of IP or NON-IP.

Used in createRxSock, createTxSpsSock, and createTxEventSock

**Enumerator**

**TRAFFIC\_IP** IP message traffic  
**TRAFFIC\_NON\_IP** NON-IP message traffic

### 5.13.2.5 enum telux::cv2x::RadioConcurrencyMode [strong]

Defines CV2X modes of concurrency with cellular WWAN.

Used in [Cv2xRadioCapabilities](#)

#### Enumerator

**WWAN\_NONCONCURRENT** No simultaneous WWAN + CV2X on this interface

**WWAN\_CONCURRENT** Interface supports requests for concurrent WWAN + CV2X connections.

### 5.13.2.6 enum telux::cv2x::Cv2xEvent [strong]

Defines CV2X status change events. The state can change in response to the loss of timing precision or a geofencing change.

Used in onStatusChanged in [ICv2xRadioListener](#)

#### Enumerator

**CV2X\_INACTIVE**

**CV2X\_ACTIVE**

**TX\_SUSPENDED**

**TXRX\_SUSPENDED**

### 5.13.2.7 enum telux::cv2x::Priority [strong]

Range of supported priority levels, where a lower number means a higher priority. For example, 8 is the current 3GPP standard.

Used in [Cv2xRadioCapabilities](#) and [SpsFlowInfo](#)

#### Enumerator

**MOST\_URGENT**

**PRIORITY\_1**

**PRIORITY\_2**

**PRIORITY\_3**

**PRIORITY\_4**

**PRIORITY\_5**

**PRIORITY\_6**

**PRIORITY\_BACKGROUND**

**PRIORITY\_UNKNOWN**

### 5.13.2.8 enum telux::cv2x::Periodicity [strong]

Range of supported periodicities in milliseconds.

Used in [Cv2xRadioCapabilities](#) and [SpsFlowInfo](#)

: enum class not going to be supported in future releases. Clients should stop using this. Once a class has been marked as Deprecated, the class could be removed in future releases.

**Enumerator**

**PERIODICITY\_10MS**  
**PERIODICITY\_20MS**  
**PERIODICITY\_50MS**  
**PERIODICITY\_100MS**  
**PERIODICITY\_UNKNOWN**

**5.13.2.9 enum telux::cv2x::ConfigSourceType [strong]**

V2X configuration source types listed in ascending order of priority. The system always uses the V2X configuration with the highest priority if multiple V2X configuration sources exist.

Used in [ConfigEventInfo](#)

**Enumerator**

**UNKNOWN** V2X config file source is unknown  
**PRECONFIG** V2X config file source is preconfig  
**SIM\_CARD** V2X config file source is SIM card  
**OMA\_DM** V2X config file source is OMA-DM

**5.13.2.10 enum telux::cv2x::ConfigEvent [strong]**

Defines possible values for the events relevant to CV2X config file.

Used in [ConfigEventInfo](#)

**Enumerator**

**CHANGED** V2X config file is changed  
**EXPIRED** V2X config file is expired

**5.13.2.11 enum telux::cv2x::RFTxStatus [strong]**

Fault detection for Tx chain that including PA and front end.

Used in [RFTxInfo](#)

**Enumerator**

**INACTIVE** The Tx chain is not working.  
**OPERATIONAL** The Tx chain is operational.  
**FAULT** Fault detected on the Tx chain.

**5.13.2.12 enum telux::cv2x::SegmentType [strong]**

Defines possible values for the segment type of a transport block.

Used in [TxStatusReport](#)

**Enumerator**

**FIRST** V2X packet is segmented, it's the first transport block.



**LAST** V2X packet is segmented, it's the last transport block.

**MIDDLE** V2X packet is segmented, it's a transport block between first and last.

**ONLY\_ONE** V2X packet is not segmented, it's the only one transport block.

### 5.13.2.13 enum telux::cv2x::TxType [strong]

Defines new Tx or re-Tx type relevant to a transport block.

Used in [TxStatusReport](#)

#### Enumerator

**NEW\_TX** New Tx of the V2X transport block.

**RE\_TX** Re-Tx of the V2X transport block.

## 5.13.3 Variable Documentation

### 5.13.3.1 constexpr uint8\_t telux::cv2x::MAX\_ANTENNAS\_SUPPORTED = 2u

Defines Maximum number of antennas that is supported.

Used in [TxStatusReport](#)

## 5.14 Audio

This section contains APIs related to Audio Stream operation.

### 5.14.1 Data Structure Documentation

#### 5.14.1.1 struct telux::audio::FormatParams

Frame format common parameters

#### 5.14.1.2 struct telux::audio::AmrwbpParams

Frame format codec specific parameters

##### Data Fields

- [uint32\\_t bitWidth](#)
- [AmrwbpFrameFormat frameFormat](#)

##### 5.14.1.2.1 Field Documentation

###### 5.14.1.2.1.1 uint32\_t telux::audio::AmrwbpParams::bitWidth

Bitwidth of Stream, Typical Values <16/24>.

###### 5.14.1.2.1.2 AmrwbpFrameFormat telux::audio::AmrwbpParams::frameFormat

#### 5.14.1.3 struct telux::audio::StreamConfig

Common Stream configuration parameters

##### Data fields

Type	Field	Description
<a href="#">StreamType</a>	type	
int	modemSubId	Represents modem Subscription ID, Default set to 1. Applicable only for Voice Call
uint32_t	sampleRate	Sample Rate of Stream, Typical Values <8k/16k/32k/48k>
<a href="#">ChannelType↔ Mask</a>	<a href="#">channelType↔ Mask</a>	
<a href="#">AudioFormat</a>	format	
vector< <a href="#">DeviceType</a> >	deviceTypes	
vector< <a href="#">Direc- tion</a> >	voicePaths	Represent voice path direction for in call audio. TX for Uplink and RX for Downlink.>
<a href="#">FormatParams</a> *	formatParams	

### 5.14.1.4 struct telux::audio::FormatInfo

Represents information about the audio format.

#### Data fields

Type	Field	Description
uint32_t	sampleRate	Sample Rate of audio, Typical Values <8k/16k/32k/48k>
<a href="#">ChannelType</a> ↔ <a href="#">Mask</a>	mask	parameter for configuration of channel type
<a href="#">AudioFormat</a>	format	Represents audio format
<a href="#">FormatParams</a> *	params	Represents codec specific parameters, like Frame Format

### 5.14.1.5 struct telux::audio::ChannelVolume

Stream Channel Volume parameters

#### Data fields

Type	Field	Description
<a href="#">ChannelType</a>	channelType	
float	vol	Volume range in float <0 to 1.0>. 0 represents min volume, 1 represents max volume

### 5.14.1.6 struct telux::audio::StreamVolume

Stream Channel Volume parameters consolidating entire Stream

#### Data fields

Type	Field	Description
vector< <a href="#">ChannelVolume</a> >	volume	
<a href="#">Stream</a> ↔ <a href="#">Direction</a>	dir	

### 5.14.1.7 struct telux::audio::StreamMute

Stream Mute parameters

#### Data fields

Type	Field	Description
bool	enable	enable or disable mute on stream
<a href="#">Stream</a> ↔ <a href="#">Direction</a>	dir	

### 5.14.1.8 struct telux::audio::StreamBuffer

Stream Data Buffer

#### Data fields

Type	Field	Description
vector< uint8↔ _t >	buffer	Buffer with Size encapsulated
size_t	offset	Actual Buffer Content starting position
int64_t	timestamp	For future use

### 5.14.1.9 struct telux::audio::DtmfTone

DTMF tone parameters

#### Data fields

Type	Field	Description
<a href="#">DtmfLowFreq</a>	lowFreq	
<a href="#">DtmfHighFreq</a>	highFreq	
<a href="#">Stream↔ Direction</a>	direction	

### 5.14.1.10 class telux::audio::AudioFactory

[AudioFactory](#) allows creation of audio manager.

#### Public member functions

- std::shared\_ptr< [IAudioManager](#) > [getAudioManager](#) ()

#### Static Public Member Functions

- static [AudioFactory](#) & [getInstance](#) ()

#### 5.14.1.10.1 Member Function Documentation

##### 5.14.1.10.1.1 static [AudioFactory](#)& telux::audio::AudioFactory::getInstance ( ) [static]

Get Audio Factory instance.

##### 5.14.1.10.1.2 std::shared\_ptr<[IAudioManager](#)> telux::audio::AudioFactory::getAudioManager ( )

Get instance of audio manager.

#### Returns

[IAudioManager](#) pointer.

### 5.14.1.11 class telux::audio::IVoiceListener

Listener class for getting notifications related to DTMF tone detection. The client needs to implement these methods as briefly as possible and avoid blocking calls in it. The methods in this class can be invoked from multiple different threads. Client needs to make sure that the implementation is thread-safe.

#### Public member functions

- virtual void [onDtmfToneDetection](#) ([DtmfTone](#) dtmfTone)
- virtual [~IVoiceListener](#) ()

#### 5.14.1.11.1 Constructors and Destructors

5.14.1.11.1 virtual telux::audio::IVoiceListener::~~IVoiceListener ( ) [[virtual](#)]

Destructor of [IVoiceListener](#)

#### 5.14.1.11.2 Member Function Documentation

5.14.1.11.2.1 virtual void telux::audio::IVoiceListener::onDtmfToneDetection ( [DtmfTone](#) *dtmfTone* ) [[virtual](#)]

This function is called when a DTMF tone is detected in the voice stream

#### Parameters

in	<i>dtmfTone</i>	DTMF tone properties
----	-----------------	----------------------

### 5.14.1.12 class telux::audio::IPlayListener

#### Public member functions

- virtual void [onReadyForWrite](#) ()
- virtual void [onPlayStopped](#) ()
- virtual [~IPlayListener](#) ()

#### 5.14.1.12.1 Constructors and Destructors

5.14.1.12.1.1 virtual telux::audio::IPlayListener::~~IPlayListener ( ) [[virtual](#)]

Destructor of [IPlayListener](#)

#### 5.14.1.12.2 Member Function Documentation

#### 5.14.1.12.2.1 virtual void telux::audio::IPlayListener::onReadyForWrite ( ) [virtual]

This function is called when pipeline is ready to accept new buffer. It is applicable only for compressed audio format type where a client can write and queue buffers for playback.

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.14.1.12.2.2 virtual void telux::audio::IPlayListener::onPlayStopped ( ) [virtual]

This function is called when stopAudio() is called with [StopType::STOP\\_AFTER\\_PLAY](#). It indicates that all the buffers that were present in the pipeline have been played.

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.14.1.13 class telux::audio::ITranscodeListener

#### Public member functions

- virtual void [onReadyForWrite](#) ( )
- virtual [~ITranscodeListener](#) ( )

#### 5.14.1.13.1 Constructors and Destructors

##### 5.14.1.13.1.1 virtual telux::audio::ITranscodeListener::~~ITranscodeListener ( ) [virtual]

Destructor of [ITranscodeListener](#)

#### 5.14.1.13.2 Member Function Documentation

##### 5.14.1.13.2.1 virtual void telux::audio::ITranscodeListener::onReadyForWrite ( ) [virtual]

This function is called when pipeline is ready to accept new buffer. It is applicable only for compressed audio format type where a client can write and queue buffers for transcoding.

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.14.1.14 class telux::audio::IAudioListener

#### Public member functions

- virtual [~IAudioListener](#) ( )

### 5.14.1.14.1 Constructors and Destructors

5.14.1.14.1.1 `virtual telux::audio::IAudioListener::~IAudioListener ( ) [virtual]`

Destructor of [IAudioListener](#)

### 5.14.1.15 class telux::audio::IAudioBuffer

Stream Buffer manages the buffer to be used for read and write operations on Audio Streams. For write operations, applications should request a stream buffer, populate it with the data and then pass it to the write operation and set the dataSize that is to be written to the stream. Similarly for read operations, the application should request a stream buffer and use that in the read operation. At the end of the read, the stream buffer will contain the data read. Once an operation (read/write) has completed, the stream buffer could be reused for a subsequent read/write operation, provided [reset\(\)](#) API called on stream buffer between subsequent calls.

#### Public member functions

- virtual `size_t` [getMinSize](#) ()=0
- virtual `size_t` [getMaxSize](#) ()=0
- virtual `uint8_t *` [getRawBuffer](#) ()=0
- virtual `uint32_t` [getDataSize](#) ()=0
- virtual void [setDataSize](#) (`uint32_t` size)=0
- virtual `telux::common::Status` [reset](#) ()=0
- virtual `~IAudioBuffer` ()

### 5.14.1.15.1 Constructors and Destructors

5.14.1.15.1.1 `virtual telux::audio::IAudioBuffer::~IAudioBuffer ( ) [virtual]`

### 5.14.1.15.2 Member Function Documentation

5.14.1.15.2.1 `virtual size_t telux::audio::IAudioBuffer::getMinSize ( ) [pure virtual]`

Returns the minimum size (in bytes) of data that caller needs to read/write before calling a read/write operation on the stream.

#### Returns

minimum size

**5.14.1.15.2.2 virtual size\_t telux::audio::IAudioBuffer::getMaxSize ( ) [pure virtual]**

Returns the maximum size (in bytes) that the buffer can hold.

**Returns**

maximum size

**5.14.1.15.2.3 virtual uint8\_t\* telux::audio::IAudioBuffer::getRawBuffer ( ) [pure virtual]**

Gets the raw buffer that [IStreamBuffer](#) manages. Application should write in between(include) of [getMinSize\(\)](#) to [getMaxSize\(\)](#) number of bytes in this buffer. Application is not responsible to free the raw buffer. It will be free'd when the [IStreamBuffer](#) is destroyed.

**Returns**

raw buffer

**5.14.1.15.2.4 virtual uint32\_t telux::audio::IAudioBuffer::getDataSize ( ) [pure virtual]**

Gets the size (in bytes) of valid data present in the buffer.

**Returns**

size of valid data in the buffer

**5.14.1.15.2.5 virtual void telux::audio::IAudioBuffer::setDataSize ( uint32\_t size ) [pure virtual]**

Sets the size (in bytes) of valid data present in the buffer.

**Parameters**

<i>size</i>	size of valid data in the buffer
-------------	----------------------------------

**5.14.1.15.2.6 virtual telux::common::Status telux::audio::IAudioBuffer::reset ( ) [pure virtual]**

Reset all state and data of the buffer. This is to be called when reusing the same buffer for multiple operations.

**Returns**

status Status of the operation

**5.14.1.16 class telux::audio::IStreamBuffer**



**Public member functions**

- virtual [~IStreamBuffer](#) ()

**5.14.1.16.1 Constructors and Destructors**

**5.14.1.16.1.1** virtual [telux::audio::IStreamBuffer::~IStreamBuffer](#) ( ) [**virtual**]

**5.14.1.17 class telux::audio::IAudioManager**

Audio Manager is a primary interface for audio operations. It provide APIs to manage Streams ( like voice, play, record etc) and sound cards.

**Public member functions**

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status getDevices](#) ([GetDevicesResponseCb](#) callback=nullptr)=0
- virtual [telux::common::Status getStreamTypes](#) ([GetStreamTypesResponseCb](#) callback=nullptr)=0
- virtual [telux::common::Status createStream](#) ([StreamConfig](#) streamConfig, [CreateStreamResponseCb](#) callback=nullptr)=0
- virtual [telux::common::Status createTranscoder](#) ([FormatInfo](#) input, [FormatInfo](#) output, [CreateTranscoderResponseCb](#) callback)=0
- virtual [telux::common::Status deleteStream](#) (std::shared\_ptr< [IAudioStream](#) > stream, [DeleteStreamResponseCb](#) callback=nullptr)=0
- virtual [telux::common::Status registerListener](#) (std::weak\_ptr< [IAudioListener](#) > listener)=0
- virtual [telux::common::Status deRegisterListener](#) (std::weak\_ptr< [IAudioListener](#) > listener)=0
- virtual [~IAudioManager](#) ()

**5.14.1.17.1 Constructors and Destructors**

**5.14.1.17.1.1** virtual [telux::audio::IAudioManager::~IAudioManager](#) ( ) [**virtual**]

**5.14.1.17.2 Member Function Documentation**

**5.14.1.17.2.1** virtual bool [telux::audio::IAudioManager::isSubsystemReady](#) ( ) [**pure virtual**]

Checks the status of audio subsystems and returns the result.

**Returns**

If true that means AudioManager is ready for performing audio operations.

**5.14.1.17.2.2** `virtual std::future<bool> telux::audio::IAudioManager::onSubsystemReady ( ) [pure virtual]`

Wait for Audio subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when audio subsystem is ready.

**5.14.1.17.2.3** `virtual telux::common::Status telux::audio::IAudioManager::getDevices ( GetDevicesResponseCb callback = nullptr ) [pure virtual]`

Get the list of supported audio devices, which are currently supported in the audio subsystem

#### Parameters

in	<i>callback</i>	callback pointer to get the response of getDevices.
----	-----------------	---

#### Returns

Status of request i.e. success or suitable status code.

**5.14.1.17.2.4** `virtual telux::common::Status telux::audio::IAudioManager::getStreamTypes ( GetStreamTypesResponseCb callback = nullptr ) [pure virtual]`

Get the list of supported audio streams types, which are currently supported in the audio subsystem

#### Parameters

in	<i>callback</i>	callback pointer to get the response of getStreamTypes.
----	-----------------	---

#### Returns

Status of request i.e. success or suitable status code.

**5.14.1.17.2.5** `virtual telux::common::Status telux::audio::IAudioManager::createStream ( StreamConfig streamConfig, CreateStreamResponseCb callback = nullptr ) [pure virtual]`

Creates the stream for audio operation

#### Parameters

in	<i>streamConfig</i>	stream configuration.
in	<i>callback</i>	callback pointer to get the response of createStream.

#### Returns

Status of request i.e. success or suitable status code.

**5.14.1.17.2.6** `virtual telux::common::Status telux::audio::IAudioManager::createTranscoder ( FormatInfo input, FormatInfo output, CreateTranscoderResponseCb callback ) [pure virtual]`

Creates an instance of transcoder that can be used for transcoding operations. Each instance returned can be used for single transcoding operation. The instance can not be used for multiple transcoding operation.

#### Parameters

in	<i>input</i>	configuration of input buffers that needs to be transcoded.
in	<i>output</i>	configuration of transcoded output buffers.
in	<i>callback</i>	callback pointer to get the response of createTranscoder.

#### Returns

Status of request i.e. success or suitable status code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.14.1.17.2.7** `virtual telux::common::Status telux::audio::IAudioManager::deleteStream ( std::shared_ptr< IAudioStream > stream, DeleteStreamResponseCb callback = nullptr ) [pure virtual]`

Deletes the specified stream which was created before

#### Parameters

in	<i>stream</i>	reference to stream to be deleted.
in	<i>callback</i>	callback pointer to get the response of deleteStream.

#### Returns

Status of request i.e. success or suitable status code.

**5.14.1.17.2.8** `virtual telux::common::Status telux::audio::IAudioManager::registerListener ( std::weak_ptr< IAudioListener > listener ) [pure virtual]`

Register a listener to get notified when service status changes.

#### Parameters

in	<i>listener</i>	Pointer of IServiceListener object that processes the notification
----	-----------------	--

**Returns**

Status of registerListener i.e success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

#### 5.14.1.17.2.9 virtual telux::common::Status telux::audio::IAudioManager::deRegisterListener ( std::weak\_ptr< IAudioListener > *listener* ) [pure virtual]

Remove a previously registered listener.

**Parameters**

in	<i>listener</i>	Previously registered IServiceListener that needs to be removed
----	-----------------	---

**Returns**

Status of deRegisterListener, success or suitable status code

**Note**

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

### 5.14.1.18 class telux::audio::IAudioDevice

Audio device and it's characteristics like Direction (Sink or Source), type.

**Public member functions**

- virtual [DeviceType](#) [getType](#) ()=0
- virtual [DeviceDirection](#) [getDirection](#) ()=0
- virtual [~IAudioDevice](#) ()

#### 5.14.1.18.1 Constructors and Destructors

##### 5.14.1.18.1.1 virtual telux::audio::IAudioDevice::~IAudioDevice ( ) [virtual]

#### 5.14.1.18.2 Member Function Documentation

**5.14.1.18.2.1 virtual DeviceType telux::audio::IAudioDevice::getType ( ) [pure virtual]**

Get the type of Device (i.e SPEAKER, MIC etc)

**Returns**

DeviceType

**5.14.1.18.2.2 virtual DeviceDirection telux::audio::IAudioDevice::getDirection ( ) [pure virtual]**

Provide direction of device whether is Sink for audio data ( RX i.e. speaker, etc) or Source for audio data ( TX i.e. mic, etc)

**Returns**

DeviceDirection

**5.14.1.19 class telux::audio::IAudioStream**

[IAudioStream](#) represents single audio stream with base properties.

**Public member functions**

- virtual [StreamType](#) [getType](#) ()=0
- virtual [telux::common::Status](#) [setDevice](#) (std::vector< [DeviceType](#) > devices, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [getDevice](#) ([GetStreamDeviceResponseCb](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [setVolume](#) ([StreamVolume](#) volume, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [getVolume](#) ([StreamDirection](#) dir, [GetStreamVolumeResponseCb](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [setMute](#) ([StreamMute](#) mute, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [getMute](#) ([StreamDirection](#) dir, [GetStreamMuteResponseCb](#) callback=nullptr)=0
- virtual [~IAudioStream](#) ()

**5.14.1.19.1 Constructors and Destructors****5.14.1.19.1.1 virtual telux::audio::IAudioStream::~IAudioStream ( ) [virtual]****5.14.1.19.2 Member Function Documentation**

**5.14.1.19.2.1 virtual StreamType telux::audio::IAudioStream::getType ( ) [pure virtual]**

Get the stream type like VOICE, PLAY, CAPTURE

**Returns**

StreamType

**5.14.1.19.2.2 virtual telux::common::Status telux::audio::IAudioStream::setDevice ( std::vector< DeviceType > *devices*, telux::common::ResponseCallback *callback* = nullptr ) [pure virtual]**

Set Device of audio stream

**Parameters**

in	<i>devices</i>	Devices list.
in	<i>callback</i>	callback to get the response of setDevice.

**Returns**

Status of the request i.e. success or suitable status code.

**5.14.1.19.2.3 virtual telux::common::Status telux::audio::IAudioStream::getDevice ( GetStream↔ DeviceResponseCb *callback* = nullptr ) [pure virtual]**

Get Device of audio stream

**Parameters**

in	<i>callback</i>	callback to get the response of getDevice
----	-----------------	---

**Returns**

Status of the request i.e. success or suitable status code.

**5.14.1.19.2.4 virtual telux::common::Status telux::audio::IAudioStream::setVolume ( StreamVolume *volume*, telux::common::ResponseCallback *callback* = nullptr ) [pure virtual]**

Set Volume of audio stream

**Parameters**

in	<i>volume</i>	volume setting per channel for direction.
in	<i>callback</i>	callback to get the response of setVolume.

**Returns**

Status of the request i.e. success or suitable status code.

#### 5.14.1.19.2.5 virtual telux::common::Status telux::audio::IAudioStream::getVolume ( StreamDirection *dir*, GetStreamVolumeResponseCb *callback* = nullptr ) [pure virtual]

Get Volume of audio stream

##### Parameters

in	<i>dir</i>	Stream Direction to query volume details.
in	<i>callback</i>	callback to get the response of getVolume.

##### Returns

Status of the request i.e. success or suitable status code.

#### 5.14.1.19.2.6 virtual telux::common::Status telux::audio::IAudioStream::setMute ( StreamMute *mute*, telux::common::ResponseCallback *callback* = nullptr ) [pure virtual]

Set Mute of audio stream

##### Parameters

in	<i>mute</i>	mute setting for direction.
in	<i>callback</i>	callback to know the status of the request.

##### Returns

Status of the request i.e. success or suitable status code.

#### 5.14.1.19.2.7 virtual telux::common::Status telux::audio::IAudioStream::getMute ( StreamDirection *dir*, GetStreamMuteResponseCb *callback* = nullptr ) [pure virtual]

Get Mute of audio stream

##### Parameters

in	<i>dir</i>	Stream Direction to query mute details.
in	<i>callback</i>	callback to get the response of getMute.

##### Returns

Status of the request i.e. success or suitable status code.

### 5.14.1.20 class telux::audio::IAudioVoiceStream

[IAudioVoiceStream](#) represents single voice stream.

**Public member functions**

- virtual `telux::common::Status startAudio (telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status stopAudio (telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status playDtmfTone (DtmfTone dtmfTone, uint16_t duration, uint16_t gain, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status stopDtmfTone (StreamDirection direction, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status registerListener (std::weak_ptr< IVoiceListener > listener, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status deRegisterListener (std::weak_ptr< IVoiceListener > listener)=0`
- virtual `~IAudioVoiceStream ()`

**5.14.1.20.1 Constructors and Destructors**

5.14.1.20.1.1 virtual `telux::audio::IAudioVoiceStream::~~IAudioVoiceStream ( ) [virtual]`

**5.14.1.20.2 Member Function Documentation**

5.14.1.20.2.1 virtual `telux::common::Status telux::audio::IAudioVoiceStream::startAudio ( telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Starts audio stream

**Parameters**

<code>in</code>	<code>callback</code>	callback to get the response of startAudio.
-----------------	-----------------------	---

**Returns**

Status of the request i.e. success or suitable status code.

5.14.1.20.2.2 virtual `telux::common::Status telux::audio::IAudioVoiceStream::stopAudio ( telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Stops audio stream

**Parameters**

<code>in</code>	<code>callback</code>	callback to get the response of stopAudio.
-----------------	-----------------------	--

**Returns**

Status of the request i.e. success or suitable status code.



**5.14.1.20.2.3** virtual telux::common::Status telux::audio::IAudioVoiceStream::playDtmfTone ( DtmfTone dtmfTone, uint16\_t duration, uint16\_t gain, telux::common::ResponseCallback callback = nullptr ) [pure virtual]

Plays in-band DTMF tone on the active voice stream

#### Parameters

in	<i>dtmfTone</i>	DTMF tone properties [in] duration Duration (in milliseconds) for which the tone needs to be played. The constant infiniteDtmfDuration(=0xFFFF) represents infinite duration. [in] gain DTMF tone gain [in] callback callback to get the response of playDtmfTone.
----	-----------------	--

#### Returns

Status of the request i.e. success or suitable status code.

**5.14.1.20.2.4** virtual telux::common::Status telux::audio::IAudioVoiceStream::stopDtmfTone ( StreamDirection direction, telux::common::ResponseCallback callback = nullptr ) [pure virtual]

Stops the DTMF tone which is being played (i.e duration not expired) on the active voice stream

#### Parameters

in	<i>direction</i>	Direction associated with the DTMF tone @ [in] callback callback to get the response of stopDtmfTone.
----	------------------	---

#### Returns

Status of the request i.e. success or suitable status code.

**5.14.1.20.2.5** virtual telux::common::Status telux::audio::IAudioVoiceStream::registerListener ( std::weak\_ptr< IVoiceListener > listener, telux::common::ResponseCallback callback = nullptr ) [pure virtual]

Register a listener to get notified when a DTMF tone is detected in the active voice stream

#### Parameters

in	<i>listener</i>	Pointer of <a href="#">IVoiceListener</a> object that processes the notification [in] callback callback to get the response of registerListener
----	-----------------	---

#### Returns

Status of registerListener i.e success or suitable status code.

#### 5.14.1.20.2.6 virtual telux::common::Status telux::audio::IAudioVoiceStream::deRegisterListener ( std::weak\_ptr< IVoiceListener > *listener* ) [pure virtual]

Remove a previously registered listener.

##### Parameters

in	<i>listener</i>	Previously registered <a href="#">IVoiceListener</a> that needs to be removed
----	-----------------	---

##### Returns

Status of deRegisterListener, success or suitable status code

### 5.14.1.21 class telux::audio::IAudioPlayStream

[IAudioPlayStream](#) represents single audio playback stream.

##### Public member functions

- virtual std::shared\_ptr< [IStreamBuffer](#) > [getStreamBuffer](#) ()=0
- virtual telux::common::Status [write](#) (std::shared\_ptr< [IStreamBuffer](#) > buffer, [WriteResponseCb](#) callback=nullptr)=0
- virtual telux::common::Status [stopAudio](#) ([StopType](#) stopType, telux::common::ResponseCallback callback=nullptr)=0
- virtual telux::common::Status [registerListener](#) (std::weak\_ptr< [IPlayListener](#) > listener)=0
- virtual telux::common::Status [deRegisterListener](#) (std::weak\_ptr< [IPlayListener](#) > listener)=0
- virtual [~IAudioPlayStream](#) ()

#### 5.14.1.21.1 Constructors and Destructors

##### 5.14.1.21.1.1 virtual telux::audio::IAudioPlayStream::~~IAudioPlayStream ( ) [virtual]

#### 5.14.1.21.2 Member Function Documentation

##### 5.14.1.21.2.1 virtual std::shared\_ptr<IStreamBuffer> telux::audio::IAudioPlayStream::getStreamBuffer ( ) [pure virtual]

Get an Audio [StreamBuffer](#) to be used for playback operations

##### Returns

an Audio Buffer or a nullptr in case of error

#### 5.14.1.21.2.2 `virtual telux::common::Status telux::audio::IAudioPlayStream::write ( std::shared_ptr< IStreamBuffer > buffer, WriteResponseCb callback = nullptr ) [pure virtual]`

Write Samples to audio stream. First write starts playback operation.

Write in case of compressed audio format maintains a pipeline, if the callback returns with same number of bytes written as requested and no error occurred, user can send next buffer. If the number of bytes returned are not equal to the requested write size, then need to resend the buffer again from the leftover offset after waiting for the () event.

##### Parameters

in	<i>buffer</i>	stream buffer for write.
in	<i>callback</i>	callback to get the response of write.

##### Returns

Status of the request i.e. success or suitable status code.

#### 5.14.1.21.2.3 `virtual telux::common::Status telux::audio::IAudioPlayStream::stopAudio ( StopType stopType, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

This API is to be used to stop playback. It is applicable only for compressed audio format playback.

##### Parameters

in	<i>callback</i>	callback to get the response of stopAudio.
in	<i>stopType</i>	it specifies type of stop for stopping audio playback.

##### Returns

Status of the request i.e. success or suitable status code.

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.14.1.21.2.4 `virtual telux::common::Status telux::audio::IAudioPlayStream::registerListener ( std::weak_ptr< IPlayListener > listener ) [pure virtual]`

Register a listener to get notified for events of Play Stream

##### Parameters

in	<i>listener</i>	Pointer of <a href="#">IPlayListener</a> object that processes the notification [in] callback callback to get the response of registerListener
----	-----------------	---

**Returns**

Status of registerListener i.e success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

#### 5.14.1.21.25 virtual telux::common::Status telux::audio::IAudioPlayStream::deRegisterListener ( std::weak\_ptr< IPlayListener > *listener* ) [pure virtual]

Remove a previously registered listener.

**Parameters**

in	<i>listener</i>	Previously registered <a href="#">IPlayListener</a> that needs to be removed
----	-----------------	--

**Returns**

Status of deRegisterListener, success or suitable status code

**Note**

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

#### 5.14.1.22 class telux::audio::IAudioCaptureStream

[IAudioCaptureStream](#) represents single audio capture stream.

**Public member functions**

- virtual std::shared\_ptr< [IStreamBuffer](#) > [getStreamBuffer](#) ()=0
- virtual [telux::common::Status read](#) (std::shared\_ptr< [IStreamBuffer](#) > buffer, uint32\_t bytesToRead, [ReadResponseCb](#) callback=nullptr)=0
- virtual [~IAudioCaptureStream](#) ()

##### 5.14.1.22.1 Constructors and Destructors

###### 5.14.1.22.1.1 virtual telux::audio::IAudioCaptureStream::~IAudioCaptureStream ( ) [virtual]

##### 5.14.1.22.2 Member Function Documentation

**5.14.1.22.2.1** `virtual std::shared_ptr<IStreamBuffer> telux::audio::IAudioCaptureStream::getStreamBuffer ( ) [pure virtual]`

Get an Audio Stream Buffer to be used for capture operations

#### Returns

an Audio Buffer or nullptr in case of failure

**5.14.1.22.2.2** `virtual telux::common::Status telux::audio::IAudioCaptureStream::read ( std::shared_ptr< IStreamBuffer > buffer, uint32_t bytesToRead, ReadResponseCb callback = nullptr ) [pure virtual]`

Read Samples from audio stream. First read starts capture operation.

#### Parameters

in	<i>buffer</i>	stream buffer for read.
in	<i>bytesToRead</i>	specifying how many bytes to be read from stream.
in	<i>callback</i>	callback to get the response of read.

#### Returns

Status of the request i.e. success or suitable status code.

### 5.14.1.23 class telux::audio::IAudioLoopbackStream

[IAudioLoopbackStream](#) represents audio loopback stream.

#### Public member functions

- virtual [telux::common::Status startLoopback](#) ([telux::common::ResponseCallback](#) *callback*=nullptr)=0
- virtual [telux::common::Status stopLoopback](#) ([telux::common::ResponseCallback](#) *callback*=nullptr)=0
- virtual [~IAudioLoopbackStream](#) ()

#### 5.14.1.23.1 Constructors and Destructors

**5.14.1.23.1.1** `virtual telux::audio::IAudioLoopbackStream::~~IAudioLoopbackStream ( ) [virtual]`

#### 5.14.1.23.2 Member Function Documentation

**5.14.1.23.2.1** `virtual telux::common::Status telux::audio::IAudioLoopbackStream::startLoopback ( telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Start loopback between source and sink devices

<i>in</i>	<i>callback</i>	callback to get the response of start loopback.
-----------	-----------------	---

**Returns**

Status of the request i.e. success or suitable status code.

#### 5.14.1.23.2.2 **virtual telux::common::Status telux::audio::IAudioLoopbackStream::stopLoopback ( telux::common::ResponseCallback *callback* = nullptr ) [pure virtual]**

Stop loopback between source and sink devices

**Parameters**

<i>in</i>	<i>callback</i>	callback to get the response of stop loopback.
-----------	-----------------	--

**Returns**

Status of the request i.e. success or suitable status code.

#### 5.14.1.24 **class telux::audio::IAudioToneGeneratorStream**

[IAudioToneGeneratorStream](#) represents tone generator stream.

**Public member functions**

- virtual [telux::common::Status playTone](#) (std::vector< uint16\_t > *freq*, uint16\_t *duration*, uint16\_t *gain*, [telux::common::ResponseCallback](#) *callback*=nullptr)=0
- virtual [telux::common::Status stopTone](#) ([telux::common::ResponseCallback](#) *callback*=nullptr)=0
- virtual [~IAudioToneGeneratorStream](#) ()

##### 5.14.1.24.1 **Constructors and Destructors**

###### 5.14.1.24.1.1 **virtual telux::audio::IAudioToneGeneratorStream::~~IAudioToneGeneratorStream ( ) [virtual]**

##### 5.14.1.24.2 **Member Function Documentation**

###### 5.14.1.24.2.1 **virtual telux::common::Status telux::audio::IAudioToneGeneratorStream::playTone ( std::vector< uint16\_t > *freq*, uint16\_t *duration*, uint16\_t *gain*, telux::common::↔ [ResponseCallback](#) *callback* = nullptr ) [pure virtual]**

Play a tone on sink devices. As the duration expires, the generated tone terminates automatically.

**Parameters**

<i>in</i>	<i>freq</i>	Accepts the composition of frequencies (in Hz) to be played such as single tone or dual tone. Any additional frequencies provided will be ignored.
-----------	-------------	--

in	<i>duration</i>	Duration (in milliseconds) for which the tone needs to be played. The constant infiniteToneDuration(=0xFFFF) represents infinite duration.
in	<i>gain</i>	Tone Gain.
in	<i>callback</i>	callback to get the response of play tone.

**Returns**

Status of the request i.e. success or suitable status code.

#### 5.14.1.24.2.2 virtual telux::common::Status telux::audio::IAudioToneGeneratorStream::stopTone ( telux::common::ResponseCallback *callback* = nullptr ) [pure virtual]

Stops the tone which is being played (i.e duration not expired) on the active Tone generator stream.

**Parameters**

in	<i>callback</i>	callback to get the response of stop tone.
----	-----------------	--

**Returns**

Status of the request i.e. success or suitable status code.

#### 5.14.1.25 class telux::audio::ITranscoder

[ITranscoder](#) is used to convert one audio format to another audio format using the transcoding operation.

**Public member functions**

- virtual std::shared\_ptr< [IAudioBuffer](#) > [getWriteBuffer](#) ()=0
- virtual std::shared\_ptr< [IAudioBuffer](#) > [getReadBuffer](#) ()=0
- virtual telux::common::Status [write](#) (std::shared\_ptr< [IAudioBuffer](#) > buffer, uint32\_t isLastBuffer, [TranscoderWriteResponseCb](#) callback=nullptr)=0
- virtual telux::common::Status [tearDown](#) (telux::common::ResponseCallback callback=nullptr)=0
- virtual telux::common::Status [read](#) (std::shared\_ptr< [IAudioBuffer](#) > buffer, uint32\_t bytesToRead, [TranscoderReadResponseCb](#) callback=nullptr)=0
- virtual telux::common::Status [registerListener](#) (std::weak\_ptr< [ITranscodeListener](#) > listener)=0
- virtual telux::common::Status [deRegisterListener](#) (std::weak\_ptr< [ITranscodeListener](#) > listener)=0

#### 5.14.1.25.1 Member Function Documentation

**5.14.1.25.1.1** `virtual std::shared_ptr<IAudioBuffer> telux::audio::ITranscoder::getWriteBuffer ( )`  
**[pure virtual]**

Get a buffer to be used for writing samples for transcoding operation.

#### Returns

a buffer or nullptr in case of failure.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.14.1.25.1.2** `virtual std::shared_ptr<IAudioBuffer> telux::audio::ITranscoder::getReadBuffer ( )`  
**[pure virtual]**

Get a buffer to be used for reading samples from transcoding operation.

#### Returns

a buffer or nullptr in case of failure.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.14.1.25.1.3** `virtual telux::common::Status telux::audio::ITranscoder::write ( std::shared_ptr<IAudioBuffer > buffer, uint32_t isLastBuffer, TranscoderWriteResponseCb callback = nullptr )` **[pure virtual]**

Write Samples/Frames to transcode stream. First write starts transcoding operation.

Write in case of compressed audio format maintains a pipeline, if the callback returns with same number of bytes written as requested and no error occurred, user can send next buffer. If the number of bytes returned are not equal to the requested write size, then user needs to resend the buffer again from the leftover offset after waiting for the () event.

#### Parameters

in	<i>buffer</i>	buffer that needs to be transcoded.
in	<i>isLastBuffer</i>	represents whether this buffer is last buffer or not. Once last buffer is set no more write operations are required.
in	<i>callback</i>	callback to get the response of write.

#### Returns

Status of the request i.e. success or suitable status code.



**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.14.1.25.1.4 virtual telux::common::Status telux::audio::ITranscoder::tearDown ( telux::common::ResponseCallback *callback* = nullptr ) [pure virtual]

It is mandatory to call this API after the end of a transcode operation or to abort a transcode operation. After this API call the [ITranscoder](#) object is no longer usable.

**Parameters**

in	<i>callback</i>	callback to get the response of tearDown.
----	-----------------	---

**Returns**

Status of the request i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.14.1.25.1.5 virtual telux::common::Status telux::audio::ITranscoder::read ( std::shared\_ptr< IAudioBuffer > *buffer*, uint32\_t *bytesToRead*, TranscoderReadResponseCb *callback* = nullptr ) [pure virtual]

Reads samples/Frames from transcoder during transcoding operation.

**Parameters**

in	<i>buffer</i>	stream buffer for read.
in	<i>bytesToRead</i>	specifying how many bytes to be read from stream.
in	<i>callback</i>	callback to get the response of read.

**Returns**

Status of the request i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.14.1.25.1.6 virtual telux::common::Status telux::audio::ITranscoder::registerListener ( std::weak\_ptr< ITranscodeListener > *listener* ) [pure virtual]

Register a listener to get notified for events of Transcoder.

in	<i>listener</i>	Pointer of <a href="#">ITranscodeListener</a> object that processes the notification.
----	-----------------	---

#### Returns

Status of registerListener i.e success or suitable status code.

#### Note

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

#### 5.14.1.25.1.7 virtual telux::common::Status telux::audio::ITranscoder::deRegisterListener ( std::weak\_ptr< ITranscodeListener > *listener* ) [pure virtual]

Remove a previously registered listener.

#### Parameters

in	<i>listener</i>	Previously registered <a href="#">ITranscodeListener</a> that needs to be removed.
----	-----------------	--

#### Returns

Status of deRegisterListener, success or suitable status code.

#### Note

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

## 5.14.2 Enumeration Type Documentation

### 5.14.2.1 enum telux::audio::DeviceType

Represent type of device like SPEAKER, MIC, etc.

#### Enumerator

***DEVICE\_TYPE\_NONE***  
***DEVICE\_TYPE\_SPEAKER***  
***DEVICE\_TYPE\_MIC***  
***DEVICE\_TYPE\_OUT\_VIRTUAL***

### 5.14.2.2 enum telux::audio::DeviceDirection [strong]

Represent Device Direction RX (Sink), Tx (Source)

#### Enumerator

**NONE**

**RX**

**TX**

### 5.14.2.3 enum telux::audio::Direction [strong]

Represent Voice Direction RX (Sink), Tx (Source)

#### Enumerator

**RX**

**TX**

### 5.14.2.4 enum telux::audio::StreamType [strong]

Represent Stream Type

#### Enumerator

**NONE**

**VOICE\_CALL** Voice Call, Provides Audio Session for an active Voice

**PLAY** Playback, Provides Audio Playback Session

**CAPTURE** Capture, Provides Audio Capture/Record Session

**LOOPBACK** Loopback, Provides loopback between source and sink devices

**TONE\_GENERATOR** Tone Generator, Generates tone on sink device

### 5.14.2.5 enum telux::audio::StreamDirection [strong]

Represent Stream Direction

#### Enumerator

**NONE**

**RX** Represents Session Directed towards Sink Device

**TX** Represents Session Directed from Source Device

### 5.14.2.6 enum telux::audio::ChannelType

Represent Stream's types of Channel

#### Enumerator

**LEFT** Represents left channel

**RIGHT** Represents right channel

### 5.14.2.7 enum telux::audio::AudioFormat [strong]

Specifies Stream data format

#### Enumerator

**UNKNOWN** Unknown format  
**PCM\_16BIT\_SIGNED** 16 bit signed PCM format  
**AMRNB** AMRNB format  
**AMRWB** AMRWB format  
**AMRWB\_PLUS** AMRWB+ format

### 5.14.2.8 enum telux::audio::DtmfLowFreq [strong]

Represents the possible lower frequencies(in Hz) in a standard DTMF tone

#### Enumerator

**FREQ\_697**  
**FREQ\_770**  
**FREQ\_852**  
**FREQ\_941**

### 5.14.2.9 enum telux::audio::DtmfHighFreq [strong]

Represents the possible higher frequencies(in Hz) in a standard DTMF tone

#### Enumerator

**FREQ\_1209**  
**FREQ\_1336**  
**FREQ\_1477**  
**FREQ\_1633**

### 5.14.2.10 enum telux::audio::AmrwbpFrameFormat [strong]

Representative of type of frame structure. Typical transport interface or file storage.

#### Enumerator

**UNKNOWN** Unknown format  
**TRANSPORT\_INTERFACE\_FORMAT**  
**FILE\_STORAGE\_FORMAT**

### 5.14.2.11 enum telux::audio::StopType [strong]

Represents type of stop for compressed audio format playback. Audio playback can be stopped in two ways force stop and after playing all buffers in the pipeline.

#### Enumerator

**FORCE\_STOP** Stop Playing Immediately and clear buffer pipeline

**STOP\_AFTER\_PLAY** Stop Play once after all buffers in pipeline are played

## 5.15 Thermal Management

This section contains APIs related to Thermal Management such as read list of thermal zones, cooling devices and binding info.

This section contains APIs related to Thermal Shutdown Management such as set/get thermal auto-shutdown mode, receive notifications on every auto-shutdown update.

### 5.15.1 Data Structure Documentation

#### 5.15.1.1 class telux::therm::ThermalFactory

[ThermalFactory](#) allows creation of thermal manager.

##### Public member functions

- `std::shared_ptr< IThermalManager > getThermalManager ()`
- `std::shared_ptr< IThermalShutdownManager > getThermalShutdownManager ()`

##### Static Public Member Functions

- static `ThermalFactory & getInstance ()`

#### 5.15.1.1.1 Member Function Documentation

##### 5.15.1.1.1.1 static ThermalFactory& telux::therm::ThermalFactory::getInstance ( ) [static]

Get Thermal Factory instance.

##### 5.15.1.1.1.2 std::shared\_ptr<IThermalManager> telux::therm::ThermalFactory::getThermalManager ( )

Get thermal manager instance to get list of thermal zones (sensors) and cooling devices supported by the device

##### Returns

Pointer of [IThermalManager](#) object.

##### 5.15.1.1.1.3 std::shared\_ptr<IThermalShutdownManager> telux::therm::ThermalFactory::getThermalShutdownManager ( )

Get thermal shutdown manager instance to control automatic thermal shutdown and get relevant notifications

##### Returns

Pointer of [IThermalShutdownManager](#) object.

### 5.15.1.2 struct telux::therm::BoundCoolingDevice

Defines the trip points to which cooling device is bound.

#### Data fields

Type	Field	Description
int	cooling↔ DeviceId	Cooling device Id associated with trip points
vector< shared_ptr< <a href="#">ITripPoint</a> > >	bindingInfo	List of trippoints bound to the cooling device

### 5.15.1.3 class telux::therm::IThermalManager

[IThermalManager](#) provides interface to get thermal zone and cooling device information.

#### Public member functions

- virtual std::vector< std::shared\_ptr< [IThermalZone](#) > > [getThermalZones](#) ()=0
- virtual std::vector< std::shared\_ptr< [ICoolingDevice](#) > > [getCoolingDevices](#) ()=0
- virtual std::shared\_ptr< [IThermalZone](#) > [getThermalZone](#) (int thermalZoneId)=0
- virtual std::shared\_ptr< [ICoolingDevice](#) > [getCoolingDevice](#) (int coolingDeviceId)=0
- virtual [~IThermalManager](#) ()

#### 5.15.1.3.1 Constructors and Destructors

5.15.1.3.1.1 virtual telux::therm::IThermalManager::~IThermalManager ( ) [virtual]

Destructor of [IThermalManager](#)

#### 5.15.1.3.2 Member Function Documentation

5.15.1.3.2.1 virtual std::vector<std::shared\_ptr<IThermalZone> > telux::therm::IThermalManager↔  
::getThermalZones ( ) [pure virtual]

Retrieves the list of thermal zone info like type, temperature and trip points.

#### Returns

List of thermal zones.

**5.15.1.3.2.2** `virtual std::vector<std::shared_ptr<ICoolingDevice> > telux::therm::IThermalManager::getCoolingDevices ( ) [pure virtual]`

Retrieves the list of thermal cooling device info like type, maximum throttle state and currently requested throttle state.

#### Returns

List of cooling devices.

**5.15.1.3.2.3** `virtual std::shared_ptr<IThermalZone> telux::therm::IThermalManager::getThermalZone ( int thermalZoneId ) [pure virtual]`

Retrieves the thermal zone details like temperature, type and trip point info for the given thermal zone identifier.

#### Parameters

in	<i>thermalZoneId</i>	Thermal zone identifier
----	----------------------	-------------------------

#### Returns

Pointer to thermal zone.

**5.15.1.3.2.4** `virtual std::shared_ptr<ICoolingDevice> telux::therm::IThermalManager::getCoolingDevice ( int coolingDeviceId ) [pure virtual]`

Retrieves the cooling device details like type of the device, maximum cooling level and current cooling level for the given cooling device identifier.

#### Parameters

in	<i>coolingDeviceId</i>	Cooling device identifier
----	------------------------	---------------------------

#### Returns

Pointer to cooling device.

### 5.15.1.4 class telux::therm::ITripPoint

[ITripPoint](#) provides interface to get trip point type, trip point temperature and hysteresis value for that trip point.

#### Public member functions

- virtual [TripType](#) `getType ()=0`
- virtual int `getThresholdTemp ()=0`
- virtual int `getHysteresis ()=0`



- virtual [~ITripPoint](#) ()

### 5.15.1.4.1 Constructors and Destructors

#### 5.15.1.4.1.1 virtual `telux::therm::ITripPoint::~ITripPoint ( )` [virtual]

Destructor of [ITripPoint](#)

### 5.15.1.4.2 Member Function Documentation

#### 5.15.1.4.2.1 virtual `TripType telux::therm::ITripPoint::getType ( )` [pure virtual]

Retrieves trip point type.

##### Returns

Type of trip point if available else return UNKNOWN.

- [TripType](#)

#### 5.15.1.4.2.2 virtual `int telux::therm::ITripPoint::getThresholdTemp ( )` [pure virtual]

Retrieves the temperature above which certain trip point will be fired.

- Units: MilliDegree Celsius

##### Returns

Threshold temperature

#### 5.15.1.4.2.3 virtual `int telux::therm::ITripPoint::getHysteresis ( )` [pure virtual]

Retrieves hysteresis value that is the difference between current temperature of the device and the temperature above which certain trip point will be fired. Units: MilliDegree Celsius

##### Returns

Hysteresis value

### 5.15.1.5 class `telux::therm::IThermalZone`

[IThermalZone](#) provides interface to get type of the sensor, the current temperature reading, trip points and the cooling devices binded etc.

##### Public member functions

- virtual `int getId ()=0`
- virtual `std::string getDescription ()=0`

- virtual int `getCurrentTemp` ()=0
- virtual int `getPassiveTemp` ()=0
- virtual std::vector< std::shared\_ptr< `ITripPoint` > > `getTripPoints` ()=0
- virtual std::vector< `BoundCoolingDevice` > `getBoundCoolingDevices` ()=0
- virtual `~IThermalZone` ()

### 5.15.1.5.1 Constructors and Destructors

#### 5.15.1.5.1.1 virtual telux::therm::IThermalZone::~IThermalZone ( ) [virtual]

Destructor of `IThermalZone`

### 5.15.1.5.2 Member Function Documentation

#### 5.15.1.5.2.1 virtual int telux::therm::IThermalZone::getId ( ) [pure virtual]

Retrieves the identifier for thermal zone.

#### Returns

Identifier for thermal zone

#### 5.15.1.5.2.2 virtual std::string telux::therm::IThermalZone::getDescription ( ) [pure virtual]

Retrieves the type of sensor.

#### Returns

Sensor type

#### 5.15.1.5.2.3 virtual int telux::therm::IThermalZone::getCurrentTemp ( ) [pure virtual]

Retrieves the current temperature of the device. Units: MilliDegree Celsius

#### Returns

Current temperature

#### 5.15.1.5.2.4 virtual int telux::therm::IThermalZone::getPassiveTemp ( ) [pure virtual]

Retrieves the temperature of passive trip point for the zone. Default value is 0. Valid values: 0 (disabled) or greater than 1000 (enabled), Units: MilliDegree Celsius

#### Returns

Temperature of passive trip point

**5.15.1.5.2.5** `virtual std::vector<std::shared_ptr<ITripPoint> > telux::therm::IThermalZone::getTripPoints ( ) [pure virtual]`

Retrieves trip point information like trip type, trip temperature and hysteresis.

#### Returns

Trip point info list

**5.15.1.5.2.6** `virtual std::vector<BoundCoolingDevice> telux::therm::IThermalZone::getBoundCoolingDevices ( ) [pure virtual]`

Retrieves the list of cooling device and the associated trip points bound to cooling device in given thermal zone.

#### Returns

List of bound cooling device for the given thermal zone.

### 5.15.1.6 class telux::therm::ICoolingDevice

[ICoolingDevice](#) provides interface to get type of the cooling device, the maximum throttle state and the currently requested throttle state of the cooling device.

#### Public member functions

- virtual int [getId](#) ()=0
- virtual std::string [getDescription](#) ()=0
- virtual int [getMaxCoolingLevel](#) ()=0
- virtual int [getCurrentCoolingLevel](#) ()=0
- virtual [~ICoolingDevice](#) ()

#### 5.15.1.6.1 Constructors and Destructors

**5.15.1.6.1.1** `virtual telux::therm::ICoolingDevice::~~ICoolingDevice ( ) [virtual]`

Destructor of [ICoolingDevice](#)

#### 5.15.1.6.2 Member Function Documentation

**5.15.1.6.2.1 virtual int telux::therm::ICoolingDevice::getId ( ) [pure virtual]**

Retrieves the identifier of the thermal cooling device.

**Returns**

Cooling device identifier

**5.15.1.6.2.2 virtual std::string telux::therm::ICoolingDevice::getDescription ( ) [pure virtual]**

Retrieves the type of the cooling device.

**Returns**

Cooling device type

**5.15.1.6.2.3 virtual int telux::therm::ICoolingDevice::getMaxCoolingLevel ( ) [pure virtual]**

Retrieves the maximum cooling level of the cooling device.

**Returns**

Maximum cooling level of the thermal cooling device

**5.15.1.6.2.4 virtual int telux::therm::ICoolingDevice::getCurrentCoolingLevel ( ) [pure virtual]**

Retrieves the current cooling level of the cooling device. This value can be between 0 and max cooling level. Max cooling level is different for different cooling devices like fan, processor etc.

**Returns**

Current cooling level of the thermal cooling device

**5.15.1.7 class telux::therm::IThermalShutdownListener**

Listener class for getting notifications when automatic thermal shutdown mode is enabled/ disabled or will be enabled imminently. The client needs to implement these methods as briefly as possible and avoid blocking calls in it. The methods in this class can be invoked from multiple different threads. Client needs to make sure that the implementation is thread-safe.

**Public member functions**

- virtual void [onShutdownEnabled](#) ( )
- virtual void [onShutdownDisabled](#) ( )
- virtual void [onImminentShutdownEnablement](#) (uint32\_t imminentDuration)
- virtual [~IThermalShutdownListener](#) ( )

### 5.15.1.7.1 Constructors and Destructors

5.15.1.7.1.1 **virtual** `telux::therm::IThermalShutdownListener::~~IThermalShutdownListener ( )`  
**[virtual]**

Destructor of [IThermalShutdownListener](#)

### 5.15.1.7.2 Member Function Documentation

5.15.1.7.2.1 **virtual void** `telux::therm::IThermalShutdownListener::onShutdownEnabled ( )`  
**[virtual]**

This function is called when the automatic shutdown mode changes to ENABLE

5.15.1.7.2.2 **virtual void** `telux::therm::IThermalShutdownListener::onShutdownDisabled ( )`  
**[virtual]**

This function is called when the automatic shutdown mode changes to DISABLE

5.15.1.7.2.3 **virtual void** `telux::therm::IThermalShutdownListener::onImminentShutdownEnablement (   
 uint32_t imminentDuration )` **[virtual]**

This function is called when the automatic shutdown mode is about to change to ENABLE. Clients that want to keep the shutdown mode disabled, needs to set it accordingly with in the `imminentDuration` time. If disabled successfully within `imminentDuration` time, the system timer for auto-enablement will be reset.

#### Parameters

<code>in</code>	<i>imminentDuration</i>	Time elapsed(in seconds) for the shutdown mode to be enabled
-----------------	-------------------------	--

### 5.15.1.8 class `telux::therm::IThermalShutdownManager`

[IThermalShutdownManager](#) class provides interface to enable/disable automatic thermal shutdown. Additionally it facilitates to register for notifications when the automatic shutdown mode changes.

#### Public member functions

- virtual bool `isReady ()=0`
- virtual `std::future< bool > onReady ()=0`
- virtual `telux::common::Status registerListener (std::weak_ptr< IThermalShutdownListener > listener)=0`
- virtual `telux::common::Status deregisterListener (std::weak_ptr< IThermalShutdownListener > listener)=0`
- virtual `telux::common::Status setAutoShutdownMode (AutoShutdownMode mode, telux::common::ResponseCallback callback=nullptr, uint32_t timeout=DEFAULT_TIMEOUT)=0`

- virtual `telux::common::Status getAutoShutdownMode (GetAutoShutdownModeResponseCb callback)=0`
- virtual `~IThermalShutdownManager ()`

**5.15.1.8.1 Constructors and Destructors**

**5.15.1.8.1.1** virtual `telux::therm::IThermalShutdownManager::~IThermalShutdownManager ( ) [virtual]`

Destructor of [IThermalShutdownManager](#)

**5.15.1.8.2 Member Function Documentation**

**5.15.1.8.2.1** virtual `bool telux::therm::IThermalShutdownManager::isReady ( ) [pure virtual]`

Checks the status of thermal shutdown management service and if the other APIs are ready for use and returns the result.

**Returns**

True if the services are ready otherwise false.

**5.15.1.8.2.2** virtual `std::future<bool> telux::therm::IThermalShutdownManager::onReady ( ) [pure virtual]`

Wait for thermal shutdown management service to be ready.

**Returns**

A future that caller can wait on to be notified when thermal shutdown management service is ready.

**5.15.1.8.2.3** virtual `telux::common::Status telux::therm::IThermalShutdownManager::registerListener ( std::weak_ptr< IThermalShutdownListener > listener ) [pure virtual]`

Register a listener for updates on automatic shutdown mode changes

**Parameters**

<i>in</i>	<i>listener</i>	Pointer of <a href="#">IThermalShutdownListener</a> object that processes the notification
-----------	-----------------	--

**Returns**

Status of registerListener i.e success or suitable status code.

**5.15.1.8.2.4 virtual telux::common::Status telux::therm::IThermalShutdownManager::deregisterListener ( std::weak\_ptr< IThermalShutdownListener > *listener* ) [pure virtual]**

Remove a previously registered listener.

#### Parameters

in	<i>listener</i>	Previously registered <a href="#">IThermalShutdownListener</a> that needs to be removed
----	-----------------	---

#### Returns

Status of deregisterListener, success or suitable status code

**5.15.1.8.2.5 virtual telux::common::Status telux::therm::IThermalShutdownManager::setAuto↔ ShutdownMode ( AutoShutdownMode *mode*, telux::common::ResponseCallback *callback* = nullptr, uint32\_t *timeout* = DEFAULT\_TIMEOUT ) [pure virtual]**

Set automatic thermal shutdown mode. When set to DISABLE mode successfully, it remains in DISABLE mode briefly and automatically changes to ENABLE mode after notifying the clients.

#### Parameters

in	<i>mode</i>	desired AutoShutdownMode to be set
in	<i>callback</i>	Optional callback to get the response of the command
in	<i>timeout</i>	Optional timeout(in seconds) for which auto-shutdown remains disabled.

#### Returns

Status of setAutoShutdownMode i.e. success or suitable status code.

**5.15.1.8.2.6 virtual telux::common::Status telux::therm::IThermalShutdownManager::get↔ AutoShutdownMode ( GetAutoShutdownModeResponseCb *callback* ) [pure virtual]**

Get automatic thermal shutdown mode.

#### Parameters

in	<i>callback</i>	GetAutoShutdownModeResponseCb to get response of the request
----	-----------------	--

#### Returns

Status of getAutoShutdownMode i.e. success or suitable status code.

## 5.15.2 Enumeration Type Documentation

### 5.15.2.1 enum telux::therm::AutoShutdownMode [strong]

Defines the status of automatic thermal shutdown

#### Enumerator

- UNKNOWN** Automatic thermal shutdown status is unknown
- ENABLE** Automatic thermal shutdown is enabled
- DISABLE** Automatic thermal shutdown is disabled

### 5.15.2.2 enum telux::therm::TripType [strong]

Defines the type of trip points, it can be one of the values for ACPI (Advanced Configuration and Power Interface) thermal zone

#### Enumerator

- UNKNOWN** Trip type is unknown
- CRITICAL** Trip point at which system shuts down
- HOT** Trip point to notify emergency
- PASSIVE** Trip point at which kernel lowers the CPU's frequency and throttle the processor down
- ACTIVE** Trip point at which processor fan turns on
- CONFIGURABLE\_HIGH** Triggering threshold at which mitigation starts. This type is added to support legacy targets
- CONFIGURABLE\_LOW** Clearing threshold at which mitigation stops. This type is added to support legacy targets

## 5.15.3 Variable Documentation

### 5.15.3.1 const uint32\_t telux::therm::DEFAULT\_TIMEOUT = 30

Default time out (in seconds) for thermal auto-shutdown service to re-enable thermal auto-shutdown.



## 5.16 TCU Activity Manager

This section contains APIs related to TCU activity state management.

### 5.16.1 Data Structure Documentation

#### 5.16.1.1 class telux::power::PowerFactory

[PowerFactory](#) allows creation of TCU-activity manager instance.

##### Public member functions

- `std::shared_ptr< ITcuActivityManager > getTcuActivityManager (ClientType clientType=ClientType::SLAVE)`

##### Static Public Member Functions

- static `PowerFactory & getInstance ()`

#### 5.16.1.1.1 Member Function Documentation

##### 5.16.1.1.1.1 static PowerFactory& telux::power::PowerFactory::getInstance ( ) [static]

API to get the factory instance for TCU-activity management

##### 5.16.1.1.1.2 std::shared\_ptr<ITcuActivityManager> telux::power::PowerFactory::getTcuActivityManager ( ClientType clientType = ClientType::SLAVE )

API to get the TCU-activity Manager instance

##### Parameters

in	<i>type</i>	Type of the client that is going to access <a href="#">ITcuActivityManager</a> APIs <a href="#">ClientType</a>
----	-------------	--

##### Returns

Pointer of [ITcuActivityManager](#) object.

#### 5.16.1.2 class telux::power::ITcuActivityListener

Listener class for getting notifications related to TCU-activity state and also the updates related to TCU-activity service status. The client needs to implement these methods as briefly as possible and avoid blocking calls in it. The methods in this class can be invoked from multiple different threads. Client needs to make sure that the implementation is thread-safe.

**Public member functions**

- virtual void [onTcuActivityStateUpdate](#) ([TcuActivityState](#) state)
- virtual void [onSlaveAckStatusUpdate](#) ([telux::common::Status](#) status)
- virtual [~ITcuActivityListener](#) ()

**5.16.1.2.1 Constructors and Destructors**

**5.16.1.2.1.1** virtual [telux::power::ITcuActivityListener::~ITcuActivityListener](#) ( ) [[virtual](#)]

Destructor of [ITcuActivityListener](#)

**5.16.1.2.2 Member Function Documentation**

**5.16.1.2.2.1** virtual void [telux::power::ITcuActivityListener::onTcuActivityStateUpdate](#) ( [TcuActivity↔State](#) *state* ) [[virtual](#)]

This function is called when the TCU-activity state is going to change.

**Parameters**

in	<i>state</i>	TCU-activity state that system is about to enter
----	--------------	--

**5.16.1.2.2.2** virtual void [telux::power::ITcuActivityListener::onSlaveAckStatusUpdate](#) ( [telux↔::common::Status](#) *status* ) [[virtual](#)]

This function is called with the overall acknowledgement status from all the SLAVE clients, for state change triggered previously by MASTER client.

This API will be invoked only for the MASTER client. If at least one SLAVE client does not acknowledge within the configured timeout, then [Status::EXPIRED](#) would be reported.

**Parameters**

in	<i>status</i>	status of the SLAVE clients acknowledgements
----	---------------	--

**5.16.1.3 class [telux::power::ITcuActivityManager](#)**

[ITcuActivityManager](#) provides interface to register and de-register listeners (to get TCU-activity state updates). And also API to initiate TCU-activity state transition.

**Public member functions**

- virtual bool [isReady](#) ()=0
- virtual [std::future< bool >](#) [onReady](#) ()=0
- virtual [telux::common::Status](#) [registerListener](#) ([std::weak\\_ptr< ITcuActivityListener >](#) listener)=0

- virtual `telux::common::Status deregisterListener (std::weak_ptr< ITcuActivityListener > listener)=0`
- virtual `telux::common::Status registerServiceStateListener (std::weak_ptr< telux::common::IServiceStatusListener > listener)=0`
- virtual `telux::common::Status deregisterServiceStateListener (std::weak_ptr< telux::common::IServiceStatusListener > listener)=0`
- virtual `telux::common::Status setActivityState (TcuActivityState state, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `TcuActivityState getActivityState ()=0`
- virtual `telux::common::Status sendActivityStateAck (TcuActivityStateAck ack)=0`
- virtual `~ITcuActivityManager ()`

### 5.16.1.3.1 Constructors and Destructors

**5.16.1.3.1.1 virtual `telux::power::ITcuActivityManager::~~ITcuActivityManager ( ) [virtual]`**

Destructor of `ITcuActivityManager`

### 5.16.1.3.2 Member Function Documentation

**5.16.1.3.2.1 virtual `bool telux::power::ITcuActivityManager::isReady ( ) [pure virtual]`**

Checks the status of TCU-activity services and if the other APIs are ready for use, and returns the result.

#### Returns

True if the services are ready otherwise false.

**5.16.1.3.2.2 virtual `std::future<bool> telux::power::ITcuActivityManager::onReady ( ) [pure virtual]`**

Wait for TCU-activity services to be ready.

#### Returns

A future that caller can wait on to be notified when TCU-activity services are ready.

**5.16.1.3.2.3 virtual `telux::common::Status telux::power::ITcuActivityManager::registerListener ( std::weak_ptr< ITcuActivityListener > listener ) [pure virtual]`**

Register a listener for updates on TCU-activity state changes.

#### Parameters

in	<i>listener</i>	Pointer of <code>ITcuActivityListener</code> object that processes the notification
----	-----------------	---

**Returns**

Status of registerListener i.e success or suitable status code.

**5.16.1.3.2.4** `virtual telux::common::Status telux::power::ITcuActivityManager::deregisterListener ( std::weak_ptr< ITcuActivityListener > listener ) [pure virtual]`

Remove a previously registered listener.

**Parameters**

in	<i>listener</i>	Previously registered <a href="#">ITcuActivityListener</a> that needs to be removed
----	-----------------	---

**Returns**

Status of deregisterListener, success or suitable status code

**5.16.1.3.2.5** `virtual telux::common::Status telux::power::ITcuActivityManager::registerServiceStateListener ( std::weak_ptr< telux::common::IServiceStatusListener > listener ) [pure virtual]`

Register a listener for updates on TCU-activity management service status.

**Parameters**

in	<i>listener</i>	Pointer of IServiceStatusListener object that processes the notification
----	-----------------	--

**Returns**

Status of registerServiceStateListener i.e success or suitable status code.

**5.16.1.3.2.6** `virtual telux::common::Status telux::power::ITcuActivityManager::deregisterServiceStateListener ( std::weak_ptr< telux::common::IServiceStatusListener > listener ) [pure virtual]`

Remove a previously registered listener for service status updates.

**Parameters**

in	<i>listener</i>	Previously registered IServiceStatusListener that needs to be removed
----	-----------------	---

**Returns**

Status of deregisterServiceStateListener, success or suitable status code

**5.16.1.3.2.7 virtual telux::common::Status telux::power::ITcuActivityManager::setActivityState ( TcuActivityState *state*, telux::common::ResponseCallback *callback* = *nullptr* ) [pure virtual]**

Initiate a TCU-activity state transition.

This API needs to be used cautiously, as it could change the power-state of the system and may affect other processes.

This API is meant for clients that have instantiated the [ITcuActivityManager](#) instance using [ClientType::MASTER](#)

#### Parameters

in	<i>state</i>	TCU-activity state that the System is intended to enter
in	<i>callback</i>	Optional callback to get the response for the TCU-activity state transition command

#### Returns

Status of setActivityState i.e. success or suitable status code.

**5.16.1.3.2.8 virtual TcuActivityState telux::power::ITcuActivityManager::getActivityState ( ) [pure virtual]**

Get the current TCU-activity state.

#### Returns

TcuActivityState

**5.16.1.3.2.9 virtual telux::common::Status telux::power::ITcuActivityManager::sendActivityStateAck ( TcuActivityStateAck *ack* ) [pure virtual]**

API to send the acknowledgement, after processing a TCU-activity state notification. This indicates that the client is prepared for state transition. Only one acknowledgement is expected from a single client process (may have multiple listeners).

#### Parameters

in	<i>ack</i>	Acknowledgement for a TCU-activity state notification.
----	------------	--

#### Returns

Status of sendActivityStateAck i.e. success or suitable status code.

## 5.16.2 Enumeration Type Documentation

### 5.16.2.1 enum telux::power::TcuActivityState [strong]

Defines the supported TCU-activity states that the listeners will be notified about.

#### Enumerator

**UNKNOWN** To indicate that system state information is not available  
**SUSPEND** System is going to SUSPEND state  
**RESUME** System is going to RESUME state  
**SHUTDOWN** System is going to SHUTDOWN

### 5.16.2.2 enum telux::power::TcuActivityStateAck [strong]

Defines the acknowledgements to TCU-activity states. The client process sends this after processing the TcuActivityState notification, indicating that it is prepared for state transition

Acknowledgement for [TcuActivityState::RESUME](#) is not required, as the state transition has already happened.

#### Enumerator

**SUSPEND\_ACK** processed [TcuActivityState::SUSPEND](#) notification  
**SHUTDOWN\_ACK** processed [TcuActivityState::SHUTDOWN](#) notification

### 5.16.2.3 enum telux::power::ClientType [strong]

Defines the type of the client that would be using the [ITcuActivityManager](#) APIs. Client that just needs the TcuActivityState notifications needs to choose [ClientType::SLAVE](#). And the client that determines the TcuActivityState would choose [ClientType::MASTER](#). Only a Master client can set the TcuActivityState. In a system, there should be a single Master client.

The ClientType needs to be chosen while instantiating the [ITcuActivityManager](#), using the API [PowerFactory::getTcuActivityManager](#)

#### Enumerator

**SLAVE** Client is a slave and interested in state change notification  
**MASTER** Client makes the decision on when the TcuActivityState should change

## 5.17 Remote SIM Provisioning

This section contains APIs related to Remote SIM provisioning.

### 5.17.1 Data Structure Documentation

#### 5.17.1.1 class telux::tel::SimProfile

[SimProfile](#) class represents single eUICC profile on the card.

##### Public member functions

- [SimProfile](#) (int profileId, [ProfileType](#) profileType, const std::string &iccid, bool [isActive](#), const std::string &nickName, const std::string &spn, const std::string &name, [IconType](#) iconType, std::vector< uint8\_t > icon, [ProfileClass](#) profileClass, [PolicyRuleMask](#) policyRuleMask)
- int [getSlotId](#) ()
- int [getProfileId](#) ()
- [ProfileType](#) [getType](#) ()
- const std::string & [getIccid](#) ()
- bool [isActive](#) ()
- const std::string & [getNickName](#) ()
- const std::string & [getSPN](#) ()
- const std::string & [getName](#) ()
- [IconType](#) [getIconType](#) ()
- std::vector< uint8\_t > [getIcon](#) ()
- [ProfileClass](#) [getClass](#) ()
- [PolicyRuleMask](#) [getPolicyRule](#) ()
- std::string [toString](#) ()

##### 5.17.1.1.1 Constructors and Destructors

5.17.1.1.1.1 `telux::tel::SimProfile::SimProfile ( int profileId, ProfileType profileType, const std::string & iccid, bool isActive, const std::string & nickName, const std::string & spn, const std::string & name, IconType iconType, std::vector< uint8_t > icon, ProfileClass profileClass, PolicyRuleMask policyRuleMask )`

##### 5.17.1.1.2 Member Function Documentation

**5.17.1.1.2.1 int telux::tel::SimProfile::getSlotId ( )**

Get slot id associated for this profile

**Returns**

SlotId

**5.17.1.1.2.2 int telux::tel::SimProfile::getProfileId ( )**

Get profile identifier.

**Returns**

unique identifier for the profile

**5.17.1.1.2.3 ProfileType telux::tel::SimProfile::getType ( )**

Get profile Type.

**Returns**

profile type

**5.17.1.1.2.4 const std::string& telux::tel::SimProfile::getIccid ( )**

Get profile ICCID.

**Returns**

profile ICCID coded as in EF-ICCID

**5.17.1.1.2.5 bool telux::tel::SimProfile::isActive ( )**

Indicates the profile state whether active or not.

**Returns**

true if profile is Active

**5.17.1.1.2.6 const std::string& telux::tel::SimProfile::getNickName ( )**

Get profile nick name.

**Returns**

profile nick name



**5.17.1.1.2.7 const std::string& telux::tel::SimProfile::getSPN ( )**

Get profile service provider name.

**Returns**

profile service provider name.

**5.17.1.1.2.8 const std::string& telux::tel::SimProfile::getName ( )**

Get profile name.

**Returns**

profile name

**5.17.1.1.2.9 IconType telux::tel::SimProfile::getIconType ( )**

Get profile icon type.

**Returns**

profile icon type

**5.17.1.1.2.10 std::vector<uint8\_t> telux::tel::SimProfile::getIcon ( )**

Get profile icon content.

**Returns**

profile icon content

**5.17.1.1.2.11 ProfileClass telux::tel::SimProfile::getClass ( )**

Get profile class.

**Returns**

profile class

**5.17.1.1.2.12 PolicyRuleMask telux::tel::SimProfile::getPolicyRule ( )**

Get profile policy rules.

**Returns**

mask of profile policy rules

### 5.17.1.1.2.13 std::string telux::tel::SimProfile::toString ( )

Get the text related informative representation of this object.

#### Returns

String containing informative string.

### 5.17.1.2 class telux::tel::ISimProfileListener

The interface listens for profile download indication and keep track of download and install progress of profile.

The methods in the listener can be invoked from multiple threads. It is client's responsibility to make sure the implementation is thread safe.

#### Public member functions

- virtual void [onAddProfileUpdate](#) (SlotId slotId, bool userConsentRequired, [DownloadStatus](#) status, uint8\_t percentage, [DownloadErrorCause](#) cause, [PolicyRuleMask](#) mask)
- virtual [~ISimProfileListener](#) ()

#### 5.17.1.2.1 Constructors and Destructors

##### 5.17.1.2.1.1 virtual telux::tel::ISimProfileListener::~~ISimProfileListener ( ) [virtual]

Destructor of [ISimProfileListener](#)

#### 5.17.1.2.2 Member Function Documentation

##### 5.17.1.2.2.1 virtual void telux::tel::ISimProfileListener::onAddProfileUpdate ( SlotId *slotId*, bool *userConsentRequired*, [DownloadStatus](#) *status*, uint8\_t *percentage*, [DownloadErrorCause](#) *cause*, [PolicyRuleMask](#) *mask* ) [virtual]

This function is called when indication about status of profile download and installation comes.

#### Parameters

in	<i>slotId</i>	Slot on which profile get downloaded and installed.
in	<i>userConsent↔ Required</i>	User consent required or not.
in	<i>status</i>	<a href="#">ProfileDownloadStatus</a> .
in	<i>percentage</i>	Download and installation percentage.
in	<i>cause</i>	<a href="#">ProfileDownloadErrorCause</a> .
in	<i>mask</i>	<a href="#">PprMask</a> (Profile policy rules Mask)

### 5.17.1.3 class telux::tel::ISimProfileManager

[ISimProfileManager](#) is a primary interface for remote eUICCs (eSIMs or embedded SIMs) provisioning. This interface provides APIs to add, delete, set profile, update nickname, provide user consent, get Eid on the eUICC.

#### Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status addProfile](#) (SlotId slotId, const std::string &activationCode, const std::string &confirmationCode="", bool userConsentSupported=false, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status deleteProfile](#) (SlotId slotId, int profileId, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status setProfile](#) (SlotId slotId, int profileId, bool enable=false, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status updateNickName](#) (SlotId slotId, int profileId, const std::string &nickName, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status requestProfileList](#) (SlotId slotId, [ProfileListResponseCb](#) callback)=0
- virtual [telux::common::Status requestEid](#) (SlotId slotId, [EidResponseCb](#) callback)=0
- virtual [telux::common::Status provideUserConsent](#) (SlotId slotId, bool userConsent=false, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status registerListener](#) (std::weak\_ptr< [ISimProfileListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak\_ptr< [ISimProfileListener](#) > listener)=0
- virtual [~ISimProfileManager](#) ()

#### 5.17.1.3.1 Constructors and Destructors

5.17.1.3.1.1 virtual telux::tel::ISimProfileManager::~ISimProfileManager ( ) [virtual]

Destructor for [ISimProfileManager](#)

#### 5.17.1.3.2 Member Function Documentation

5.17.1.3.2.1 virtual bool telux::tel::ISimProfileManager::isSubsystemReady ( ) [pure virtual]

Checks if the eUICC subsystem is ready.

#### Returns

True if [ISimProfileManager](#) is ready for service, otherwise returns false.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.17.1.3.2.2** `virtual std::future<bool> telux::tel::ISimProfileManager::onSubsystemReady ( ) [pure virtual]`

Wait for eUICC subsystem to be ready.

**Returns**

A future that caller can wait on to be notified when card manager is ready.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.17.1.3.2.3** `virtual telux::common::Status telux::tel::ISimProfileManager::addProfile ( SlotId slotId, const std::string & activationCode, const std::string & confirmationCode = "", bool userConsentSupported = false, common::ResponseCallback callback = nullptr ) [pure virtual]`

Add new profile to eUICC card and download and install the profile on eUICC.

**Parameters**

in	<i>slotId</i>	Slot identifier corresponding to the card.
in	<i>activationCode</i>	Activation code.
in	<i>confirmationCode</i>	Optional confirmation code required for downloading the profile.
in	<i>userConsentSupported</i>	Optional User consent supported or not.
in	<i>callback</i>	Callback function to get the result of add profile.

**Returns**

Status of add profile i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.17.1.3.2.4** `virtual telux::common::Status telux::tel::ISimProfileManager::deleteProfile ( SlotId slotId, int profileId, common::ResponseCallback callback = nullptr ) [pure virtual]`

Delete profile from eUICC card.

in	<i>slotId</i>	Slot identifier corresponding to the card.
in	<i>profileId</i>	Profile identifier
in	<i>callback</i>	Callback function to get the result of delete profile.

**Returns**

Status of delete profile i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.17.1.3.2.5** `virtual telux::common::Status telux::tel::ISimProfileManager::setProfile ( SlotId slotId, int profileId, bool enable = false, common::ResponseCallback callback = nullptr ) [pure virtual]`

Enable or disable profile which allows to switch to other profile on eUICC card.

**Parameters**

in	<i>slotId</i>	Slot identifier corresponding to the card.
in	<i>profileId</i>	Profile identifier.
in	<i>enable</i>	Indicates whether a profile must be enabled or disabled. true - Enable and false - Disable.
in	<i>callback</i>	Callback function to get the result of set profile.

**Returns**

Status of set profile i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.17.1.3.2.6** `virtual telux::common::Status telux::tel::ISimProfileManager::updateNickName ( SlotId slotId, int profileId, const std::string & nickName, common::ResponseCallback callback = nullptr ) [pure virtual]`

Update nick name of the profile

**Parameters**

in	<i>slotId</i>	Slot identifier corresponding to the card.
in	<i>profileId</i>	Profile identifier
in	<i>nickName</i>	New nick name for profile.
in	<i>callback</i>	Callback function to get the result of update nickname.

**Returns**

Status of update nick name i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

**5.17.1.3.2.7 virtual telux::common::Status telux::tel::ISimProfileManager::requestProfileList ( SlotId slotId, ProfileListResponseCb callback ) [pure virtual]**

Request list of profiles supported by the eUICC card.

**Parameters**

in	<i>slotId</i>	Slot identifier corresponding to the card.
in	<i>callback</i>	Callback function to get the result of request profile list.

**Returns**

Status of request profile list i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

**5.17.1.3.2.8 virtual telux::common::Status telux::tel::ISimProfileManager::requestEid ( SlotId slotId, EidResponseCb callback ) [pure virtual]**

Request eUICC identifier(EID) for the slot.

**Parameters**

in	<i>slotId</i>	Slot identifier corresponding to the card.
in	<i>callback</i>	Callback function to get the result of request EID.

**Returns**

Status of request EID.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

**5.17.1.3.2.9** `virtual telux::common::Status telux::tel::ISimProfileManager::provideUserConsent ( SlotId slotId, bool userConsent = false, common::ResponseCallback callback = nullptr ) [pure virtual]`

Provide user consent required for downloading and installing profile.

#### Parameters

in	<i>slotId</i>	Slot identifier corresponding to the card.
in	<i>userConsent</i>	Consent for profile download and install. True means user consent to download and install.
in	<i>callback</i>	Callback function to get the result of user consent request.

#### Returns

Status of user consent request i.e. success or suitable error code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.17.1.3.2.10** `virtual telux::common::Status telux::tel::ISimProfileManager::registerListener ( std::weak_ptr< ISimProfileListener > listener ) [pure virtual]`

Register a listener to listen for status of specific events like download and installation of profile on eUICC.

#### Parameters

in	<i>listener</i>	Pointer of <a href="#">ISimProfileListener</a> object that processes the notification.
----	-----------------	--

#### Returns

Status of registerListener success or suitable status code

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.17.1.3.2.11** `virtual telux::common::Status telux::tel::ISimProfileManager::deregisterListener ( std::weak_ptr< ISimProfileListener > listener ) [pure virtual]`

De-register the listener.

**Parameters**

in	<i>listener</i>	Pointer of <a href="#">ISimProfileListener</a> object that needs to be removed
----	-----------------	--

**Returns**

Status of deregisterListener success or suitable status code

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.17.2 Enumeration Type Documentation****5.17.2.1 enum telux::tel::ProfileType [strong]**

Indicates profile type of card

**Enumerator**

**UNKNOWN**  
**REGULAR** Regular profile  
**EMERGENCY** Emergency profile

**5.17.2.2 enum telux::tel::IconType [strong]**

Indicates profile icon type.

**Enumerator**

**NONE** No icon information  
**JPEG** JPEG icon  
**PNG** PNG icon

**5.17.2.3 enum telux::tel::ProfileClass [strong]**

Indicates profile class.

**Enumerator**

**UNKNOWN** No info about profile class  
**TEST** Test profile  
**PROVISIONING** Provisioning profile  
**OPERATIONAL** Operational profile

**5.17.2.4 enum telux::tel::DownloadStatus [strong]**

Indicates profile download status.



**Enumerator**

**DOWNLOAD\_ERROR** Profile download error  
**DOWNLOAD\_IN\_PROGRESS** Profile download in progress with download percentage  
**DOWNLOAD\_COMPLETE\_INSTALLATION\_IN\_PROGRESS** Profile download is complete and installation is in progress  
**INSTALLATION\_COMPLETE** Profile installation is complete  
**USER\_CONSENT\_REQUIRED** User consent is required for proceeding with download/installation of profile

**5.17.2.5 enum telux::tel::DownloadErrorCause [strong]**

Indicates profile download error cause.

**Enumerator**

**GENERIC** Generic error  
**SIM** Error from the SIM card  
**NETWORK** Error from the network  
**MEMORY** Error due to no memory

**5.17.2.6 enum telux::tel::PolicyRuleType**

Defines profile policy rules(PPR). Each value represents corresponding bit for PprMask bitset.

**Enumerator**

**PROFILE\_DISABLE\_NOT\_ALLOWED** Disabling of the profile is not allowed  
**PROFILE\_DELETE\_NOT\_ALLOWED** Deletion of the profile is not allowed  
**PROFILE\_DELETE\_ON\_DISABLE** Deletion of the profile is required on successful disabling

## 5.18 Remote SIM

This section contains APIs related to Remote SIM operations.

### 5.18.1 Data Structure Documentation

#### 5.18.1.1 class telux::tel::IRemoteSimListener

A listener class for getting remote SIM notifications.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

##### Public member functions

- virtual void [onApuTransfer](#) (const unsigned int id, const std::vector< uint8\_t > &apdu)
- virtual void [onCardConnect](#) ()
- virtual void [onCardDisconnect](#) ()
- virtual void [onCardPowerUp](#) ()
- virtual void [onCardPowerDown](#) ()
- virtual void [onCardReset](#) ()
- virtual [~IRemoteSimListener](#) ()

##### 5.18.1.1.1 Constructors and Destructors

5.18.1.1.1.1 virtual telux::tel::IRemoteSimListener::~~IRemoteSimListener ( ) [virtual]

Destructor of [IRemoteSimListener](#)

##### 5.18.1.1.2 Member Function Documentation

5.18.1.1.2.1 virtual void telux::tel::IRemoteSimListener::onApuTransfer ( const unsigned int *id*, const std::vector< uint8\_t > & *apdu* ) [virtual]

This function is called when the modem wants to transmit a command APDU.

##### Parameters

in	<i>id</i>	Identifier for a command and response APDU pair
in	<i>apdu</i>	APDU request sent to the control point (max size = 261, per ETSI TS 102 221, section 10.1.4)

5.18.1.1.2.2 virtual void telux::tel::IRemoteSimListener::onCardConnect ( ) [virtual]

This function is called when the modem wants to establish a connection.

**5.18.1.1.2.3 virtual void telux::tel::IRemoteSimListener::onCardDisconnect ( ) [virtual]**

This function is called when the modem wants to tear down a connection.

**5.18.1.1.2.4 virtual void telux::tel::IRemoteSimListener::onCardPowerUp ( ) [virtual]**

This function is called when the modem wants to power up the card.

**5.18.1.1.2.5 virtual void telux::tel::IRemoteSimListener::onCardPowerDown ( ) [virtual]**

This function is called when the modem wants to power down the card.

**5.18.1.1.2.6 virtual void telux::tel::IRemoteSimListener::onCardReset ( ) [virtual]**

This function is called when the modem wants to warm reset the card.

**5.18.1.2 class telux::tel::IRemoteSimManager**

[IRemoteSimManager](#) provides APIs for remote SIM related operations. This allows a device to use a SIM card on another device for its WWAN modem functionality. The SIM provider service is the endpoint that interfaces with the SIM card (e.g. over bluetooth) and sends/receives data to the other endpoint, the modem. The modem sends requests to the SIM provider service to interact with the SIM card (e.g. power up, transmit APDU, etc.), and is notified of events (e.g. card errors, resets, etc.). This API is used by the SIM provider endpoint to provide a SIM card to the modem.

**Public member functions**

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status sendReset](#) ([telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status sendConnectionAvailable](#) ([telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status sendConnectionUnavailable](#) ([telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status sendCardReset](#) (const std::vector< uint8\_t > &atr, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status sendCardError](#) (const [CardErrorCause](#) cause=[CardErrorCause::INVALID](#), [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status sendCardInserted](#) (const std::vector< uint8\_t > &atr, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status sendCardRemoved](#) ([telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status sendCardWakeup](#) ([telux::common::ResponseCallback](#) callback=nullptr)=0

- virtual `telux::common::Status sendApdu` (const unsigned int id, const std::vector< uint8\_t > &apdu, const bool isSuccess=true, const unsigned int totalSize=0, const unsigned int offset=0, `telux::common::ResponseCallback` callback=nullptr)=0
- virtual `telux::common::Status registerListener` (std::weak\_ptr< `IRemoteSimListener` > listener)=0
- virtual `telux::common::Status deregisterListener` (std::weak\_ptr< `IRemoteSimListener` > listener)=0
- virtual int `getSlotId` ()=0
- virtual `~IRemoteSimManager` ()

### 5.18.1.2.1 Constructors and Destructors

5.18.1.2.1.1 virtual `telux::tel::IRemoteSimManager::~IRemoteSimManager ( ) [virtual]`

Destructor of `IRemoteSimManager`

### 5.18.1.2.2 Member Function Documentation

5.18.1.2.2.1 virtual `bool telux::tel::IRemoteSimManager::isSubsystemReady ( ) [pure virtual]`

Checks the status of remote SIM subsystem and returns the result.

#### Returns

True if remote SIM subsystem is ready for service otherwise false.

5.18.1.2.2.2 virtual `std::future<bool> telux::tel::IRemoteSimManager::onSubsystemReady ( ) [pure virtual]`

Wait for remote SIM subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when remote SIM subsystem is ready.

5.18.1.2.2.3 virtual `telux::common::Status telux::tel::IRemoteSimManager::sendReset ( telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Send reset command to the modem to reset state variables.

#### Parameters

out	<i>callback</i>	Callback function pointer to get the response of sendReset.
-----	-----------------	---

#### Returns

Status of sendReset i.e. success or suitable status code.

#### 5.18.1.2.2.4 virtual telux::common::Status telux::tel::IRemoteSimManager::sendConnectionAvailable ( telux::common::ResponseCallback *callback* = *nullptr* ) [pure virtual]

Send connection available event to the modem.

##### Parameters

out	<i>callback</i>	Callback function pointer to get the response.
-----	-----------------	--

##### Returns

Status of sendConnectionAvailable i.e. success or suitable status code.

#### 5.18.1.2.2.5 virtual telux::common::Status telux::tel::IRemoteSimManager::sendConnectionUnavailable ( telux::common::ResponseCallback *callback* = *nullptr* ) [pure virtual]

Send connection unavailable event to the modem.

##### Parameters

out	<i>callback</i>	Callback function pointer to get the response.
-----	-----------------	--

##### Returns

Status of sendConnectionUnavailable i.e. success or suitable status code.

#### 5.18.1.2.2.6 virtual telux::common::Status telux::tel::IRemoteSimManager::sendCardReset ( const std::vector< uint8\_t > & *atr*, telux::common::ResponseCallback *callback* = *nullptr* ) [pure virtual]

Send card reset event to the modem.

##### Parameters

in	<i>atr</i>	Answer to Reset bytes (max size = 32, per ISO/IEC 7816-3:2006 section 8.1).
out	<i>callback</i>	Callback function pointer to get the response of sendCardReset.

##### Returns

Status of sendCardReset i.e. success or suitable status code.

#### 5.18.1.2.2.7 virtual telux::common::Status telux::tel::IRemoteSimManager::sendCardError ( const CardErrorCause *cause* = CardErrorCause::INVALID, telux::common::ResponseCallback *callback* = *nullptr* ) [pure virtual]

Send card error event to the modem.

in	<i>cause</i>	Card Error cause.
out	<i>callback</i>	Callback function pointer to get the response of sendCardError.

**Returns**

Status of sendCardError i.e. success or suitable status code.

**5.18.1.2.2.8** `virtual telux::common::Status telux::tel::IRemoteSimManager::sendCardInserted ( const std::vector< uint8_t > & atr, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Send card inserted event to the modem.

**Parameters**

in	<i>atr</i>	Answer to Reset bytes (max size = 32, per ISO/IEC 7816-3:2006 section 8.1).
out	<i>callback</i>	Callback function pointer to get the response of sendCardInserted.

**Returns**

Status of sendCardInserted i.e. success or suitable status code.

**5.18.1.2.2.9** `virtual telux::common::Status telux::tel::IRemoteSimManager::sendCardRemoved ( telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Send card removed event to the modem.

**Parameters**

out	<i>callback</i>	Callback function pointer to get the response of sendCardRemoved.
-----	-----------------	---

**Returns**

Status of sendCardRemoved i.e. success or suitable status code.

**5.18.1.2.2.10** `virtual telux::common::Status telux::tel::IRemoteSimManager::sendCardWakeup ( telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Send card wakeup event to the modem.

**Parameters**

out	<i>callback</i>	Callback function pointer to get the response of sendCardWakeup.
-----	-----------------	--

**Returns**

Status of sendCardWakeup i.e. success or suitable status code.

**5.18.1.2.2.11** `virtual telux::common::Status telux::tel::IRemoteSimManager::sendApdu ( const unsigned int id, const std::vector< uint8_t > & apdu, const bool isSuccess = true, const unsigned int totalSize = 0, const unsigned int offset = 0, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Sends an APDU message to the modem, in response to a previous APDU sent by the modem.

**Parameters**

in	<i>id</i>	Identifier for command and response APDU pair.
in	<i>apdu</i>	Response APDU (max size = 1024).
in	<i>isSuccess</i>	Whether APDU transaction completed successfully.
in	<i>totalSize</i>	Total length of the APDU message (used when the response is larger than 1024 bytes and must be passed in multiple segments).
in	<i>offset</i>	Offset of this APDU segment in the original message.
out	<i>callback</i>	Callback function pointer to get the response of sendApdu.

**Returns**

Status of sendApdu i.e. success or suitable status code.

**5.18.1.2.2.12** `virtual telux::common::Status telux::tel::IRemoteSimManager::registerListener ( std::weak_ptr< IRemoteSimListener > listener ) [pure virtual]`

Register a listener for specific updates from the modem.

**Parameters**

in	<i>listener</i>	Pointer of <a href="#">IRemoteSimListener</a> object that processes the notification
----	-----------------	--

**Returns**

Status of registerListener i.e success or suitable status code.

**5.18.1.2.2.13** `virtual telux::common::Status telux::tel::IRemoteSimManager::deregisterListener ( std::weak_ptr< IRemoteSimListener > listener ) [pure virtual]`

Deregister the previously added listener.

**Parameters**

in	<i>listener</i>	Previously registered <a href="#">IRemoteSimListener</a> that needs to be deregistered
----	-----------------	--

**Returns**

Status of deregisterListener success or suitable status code

**5.18.1.2.2.14 virtual int telux::tel::IRemoteSimManager::getSlotId ( ) [pure virtual]**

Get associated slot ID for the RemoteSimManager

**Returns**

The slot ID associated with this [IRemoteSimManager](#)

**5.18.2 Enumeration Type Documentation****5.18.2.1 enum telux::tel::CardErrorCause [strong]**

Defines the card error cause, sent to the modem by the SIM provider

**Enumerator**

**INVALID** Card error cause value will not be passed to modem

**UNKNOWN\_ERROR** Unknown error

**NO\_LINK\_ESTABLISHED** No link was established

**COMMAND\_TIMEOUT** Command timeout

**POWER\_DOWN** Error due to a card power down



## 5.19 Modem Config

This section contains APIs related to Modem Config operations.

### 5.19.1 Data Structure Documentation

#### 5.19.1.1 class telux::config::ConfigFactory

[ConfigFactory](#) allows creation of config related classes.

##### Public member functions

- `std::shared_ptr< IModemConfigManager > getModemConfigManager ()`

##### Static Public Member Functions

- `static ConfigFactory & getInstance ()`

#### 5.19.1.1.1 Member Function Documentation

##### 5.19.1.1.1.1 static ConfigFactory& telux::config::ConfigFactory::getInstance ( ) [static]

Get instance of Config Factory

##### 5.19.1.1.1.2 std::shared\_ptr<IModemConfigManager> telux::config::ConfigFactory::getModemConfigManager ( )

Get instance of ModemConfig manager

##### Returns

pointer of [IModemConfigManager](#) object.

#### 5.19.1.2 struct telux::config::ConfigInfo

##### Data fields

Type	Field	Description
<a href="#">ConfigId</a>	id	id - stores the id of the configuration type - stores config type size - stores the size of the configuration desc - stores the configuration description version - stores version of the config file
<a href="#">ConfigType</a>	type	
uint32_t	size	
string	desc	
uint32_t	version	

### 5.19.1.3 class telux::config::IModemConfigListener

Listener class for getting notifications related to configuration change detection. The client needs to implement these methods as briefly as possible and avoid blocking calls in it. The methods in this class can be invoked from multiple different threads. Client needs to make sure that the implementation is thread-safe.

#### Public member functions

- virtual void `onConfigUpdateStatus` (`ConfigUpdateStatus` status, int slotId)
- virtual `~IModemConfigListener` ()

#### 5.19.1.3.1 Constructors and Destructors

5.19.1.3.1.1 virtual `telux::config::IModemConfigListener::~~IModemConfigListener` ( ) [`virtual`]

Destructor of `IModemConfigListener`

#### 5.19.1.3.2 Member Function Documentation

5.19.1.3.2.1 virtual void `telux::config::IModemConfigListener::onConfigUpdateStatus` ( `ConfigUpdateStatus` status, int slotId ) [`virtual`]

This function is called when a configuration update is detected. It is applicable only to SOFTWARE config.

#### Parameters

in	<i>status</i>	update status of config.
in	<i>slotId</i>	slotId where update is detected.

### 5.19.1.4 class telux::config::IModemConfigManager

`IModemConfigManager` provides interface to list config files present in modem's storage. load a new config file in modem, activate a config file, get active config file information, deactivate a config file, delete config file from the modem's storage, get and set mode of config auto selection, register and deregister listener for config update in modem. The config files are also referred to as MBNs.

#### Public member functions

- virtual bool `isSubsystemReady` ()=0
- virtual `std::future< bool >` `onSubsystemReady` ()=0
- virtual `telux::common::Status` `requestConfigList` (`ConfigListCallback` cb)=0
- virtual `telux::common::Status` `loadConfigFile` (`std::string` filePath, `ConfigType` configType, `telux::common::ResponseCallback` cb=nullptr)=0
- virtual `telux::common::Status` `activateConfig` (`ConfigType` configType, `ConfigId` configId, int slotId=DEFAULT\_SLOT\_ID, `telux::common::ResponseCallback` cb=nullptr)=0

- virtual `telux::common::Status getActiveConfig (ConfigType configType, GetActiveConfigCallback cb, int slotId=DEFAULT_SLOT_ID)=0`
- virtual `telux::common::Status deactivateConfig (ConfigType configType, int slotId=DEFAULT_SLOT_ID, telux::common::ResponseCallback cb=nullptr)=0`
- virtual `telux::common::Status deleteConfig (ConfigType configType, ConfigId configId="", telux::common::ResponseCallback cb=nullptr)=0`
- virtual `telux::common::Status getAutoSelectionMode (GetAutoSelectionModeCallback cb, int slotId=DEFAULT_SLOT_ID)=0`
- virtual `telux::common::Status setAutoSelectionMode (AutoSelectionMode mode, int slotId=DEFAULT_SLOT_ID, telux::common::ResponseCallback cb=nullptr)=0`
- virtual `telux::common::Status registerListener (std::weak_ptr< IModemConfigListener > listener)=0`
- virtual `telux::common::Status deregisterListener (std::weak_ptr< IModemConfigListener > listener)=0`

#### 5.19.1.4.1 Member Function Documentation

**5.19.1.4.1.1 virtual bool telux::config::IModemConfigManager::isSubsystemReady ( ) [pure virtual]**

Checks the status of modem config subsystem and returns the result.

##### Returns

If true that means ModemConfigManager is ready for performing config operations.

**5.19.1.4.1.2 virtual std::future<bool> telux::config::IModemConfigManager::onSubsystemReady ( ) [pure virtual]**

Wait for modem config subsystem to be ready.

##### Returns

A future that caller can wait on to be notified when modem config subsystem is ready.

**5.19.1.4.1.3 virtual telux::common::Status telux::config::IModemConfigManager::requestConfigList ( ConfigListCallback cb ) [pure virtual]**

Fetching the list of config files present in modem's storage.

##### Parameters

<i>in</i>	<i>cb</i>	- callback to the Response function.
-----------	-----------	--------------------------------------

returns SUCCESS if the request to get config list is sent successfully.

**5.19.1.4.1.4 virtual telux::common::Status telux::config::IModemConfigManager::loadConfigFile ( std::string *filePath*, ConfigType *configType*, telux::common::ResponseCallback *cb* = *nullptr* ) [pure virtual]**

Loads a new config file into the modem's storage. This is a persistent operation. Only the config files loaded into the modem's storage can be activated.

#### Parameters

in	<i>filePath</i>	- it defines the path to the config file.
in	<i>configType</i>	- type of the config file.
in	<i>cb</i>	- callback to the response function.

returns SUCCESS if the request to load config file is sent successfully.

**5.19.1.4.1.5 virtual telux::common::Status telux::config::IModemConfigManager::activateConfig ( ConfigType *configType*, ConfigId *configId*, int *slotId* = *DEFAULT\_SLOT\_ID*, telux::common::ResponseCallback *cb* = *nullptr* ) [pure virtual]**

Activates the config file on specified slot id. A file for activation must be loaded or should already be present in modem's storage.

#### Parameters

in	<i>configType</i>	- type of the config file.
in	<i>configId</i>	- id of the config file.
in	<i>slotId</i>	- it defines the slot id to be selected.
in	<i>cb</i>	- callback to the response function.

#### Returns

SUCCESS if the request to activate config file is sent successfully.

**5.19.1.4.1.6 virtual telux::common::Status telux::config::IModemConfigManager::getActiveConfig ( ConfigType *configType*, GetActiveConfigCallback *cb*, int *slotId* = *DEFAULT\_SLOT\_ID* ) [pure virtual]**

Get the currently active config file information for the specified slot id. In case default config files are activated, would return error.

#### Parameters

in	<i>configType</i>	- type of the config file.
in	<i>cb</i>	- callback to the response function.
in	<i>slotId</i>	- it defines the slot id to be selected.

#### Returns

SUCCESS if the request to get active config information is sent successfully.

**5.19.1.4.1.7** `virtual telux::common::Status telux::config::IModemConfigManager::deactivateConfig ( ConfigType configType, int slotId = DEFAULT_SLOT_ID, telux::common::Response↔ Callback cb = nullptr ) [pure virtual]`

Deactivates the config file for the specified slot id.

#### Parameters

in	<i>configType</i>	- type of the config file.
in	<i>slotId</i>	- slot id to be selected for deactivation of config.
in	<i>cb</i>	- callback to the response function.

#### Returns

SUCCESS if the request to deactivate config file is sent successfully

**5.19.1.4.1.8** `virtual telux::common::Status telux::config::IModemConfigManager::deleteConfig ( ConfigType configType, ConfigId configId = "", telux::common::ResponseCallback cb = nullptr ) [pure virtual]`

Deletes the config file from the modem's storage.

#### Parameters

in	<i>configType</i>	- type of the config file.
in	<i>configId</i>	- id of the config file. This parameter is optional if not provided all the config files of the given config type are deleted from modem's storage.
in	<i>cb</i>	- callback to the Response function.

#### Returns

SUCCESS if the request to delete config file is sent successfully

**5.19.1.4.1.9** `virtual telux::common::Status telux::config::IModemConfigManager::getAutoSelection↔ Mode ( GetAutoSelectionModeCallback cb, int slotId = DEFAULT_SLOT_ID ) [pure virtual]`

Fetching the mode of config auto selection for specified slot id.

#### Parameters

in	<i>cb</i>	- callback to the response function.
in	<i>slotId</i>	- slot id of config.

#### Returns

SUCCESS if the request to get selection mode is sent successfully

**5.19.1.4.1.10** virtual telux::common::Status telux::config::IModemConfigManager::setAutoSelectionMode ( AutoSelectionMode *mode*, int *slotId* = DEFAULT\_SLOT\_ID, telux::common::ResponseCallback *cb* = nullptr ) [pure virtual]

Setting the mode of config auto selection for specified slot id.

#### Parameters

in	<i>mode</i>	- auto selection mode status.
in	<i>slotId</i>	- slot id of the config.
in	<i>cb</i>	- callback to the response function.

#### Returns

SUCCESS if the request to set selection mode is sent successfully.

**5.19.1.4.1.11** virtual telux::common::Status telux::config::IModemConfigManager::registerListener ( std::weak\_ptr< IModemConfigListener > *listener* ) [pure virtual]

Registers the listener for indications.

#### Parameters

in	<i>listener</i>	- pointer to implemented listener.
----	-----------------	------------------------------------

#### Returns

SUCCESS if the request to register listener is sent successfully.

**5.19.1.4.1.12** virtual telux::common::Status telux::config::IModemConfigManager::deregisterListener ( std::weak\_ptr< IModemConfigListener > *listener* ) [pure virtual]

Deregisters the listener from indications.

#### Parameters

in	<i>listener</i>	- pointer to registered listener.
----	-----------------	-----------------------------------

#### Returns

SUCCESS if the request to deregister listener is sent successfully.

## 5.19.2 Enumeration Type Documentation

**5.19.2.1** enum telux::config::ConfigType [strong]

**Enumerator**

**HARDWARE** For hardware or platform related configuration files

**SOFTWARE** For software or carrier related configuration files

**5.19.2.2 enum telux::config::AutoSelectionMode [strong]**

Selection Mode defines status of auto selection mode for configs.

**Enumerator**

**DISABLED** Auto selection disabled

**ENABLED** Auto selection enabled

**5.19.2.3 enum telux::config::ConfigUpdateStatus [strong]**

ConfigUpdateStatus represent status of config update, a update of config happens when a software config is activated and all segments using the config are updated with new config.

**Enumerator**

**START** start of updation process

**COMPLETE** end of updation process

## 5.20 Wlan

- [Wlan Device Management](#)
- [Access Point Management](#)
- [Station Management](#)

This section contains APIs related to Wlan management operations.

### 5.20.1 Define Documentation

#### 5.20.1.1 #define INVALID\_AP\_ID 0

### 5.20.2 Data Structure Documentation

#### 5.20.2.1 struct telux::wlan::ApNetConfig

Ap Network Configuration

##### Data fields

Type	Field	Description
<a href="#">ApInfo</a>	info	AP type
<a href="#">ApInterworking</a>	interworking	AP network access (internet/local)

#### 5.20.2.2 struct telux::wlan::ApConfig

Ap Configuration

##### Data fields

Type	Field	Description
<a href="#">Id</a>	id	AP id
vector< <a href="#">Ap↔ NetConfig</a> >	network	Configurations supported by AP

#### 5.20.2.3 struct telux::wlan::DeviceInfo

Wlan Client Device Info

##### Data fields

Type	Field	Description
<a href="#">Id</a>	id	AP id device is connected to
string	name	User friendly string that identifies Wi-Fi device
string	ipv4Address	IPv4 Address of Wi-Fi device
string	ipv6Address	IPv6 Address of Wi-Fi device
string	macAddress	MAC Address of Wi-Fi device



### 5.20.2.4 struct telux::wlan::DeviceStats

Wlan Client Device Statistics

#### Data fields

Type	Field	Description
Id	id	AP id device is connected to
string	macAddress	MAC Address of Wi-Fi device
uint64_t	bytesTx	Number of bytes transmitted
uint64_t	bytesRx	Number of bytes received

### 5.20.2.5 class telux::wlan::IStaInterfaceManager

Manager class for configuring Wlan Station Mode.

#### Public member functions

- virtual `telux::common::ErrorCode setIpConfig (Id staId, StaIpConfig ipConfig, StaStaticIpConfig staticIpConfig)=0`
- virtual `telux::common::ErrorCode setBridgeMode (Id staId, StaBridgeMode bridgeMode)=0`
- virtual `telux::common::ErrorCode getConfig (std::vector< StaConfig > &config)=0`
- virtual `telux::common::ErrorCode getStatus (std::vector< StaStatus > &status)=0`
- virtual `telux::common::ErrorCode manageStaService (Id staId, ServiceOperation opr)=0`
- virtual `telux::common::ErrorCode registerListener (std::weak_ptr< IStaListener > listener)=0`
- virtual `telux::common::ErrorCode deregisterListener (std::weak_ptr< IStaListener > listener)=0`
- virtual `~IStaInterfaceManager ()`

### 5.20.2.6 class telux::wlan::IStaListener

#### Public member functions

- virtual void `onStationStatusChanged (std::vector< StaStatus > staStatus)`
- virtual void `onStationBandChanged (BandType radio)`
- virtual `~IStaListener ()`

### 5.20.2.7 struct telux::wlan::ApInfo

AP Info - captures ap type (private/guest)

#### Data fields

Type	Field	Description
ApType	apType	Ap type (private/guest)

### 5.20.2.8 struct telux::wlan::ApNetInfo

Ap Network Info

#### Data fields

Type	Field	Description
<a href="#">ApInfo</a>	info	Ap information (AP type)

### 5.20.2.9 struct telux::wlan::ApStatus

AP Status for enabled Networks

#### Data fields

Type	Field	Description
<a href="#">Id</a>	id	AP id
string	name	AP network interface name
string	ipv4Address	Local AP IP V4 address
string	macAddress	AP MAC address
vector< <a href="#">Ap↔ NetInfo</a> >	network	Settings for AP info

### 5.20.2.10 struct telux::wlan::StaStatus

Station Status

#### Data fields

Type	Field	Description
<a href="#">Id</a>	id	Station Id
string	name	Network interface name
string	ipv4Address	Public IP V4 address
string	ipv6Address	Public IP V6 address
string	macAddress	MAC address
<a href="#">StaInterface↔ Status</a>	status	Interface status

### 5.20.2.11 struct telux::wlan::InterfaceStatus

Wlan Interface status

#### Data fields

Type	Field	Description
<a href="#">HwDeviceType</a>	device	
vector< <a href="#">Ap↔ Status</a> >	apStatus	WiFi hardware type Vector of active APs status

Type	Field	Description
vector< <a href="#">StaStatus</a> >	staStatus	Vector of active Sta status

### 5.20.2.12 class telux::wlan::IWlanDeviceManager

WlanDeviceManager is a primary interface for configuring Wireless LAN. it provide APIs to enable, configure, activate, and modify modes.

#### Public member functions

- virtual [telux::common::ServiceStatus](#) getServiceStatus ()=0
- virtual [telux::common::ErrorCode](#) enable (bool enable)=0
- virtual [telux::common::ErrorCode](#) setMode (int numOfAp, int numOfSta)=0
- virtual [telux::common::ErrorCode](#) getConfig (int &numAp, int &numSta)=0
- virtual [telux::common::ErrorCode](#) getStatus (bool &isEnabled, std::vector< [InterfaceStatus](#) > &status)=0
- virtual [telux::common::ErrorCode](#) registerListener (std::weak\_ptr< [IWlanListener](#) > listener)=0
- virtual [telux::common::ErrorCode](#) deregisterListener (std::weak\_ptr< [IWlanListener](#) > listener)=0
- virtual [~IWlanDeviceManager](#) ()

### 5.20.2.13 class telux::wlan::IWlanListener

#### Public member functions

- virtual void onServiceStatusChange ([telux::common::ServiceStatus](#) status)
- virtual void onEnableChanged (bool enable)
- virtual [~IWlanListener](#) ()

### 5.20.2.14 class telux::wlan::WlanFactory

[WlanFactory](#) is the central factory to create all wlan classes.

#### Public member functions

- virtual std::shared\_ptr< [IWlanDeviceManager](#) > [getWlanDeviceManager](#) ([telux::common::InitResponseCb](#) clientCallback=nullptr)=0
- virtual std::shared\_ptr< [IApInterfaceManager](#) > [getApInterfaceManager](#) ()=0
- virtual std::shared\_ptr< [IStaInterfaceManager](#) > [getStaInterfaceManager](#) ()=0

#### Static Public Member Functions

- static [WlanFactory](#) & [getInstance](#) ()

## Protected Member Functions

- [WlanFactory](#) ()
- virtual [~WlanFactory](#) ()

### 5.20.2.14.1 Constructors and Destructors

5.20.2.14.1.1 `telux::wlan::WlanFactory::WlanFactory ( ) [protected]`

5.20.2.14.1.2 `virtual telux::wlan::WlanFactory::~~WlanFactory ( ) [protected], [virtual]`

### 5.20.2.14.2 Member Function Documentation

5.20.2.14.2.1 `static WlanFactory& telux::wlan::WlanFactory::getInstance ( ) [static]`

Get Wlan Factory instance.

5.20.2.14.2.2 `virtual std::shared_ptr<IWlanDeviceManager> telux::wlan::WlanFactory::getWlanDeviceManager ( telux::common::InitResponseCb clientCallback = nullptr ) [pure virtual]`

Get Wlan Device Manager

#### Parameters

<i>in</i>	<i>clientCallback</i>	Optional callback to get the initialization status of WlanDeviceManager <a href="#">telux::common::InitResponseCb</a>
-----------	-----------------------	---

#### Returns

instance of [IWlanDeviceManager](#)

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.20.2.14.2.3 `virtual std::shared_ptr<IApInterfaceManager> telux::wlan::WlanFactory::getApInterfaceManager ( ) [pure virtual]`

Get Access Point Interface Manager

#### Returns

instance of [IApInterfaceManager](#)

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.20.2.14.2.4** `virtual std::shared_ptr<IStaInterfaceManager> telux::wlan::WlanFactory::getStaInterfaceManager ( ) [pure virtual]`

Get Station Interface Manager

**Returns**

instance of [IStaInterfaceManager](#)

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.20.3 Enumeration Type Documentation****5.20.3.1 enum telux::wlan::ApInterworking [strong]**

AP Interworking Information

**Enumerator**

***INTERNET\_ACCESS*** AP with internet access only - No LAN access  
***FULL\_ACCESS*** AP Can Access LAN and Internet

**5.20.3.2 enum telux::wlan::ApDeviceConnectionEvent [strong]**

AP Client Connection Status

**Enumerator**

***CONNECTED***  
***DISCONNECTED***  
***IPV4\_UPDATED***  
***IPV6\_UPDATED***

**5.20.3.3 enum telux::wlan::BandType [strong]**

Radio Band Types:

**Enumerator**

***BAND\_5GHZ***  
***BAND\_2GHZ***

#### 5.20.3.4 enum telux::wlan::ConnectionStatus [strong]

Connection Status

##### Enumerator

**UNKNOWN** Device connection is unknown

**CONNECTED** Device is connected

**DISCONNECTED** Device is disconnected

#### 5.20.3.5 enum telux::wlan::ld [strong]

Identifiers for Ap, Sta, P2p

##### Enumerator

**PRIMARY**

**SECONDARY**

**TERTIARY**

**QUATERNARY**

#### 5.20.3.6 enum telux::wlan::ApType [strong]

AP Types:

##### Enumerator

**UNKNOWN**

**PRIVATE**

**GUEST**

#### 5.20.3.7 enum telux::wlan::StaInterfaceStatus [strong]

Station Interface Status

##### Enumerator

**UNKNOWN** Station interface is unknown

**CONNECTING** Station interface is connecting

**CONNECTED** Station interface is connected

**DISCONNECTED** Station interface is disconnected

**ASSOCIATION\_FAILED** Station is unable to associate with AP

**IP\_ASSIGNMENT\_FAILED** Station is unable to get IP address via DHCP

#### 5.20.3.8 enum telux::wlan::OperationType [strong]

This applies in architectures where the modem is attached to an External Application Processor(EAP). An API that sets or configure Wlan can be invoked from the EAP or from the modems Internal Application Processor (IAP). This type specifies where the operation should be carried out.

**Enumerator**

**WLAN\_LOCAL** Perform the operation on the processor where the API is invoked.

**WLAN\_REMOTE** Perform the operation on the application processor other than where the API is invoked.

**5.20.3.9 enum telux::wlan::IpFamilyType [strong]**

Preferred IP family for the connection

**Enumerator**

**UNKNOWN**

**IPV4** IPv4 data connection

**IPV6** IPv6 data connection

**IPV4V6** IPv4 and IPv6 data connection

**5.20.3.10 enum telux::wlan::ServiceOperation [strong]**

Service operations to be performed

**Enumerator**

**STOP** Stop service

**START** Start service

**RESTART** Restart service

**5.20.3.11 enum telux::wlan::InterfaceState [strong]**

Wlan Interface State

**Enumerator**

**INACTIVE** Interface is Inactive

**ACTIVE** Interface is Active

**5.20.3.12 enum telux::wlan::HwDeviceType [strong]**

Wlan Interface Device

**Enumerator**

**UNKNOWN** Wlan device is Unknown

**QCA6574** Wlan device is QCA6574

**QCA6696** Wlan device is QCA6696

**QCA6595** Wlan device is QCA6595

**5.20.4 Function Documentation**

#### 5.20.4.1 **virtual telux::common::ErrorCode telux::wlan::IApInterfaceManager::setConfig ( ApConfig *config* ) [pure virtual]**

Set Access Point config: Used to fully configure access points including venue type, radio type (2.4/5 GHz), private/guest network and all other related settings. Configurations will take effect after hostapd service is restarted by calling [telux::wlan::IApInterfaceManager::manageApService](#).

##### Parameters

in	<i>config</i>	AP configuration parameters <a href="#">telux::wlan::ApConfig</a>
----	---------------	---

##### Returns

operation error code (if any). [telux::common::ErrorCode](#) [telux::common::Status::NOTALLOWED](#) is returned if AP to be configured was not enabled in [telux::wlan::WlanDeviceManager::setMode](#).

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.20.4.2 **virtual telux::common::ErrorCode telux::wlan::IApInterfaceManager::getConfig ( std::vector< ApConfig > & *config* ) [pure virtual]**

Request Access Point Configurations

##### Parameters

in	<i>config</i>	Vector of AP configurations <a href="#">telux::wlan::ApConfig</a> as set by <a href="#">telux::wlan::IApInterfaceManager::setConfig</a>
----	---------------	---

##### Returns

operation error code (if any). [telux::common::ErrorCode](#)

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.20.4.3 **virtual telux::common::ErrorCode telux::wlan::IApInterfaceManager::getStatus ( std::vector< ApStatus > & *status* ) [pure virtual]**

Request AP Status

##### Parameters

in	<i>status</i>	Vector of AP network Status <a href="#">telux::wlan::ApStatus</a>
----	---------------	---



**Returns**

operation error code (if any). [telux::common::ErrorCode](#)

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.20.4.4 **virtual telux::common::ErrorCode telux::wlan::IApInterfaceManager::get← ConnectedDevices ( std::vector< DeviceInfo > & clientsInfo ) [pure virtual]**

Request Connected Devices to all enabled access points. Each entry in returned list will contain information about a device such as access point it is connected to and IP and MAC address as defined in [telux::wlan::DeviceInfo](#)

**Parameters**

in	<i>clientsInfo</i>	List of connected devices Info <a href="#">telux::wlan::DeviceInfo</a>
----	--------------------	--

**Returns**

operation error code (if any). [telux::common::ErrorCode](#)

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.20.4.5 **virtual telux::common::ErrorCode telux::wlan::IApInterfaceManager::get← ConnectedDevicesStats ( std::vector< DeviceStats > & clientsStats ) [pure virtual]**

Request statistics for all devices connected to all access points. Each entry in returned list will contains transmitted and recieved bytes for a device as defined in [telux::common::DeviceStats](#)

**Parameters**

in	<i>clientStats</i>	List of connected clients statistics <a href="#">telux::wlan::DeviceStats</a>
----	--------------------	---

**Returns**

operation error code (if any). [telux::common::ErrorCode](#)

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.20.4.6 virtual telux::common::ErrorCode telux::wlan::IApInterfaceManager↔ ::manageApService ( Id *apId*, ServiceOperation *opr* ) [pure virtual]

Execute an operation on hostapd service. Provides ability for client to either stop/start or restart hostapd service for selected access point. Restarting hostapd service is required for any changes made to hosapd.conf file and changes made by [telux::wlan::IApInterfaceManager::setConfig](#) to take effect. Stop/Start operation [telux::wlan::ServiceOperation](#) will Stop/Start WiFi service for access point. Access points selected to execute operation on, will temporarily go out of service when this API is called. This API should be called only when access point is configured through [telux::wlan::IDeviceManager::setMode](#)

##### Parameters

in	<i>apId</i>	AP identifier to execute operation on. <a href="#">telux::wlan::Id</a>
in	<i>opr</i>	Operation to be performed on hostapd <a href="#">telux::wlan::ServiceOperation</a>

##### Returns

operation error code (if any). [telux::common::ErrorCode](#)

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.20.4.7 virtual telux::common::ErrorCode telux::wlan::IApInterfaceManager↔ ::registerListener ( std::weak\_ptr< IApListener > *listener* ) [pure virtual]

Register a listener for specific events in Access Point Manager

##### Parameters

in	<i>listener</i>	pointer of <a href="#">IApListener</a> object that processes the notification
----	-----------------	---

##### Returns

operation error code (if any). [telux::common::ErrorCode](#)

#### 5.20.4.8 virtual telux::common::ErrorCode telux::wlan::IApInterfaceManager↔ ::deregisterListener ( std::weak\_ptr< IApListener > *listener* ) [pure virtual]

Removes a previously added listener.

##### Parameters

in	<i>listener</i>	pointer of <a href="#">IApListener</a> object that needs to be removed
----	-----------------	--

**Returns**

operation error code (if any). [telux::common::ErrorCode](#)

**5.20.4.9 virtual telux::wlan::IApInterfaceManager::~~IApInterfaceManager ( )**  
**[virtual]**

**5.20.4.10 virtual void telux::wlan::IApListener::onApDeviceStatusChanged (**  
**ApDeviceConnectionEvent event, std::vector< DeviceInfo > info )**  
**[virtual]**

This function is called when AP device status has changed

**Parameters**

in	<i>event</i>	Event detected on device <a href="#">telux::wlan::ApDeviceConnectionEvent</a>
in	<i>info</i>	Info about devices <a href="#">telux::wlan::DeviceInfo</a>

**5.20.4.11 virtual void telux::wlan::IApListener::onApBandChanged ( BandType radio**  
**) [virtual]**

This function is called when AP switch to different operation band

**Parameters**

in	<i>radio</i>	New AP operation band <a href="#">telux::wlan::BandType</a>
----	--------------	---

**5.20.4.12 virtual telux::wlan::IApListener::~~IApListener ( ) [virtual]**

**5.20.4.13 virtual telux::common::ServiceStatus telux::wlan::IWlanDeviceManager↔**  
**::getServiceStatus ( ) [pure virtual]**

Checks the readiness status of wlan manager and returns the result.

**Returns**

SERVICE\_AVAILABLE - If wlan manager is ready for service. SERVICE\_UNAVAILABLE - If wlan manager is temporarily unavailable. SERVICE\_FAILED - If wlan manager encountered an irrecoverable failure.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.20.4.14 virtual telux::common::ErrorCode telux::wlan::IWlanDeviceManager::enable ( bool *enable* ) [pure virtual]

Enable or Disable Wlan Service. Configurations set by [telux::wlan::IWlanDeviceManager::setMode](#) must be completed before enabling Wlan. If any of configurations need to be changed after Wlan is enabled, this API must be called with enable set to false followed by a call with enable set to true for the new configurations to take effect. Calling this API with enable, will start hostapd and wpa\_supplicant daemons. Further changes to hostapd and wpa\_supplicant will require calling [telux::wlan::IApInterfaceManager::manageApService](#) and [telux::wlan::IStaInterfaceManager::manageStaService](#) respectively. Client shall wait for [IWlanListener::onEnableChanged](#) indication to confirm WLAN was enabled/disabled successfully

##### Parameters

in	<i>enable</i>	true : Enable Wlan, false: Disable Wlan.
----	---------------	--

##### Returns

operation error code (if any). [telux::common::ErrorCode](#)

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.20.4.15 virtual telux::common::ErrorCode telux::wlan::IWlanDeviceManager::setMode ( int *numOfAp*, int *numOfSta* ) [pure virtual]

Set Wlan mode - number of supported APs, and stations. This API shall be called when wlan is disabled. On enablement, wlan will enable APs and Stations set in this API.

##### Parameters

in	<i>numOfAp</i>	Num of Access Points to be enabled. If no Access Point is enabled, this argument should be set to 0. Configuration of each AP is accomplished through <a href="#">telux::data::wlan::IApManager</a> instance requested from factory.
in	<i>numOfSta</i>	Num of Stations to be enabled. If no station is enabled, this argument should be set to 0. Configuration of each Station is accomplished through <a href="#">telux::data::wlan::IStaManager</a> instance requested from factory.

##### Returns

operation error code (if any). [telux::common::ErrorCode](#).

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.20.4.16 **virtual telux::common::ErrorCode telux::wlan::IWlanDeviceManager::get← Config ( int & numAp, int & numSta ) [pure virtual]**

Request Wlan configuration: Returns the configuration that was set using `telux::wlan::IWlanDevice←  
Manager::setMode`. This might differ from what configuration is has actually been enabled in the system, for instance, when the hardware cannot fully support the configuration that was set. To get the status of current configuration an Wlan enablement, `telux::wlan::IWlanDeviceManager::getStatus` should be used.

##### Parameters

in	<i>numAp</i>	Num of configured APs
in	<i>numSta</i>	Num of configured Stations

##### Returns

operation error code (if any). [telux::common::ErrorCode](#)

##### Note

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

#### 5.20.4.17 **virtual telux::common::ErrorCode telux::wlan::IWlanDeviceManager::get← Status ( bool & isEnabled, std::vector< InterfaceStatus > & status ) [pure virtual]**

Request Wlan status: Return Wlan enablement status and Interface status of APs and Station such as active/inactive, network interface name and hardware device they are mapped to. Results are valid only if Wlan is enabled.

##### Parameters

in	<i>isEnabled</i>	true: Wlan is enabled. false: Wlan is Disabled.
in	<i>status</i>	vector of interface status <a href="#">InterfaceStatus</a> .

##### Returns

operation error code (if any). [telux::common::ErrorCode](#)

##### Note

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

**5.20.4.18** `virtual telux::common::ErrorCode telux::wlan::IWlanDeviceManager<↔  
::registerListener ( std::weak_ptr< IWlanListener > listener ) [pure  
virtual]`

Register a listener for specific events in the Wlan Manager

in	<i>listener</i>	pointer of <a href="#">IWlanListener</a> object that processes the notification
----	-----------------	---

#### Returns

Status of registerListener success or suitable status code

**5.20.4.19** `virtual telux::common::ErrorCode telux::wlan::IWlanDeviceManager<↔  
::deregisterListener ( std::weak_ptr< IWlanListener > listener ) [pure  
virtual]`

Removes a previously added listener.

#### Parameters

in	<i>listener</i>	pointer of <a href="#">IWlanListener</a> object that needs to be removed
----	-----------------	--

#### Returns

Status of deregisterListener success or suitable status code

**5.20.4.20** `virtual telux::wlan::IWlanDeviceManager::~~IWlanDeviceManager ( )  
[virtual]`

Destructor for [IWlanDeviceManager](#)

**5.20.4.21** `virtual void telux::wlan::IWlanListener::onServiceStatusChange (   
telux::common::ServiceStatus status ) [virtual]`

This function is called when service status changes.

#### Parameters

in	<i>status</i>	- ServiceStatus
----	---------------	-----------------

**5.20.4.22** `virtual void telux::wlan::IWlanListener::onEnableChanged ( bool enable )  
[virtual]`

This function is called when Wlan enablement has changed

in	<i>enable</i>	True: Wlan is enabled, False: Wlan is disabled
----	---------------	--

### 5.20.4.23 virtual telux::wlan::IWlanListener::~~IWlanListener ( ) [virtual]

## 5.20.5 Variable Documentation

### 5.20.5.1 AplInfo telux::wlan::ApNetConfig::info

AP type

### 5.20.5.2 AplInterworking telux::wlan::ApNetConfig::interworking

AP network access (internet/local)

### 5.20.5.3 Id telux::wlan::ApConfig::id

AP id

### 5.20.5.4 std::vector<ApNetConfig> telux::wlan::ApConfig::network

Configurations supported by AP

### 5.20.5.5 Id telux::wlan::DeviceInfo::id

AP id device is connected to

### 5.20.5.6 std::string telux::wlan::DeviceInfo::name

User friendly string that identifies Wi-Fi device

### 5.20.5.7 std::string telux::wlan::DeviceInfo::ipv4Address

IPv4 Address of Wi-Fi device

### 5.20.5.8 std::string telux::wlan::DeviceInfo::ipv6Address

IPv6 Address of Wi-Fi device

### 5.20.5.9 std::string telux::wlan::DeviceInfo::macAddress

MAC Address of Wi-Fi device

### 5.20.5.10 Id telux::wlan::DeviceStats::id

AP id device is connected to

### 5.20.5.11 std::string telux::wlan::DeviceStats::macAddress

MAC Address of Wi-Fi device

**5.20.5.12 uint64\_t telux::wlan::DeviceStats::bytesTx**

Number of bytes transmitted

**5.20.5.13 uint64\_t telux::wlan::DeviceStats::bytesRx**

Number of bytes received

**5.20.5.14 HwDeviceType telux::wlan::InterfaceStatus::device****5.20.5.15 std::vector<ApStatus> telux::wlan::InterfaceStatus::apStatus**

WiFi hardware type

Vector of active APs status

**5.20.5.16 std::vector<StaStatus> telux::wlan::InterfaceStatus::staStatus**

Vector of active Sta status



## 5.21 Wlan Device Management

This section contains APIs related to device configuration management such as number of access points and stations enabled, enable/disable Wlan, etc.

## 5.22 Access Point Management

This section contains APIs related to access point configuration management such as private/guest, internet/local access, etc.

### 5.22.1 Data Structure Documentation

#### 5.22.1.1 class telux::wlan::IApInterfaceManager

Manager class for configuring Wlan Access Points.

##### Public member functions

- virtual `telux::common::ErrorCode setConfig (ApConfig config)=0`
- virtual `telux::common::ErrorCode getConfig (std::vector< ApConfig > &config)=0`
- virtual `telux::common::ErrorCode getStatus (std::vector< ApStatus > &status)=0`
- virtual `telux::common::ErrorCode getConnectedDevices (std::vector< DeviceInfo > &clientsInfo)=0`
- virtual `telux::common::ErrorCode getConnectedDevicesStats (std::vector< DeviceStats > &clientsStats)=0`
- virtual `telux::common::ErrorCode manageApService (Id apId, ServiceOperation opr)=0`
- virtual `telux::common::ErrorCode registerListener (std::weak_ptr< IApListener > listener)=0`
- virtual `telux::common::ErrorCode deregisterListener (std::weak_ptr< IApListener > listener)=0`
- virtual `~IApInterfaceManager ()`

#### 5.22.1.2 class telux::wlan::IApListener

##### Public member functions

- virtual void `onApDeviceStatusChanged (ApDeviceConnectionEvent event, std::vector< DeviceInfo > info)`
- virtual void `onApBandChanged (BandType radio)`
- virtual `~IApListener ()`

## 5.23 Station Management

This section contains APIs related to station configuration management such as static/dynamic IP, bridge/router mode, etc.

### 5.23.1 Data Structure Documentation

#### 5.23.1.1 struct telux::wlan::StaStaticIpConfig

Static IP Configuration

##### Data fields

Type	Field	Description
string	ipAddr	IPv4 address to be assigned.
string	gwIpAddr	IPv4 address of the gateway.
string	netMask	Subnet mask.
string	dnsAddr	DNS IPv4 address.

#### 5.23.1.2 struct telux::wlan::StaConfig

Station Configuration

##### Data fields

Type	Field	Description
<a href="#">Id</a>	staId	Id of station backhaul
<a href="#">StaIpConfig</a>	ipConfig	IP configuration of station backhaul
<a href="#">StaStaticIpConfig</a>	staticIpConfig	Static IP configuration if selected
<a href="#">StaBridgeMode</a>	bridgeMode	Station configuration as Router/bridge

### 5.23.2 Enumeration Type Documentation

#### 5.23.2.1 enum telux::wlan::StaIpConfig [strong]

Station Connection IP Type

##### Enumerator

**DYNAMIC\_IP** Station is configured with dynamic IP

**STATIC\_IP** Station is configured with Static IP

#### 5.23.2.2 enum telux::wlan::StaBridgeMode [strong]

Bridge/Router Mode

##### Enumerator

**ROUTER** Station is in Router Mode

**BRIDGE** Station is in Bridge Mode

### 5.23.3 Function Documentation

#### 5.23.3.1 virtual telux::common::ErrorCode telux::wlan::IStaInterfaceManager::setIp↔ Config ( Id *stald*, StaIpConfig *ipConfig*, StaStaticIpConfig *staticIpConfig* ) [pure virtual]

Set Station IP Configurations: Set Station IP configuration dynamic/static and static IP address if selected. If API is called when WLAN is disabled, changes will take effect when WLAN is enabled using [telux::wlan::IWlanDeviceManager::enable](#) API. If API is called when WLAN is enabled, changes will take effect after restarting wpa\_supplicant by calling [telux::wlan::IStaInterfaceManager::manageStaService](#)

##### Parameters

in	<i>stald</i>	Station Identifier <a href="#">telux::wlan::Id</a>
in	<i>ipConfig</i>	Static/Dynamic IP configuration <a href="#">telux::wlan::StaIpConfig</a> .
in	<i>staticIpConfig</i>	Static IP configuration, not used if station was configured to use dynamic IP.

##### Returns

operation error code (if any). [telux::common::ErrorCode](#).

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.23.3.2 virtual telux::common::ErrorCode telux::wlan::IStaInterfaceManager↔ ::setBridgeMode ( Id *stald*, StaBridgeMode *bridgeMode* ) [pure virtual]

Set Station backhaul to act as router or bridge: Sets Station to act as router or bridge where station internal clients get public IP addresses. If API is called when WLAN is disabled, changes will take effect when WLAN is enabled using [telux::wlan::IWlanDeviceManager::enable](#) API. If API is called when WLAN is enabled, changes will take effect after restarting wpa\_supplicant by calling [telux::wlan::IStaInterfaceManager::manageStaService](#)

##### Parameters

in	<i>stald</i>	Station Identifier <a href="#">telux::wlan::Id</a>
in	<i>bridgeMode</i>	bridgeMode <a href="#">telux::wlan::StaBridgeMode</a>

##### Returns

operation error code (if any). [telux::common::ErrorCode](#).

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.23.3.3 **virtual telux::common::ErrorCode telux::wlan::IStaInterfaceManager::get↔ Config ( std::vector< StaConfig > & config ) [pure virtual]**

Request current station configurations: Returns configurations set by [telux::wlan::IStaInterfaceManager::setIpConfig](#) and [telux::wlan::IStaInterfaceManager::setBridgeMode](#)

**Parameters**

in	<i>config</i>	Station configurations <a href="#">telux::wlan::StaConfig</a>
----	---------------	---

**Returns**

operation error code (if any). [telux::common::ErrorCode](#).

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.23.3.4 **virtual telux::common::ErrorCode telux::wlan::IStaInterfaceManager::get↔ Status ( std::vector< StaStatus > & status ) [pure virtual]**

Request current station status: Returns current Sta interface status such as network interface name and IP address.

**Parameters**

in	<i>status</i>	Station Status <a href="#">telux::wlan::StaStatus</a>
----	---------------	---

**Returns**

operation error code (if any). [telux::common::ErrorCode](#).

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.23.3.5 **virtual telux::common::ErrorCode telux::wlan::IStaInterfaceManager↔ ::manageStaService ( Id stald, ServiceOperation opr ) [pure virtual]**

Execute an operation on wpa\_supplicant service. Provides ability for client to either stop/start or restart wpa\_supplicant service for selected station. Restarting wpa\_supplicant service is required for any changes made to wpa\_supplicant.conf file to take effect. Station selected to execute operation on, will temporarily go out of service when this API is called. This API should be called only when station mode is configured through [telux::wlan::IDeviceManager::setMode](#)

in	<i>staId</i>	Station identifier to execute operation on. <a href="#">telux::wlan::Id</a>
in	<i>opr</i>	Operation to be performed on wpa_supplicant <a href="#">telux::wlan::ServiceOperation</a>

**Returns**

operation error code (if any). [telux::common::ErrorCode](#).

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.23.3.6 virtual [telux::common::ErrorCode](#) [telux::wlan::IStaInterfaceManager](#)↔ ::registerListener ( [std::weak\\_ptr](#)< [IStaListener](#) > *listener* ) [pure virtual]

Register as a listener for specific events defined in [telux::wlan::IStaListener](#)

**Parameters**

in	<i>listener</i>	pointer of <a href="#">IStaListener</a> object that processes the notification
----	-----------------	--

**Returns**

operation error code (if any). [telux::common::ErrorCode](#).

### 5.23.3.7 virtual [telux::common::ErrorCode](#) [telux::wlan::IStaInterfaceManager](#)↔ ::deregisterListener ( [std::weak\\_ptr](#)< [IStaListener](#) > *listener* ) [pure virtual]

Removes a previously added listener.

**Parameters**

in	<i>listener</i>	pointer of <a href="#">IStaListener</a> object that needs to be removed
----	-----------------	---

**Returns**

operation error code (if any). [telux::common::ErrorCode](#).

### 5.23.3.8 virtual [telux::wlan::IStaInterfaceManager](#)::~[IStaInterfaceManager](#) ( ) [virtual]

### 5.23.3.9 virtual void telux::wlan::IStaListener::onStationStatusChanged ( std::vector< StaStatus > *staStatus* ) [virtual]

This function is called when Station Status Changes

in	<i>status</i>	List of station state <a href="#">telux::wlan::StaStatus</a>
----	---------------	--

### 5.23.3.10 virtual void telux::wlan::IStaListener::onStationBandChanged ( BandType *radio* ) [virtual]

This function is called when Station switch to different operation band

#### Parameters

in	<i>radio</i>	New Station operation band <a href="#">telux::wlan::BandType</a>
----	--------------	--

### 5.23.3.11 virtual telux::wlan::IStaListener::~~IStaListener ( ) [virtual]

## 5.23.4 Variable Documentation

### 5.23.4.1 std::string telux::wlan::StaStaticIpConfig::ipAddr

IPv4 address to be assigned.

### 5.23.4.2 std::string telux::wlan::StaStaticIpConfig::gwIpAddr

IPv4 address of the gateway.

### 5.23.4.3 std::string telux::wlan::StaStaticIpConfig::netMask

Subnet mask.

### 5.23.4.4 std::string telux::wlan::StaStaticIpConfig::dnsAddr

DNS IPv4 address.

### 5.23.4.5 Id telux::wlan::StaConfig::stald

Id of station backhaul

### 5.23.4.6 StalpConfig telux::wlan::StaConfig::ipConfig

IP configuration of station backhaul

#### **5.23.4.7 StaStaticIpConfig telux::wlan::StaConfig::staticIpConfig**

Static IP configuration if selected

#### **5.23.4.8 StaBridgeMode telux::wlan::StaConfig::bridgeMode**

Station configuration as Router/bridge



## 5.24 Telematics\_cv2x\_cpp

•

### 5.24.1 Data Structure Documentation

#### 5.24.1.1 class telux::cv2x::ICv2xThrottleManagerListener

Listener class for getting filter rate update notification.

##### Public member functions

- virtual void [onFilterRateAdjustment](#) (int rate)
- virtual void [onServiceStatusChange](#) (telux::common::ServiceStatus status)
- virtual [~ICv2xThrottleManagerListener](#) ()

##### 5.24.1.1.1 Constructors and Destructors

5.24.1.1.1 virtual telux::cv2x::ICv2xThrottleManagerListener::~~ICv2xThrottleManagerListener ( )  
[virtual]

Destructor of [ICv2xThrottleManagerListener](#)

##### 5.24.1.1.2 Member Function Documentation

5.24.1.1.2.1 virtual void telux::cv2x::ICv2xThrottleManagerListener::onFilterRateAdjustment ( int rate )  
[virtual]

This API is invoked to advise the client to adjust the incoming message filtering rate by `rate` messages/second. If the `rate` is positive, it indicates the client to filter `rate` more messages/second. If the `rate` is negative, it indicates the client to filter `rate` less messages/second.

##### Parameters

in	<i>rate</i>	the reported filter rate adjustment value.
----	-------------	--

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.24.1.1.2.2 virtual void telux::cv2x::ICv2xThrottleManagerListener::onServiceStatusChange ( telux::common::ServiceStatus status ) [virtual]

This API is invoked when the service status changes for example when a subsystem restart (SSR) occurs

in	status	- ServiceStatus
----	--------	-----------------

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.24.1.2 class telux::cv2x::ICv2xThrottleManager**

ThrottleManager provides throttle manager client interface.

ThrottleManager provides APIs that allows applications to specify the incoming verification load on the system. This is used to make decisions on how to optimally use the resources available in the system. The API also provides feedback to clients on the suggested filtering that needs to be done when the incoming message verification rate exceeds the instantaneous system capacity.

**Public member functions**

- virtual [telux::common::ServiceStatus getServiceStatus \(\)](#)=0
- virtual [std::future< bool > onSubsystemReady \(\)](#)=0
- virtual [telux::common::Status registerListener \(std::weak\\_ptr< ICv2xThrottleManagerListener > listener\)](#)=0
- virtual [telux::common::Status deregisterListener \(std::weak\\_ptr< ICv2xThrottleManagerListener > listener\)](#)=0
- virtual [telux::common::Status setVerificationLoad \(int load, setVerificationLoadCallback cb\)](#)=0
- virtual [~ICv2xThrottleManager \(\)](#)

**5.24.1.2.1 Constructors and Destructors**

**5.24.1.2.1.1** virtual [telux::cv2x::ICv2xThrottleManager::~~ICv2xThrottleManager \( \)](#) [virtual]

**5.24.1.2.2 Member Function Documentation**

**5.24.1.2.2.1** virtual [telux::common::ServiceStatus telux::cv2x::ICv2xThrottleManager::getServiceStatus \( \)](#) [pure virtual]

This status indicates whether the object is in a usable state.

**Returns**

SERVICE\_AVAILABLE - If location manager is ready for service. SERVICE\_UNAVAILABLE - If location manager is temporarily unavailable. SERVICE\_FAILED - If location manager encountered an irrecoverable failure.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.24.1.2.2.2** `virtual std::future<bool> telux::cv2x::ICv2xThrottleManager::onSubsystemReady ( )`  
**[pure virtual]**

Wait for Cv2x throttle manager client to be ready.

#### Returns

A future that caller can wait on to be notified when Cv2x throttle manager client become ready.

**5.24.1.2.2.3** `virtual telux::common::Status telux::cv2x::ICv2xThrottleManager::registerListener (`  
`std::weak_ptr< ICv2xThrottleManagerListener > listener ) [pure virtual]`

Registers a listener to receive the updated filer rate adjustment data.

#### Parameters

in	<i>listener</i>	- Listener that implement <a href="#">ICv2xThrottleManagerListener</a> interface.
----	-----------------	---

**5.24.1.2.2.4** `virtual telux::common::Status telux::cv2x::ICv2xThrottleManager::deregisterListener (`  
`std::weak_ptr< ICv2xThrottleManagerListener > listener ) [pure virtual]`

Deregister a [ICv2xThrottleManagerListener](#).

#### Parameters

in	<i>listener</i>	- Previously registered Cv2xThrottleManagerListener that is deregistered.
----	-----------------	---

**5.24.1.2.2.5** `virtual telux::common::Status telux::cv2x::ICv2xThrottleManager::setVerificationLoad ( int`  
`load, setVerificationLoadCallback cb ) [pure virtual]`

Set current measured/average verification load.

#### Parameters

in	<i>load</i>	- current measured verification load(verification/second).
----	-------------	--

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.24.1.3 class telux::cv2x::ICv2xTxStatusReportListener

Listeners for CV2X Tx status report must implement this interface.

**Public member functions**

- virtual void `onTxStatusReport` (const `TxStatusReport` &info)
- virtual `~ICv2xTxStatusReportListener` ()

**5.24.1.3.1 Constructors and Destructors**

**5.24.1.3.1.1** virtual `telux::cv2x::ICv2xTxStatusReportListener::~~ICv2xTxStatusReportListener` ( )  
[virtual]

Destructor for `ICv2xTxStatusReportListener`

**5.24.1.3.2 Member Function Documentation**

**5.24.1.3.2.1** virtual void `telux::cv2x::ICv2xTxStatusReportListener::onTxStatusReport` ( const `TxStatusReport` & *info* ) [virtual]

Called when a CV2X transport block is transmitted in low layer if CV2X Tx status report has been enabled by calling `setTxStatusReport`.

**Parameters**

in	<i>info</i>	- Tx status of the transport block.
----	-------------	-------------------------------------

## 5.25 Telematics\_net

### 5.25.1 Data Structure Documentation

#### 5.25.1.1 class telux::data::net::IBridgeManager

[IBridgeManager](#) provides APIs to enable/disable and set/get/delete software bridges for various WLAN and Ethernet interfaces. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready.

##### Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status enableBridge](#) (bool enable, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status addBridge](#) ([BridgeInfo](#) config, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status requestBridgeInfo](#) ([BridgeInfoResponseCb](#) callback)=0
- virtual [telux::common::Status removeBridge](#) (std::string ifaceName, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status registerListener](#) (std::weak\_ptr< [IBridgeListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak\_ptr< [IBridgeListener](#) > listener)=0
- virtual [~IBridgeManager](#) ()

#### 5.25.1.1.1 Constructors and Destructors

5.25.1.1.1.1 virtual [telux::data::net::IBridgeManager::~~IBridgeManager](#) ( ) [**virtual**]

Destructor for [IBridgeManager](#)

#### 5.25.1.1.2 Member Function Documentation

5.25.1.1.2.1 virtual bool [telux::data::net::IBridgeManager::isSubsystemReady](#) ( ) [**pure virtual**]

Checks if the data subsystem is ready.

##### Returns

True if the Bridge Manager is ready for service, otherwise returns false.

**5.25.1.1.2.2** `virtual std::future<bool> telux::data::net::IBridgeManager::onSubsystemReady ( )`  
**[pure virtual]**

Wait for data subsystem to be ready.

#### Returns

A future that caller can wait until the Bridge Manager succeed/fail to be ready.

**5.25.1.1.2.3** `virtual telux::common::Status telux::data::net::IBridgeManager::enableBridge ( bool`  
***enable*, telux::common::ResponseCallback *callback = nullptr* ) [pure virtual]**

Enable/Disable the software bridge in the system. It will affect all the configured software bridges for various interfaces.

#### Parameters

in	<i>enable</i>	TRUE to enable, FALSE to disable the bridge
in	<i>callback</i>	Optional callback to get the response for enableBridge

#### Returns

Status of enableBridge request i.e. success or suitable status code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.25.1.1.2.4** `virtual telux::common::Status telux::data::net::IBridgeManager::addBridge ( BridgeInfo`  
***config*, telux::common::ResponseCallback *callback = nullptr* ) [pure virtual]**

Add software bridge configuration for an interface.

#### Parameters

in	<i>config</i>	configuration for an interface
in	<i>callback</i>	Optional callback to get the response for addBridge

#### Returns

Status of addBridge request i.e. success or suitable status code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.25.1.1.2.5 `virtual telux::common::Status telux::data::net::IBridgeManager::requestBridgeInfo ( BridgeInfoResponseCb callback ) [pure virtual]`

Request information about all the software bridge configurations in the system

##### Parameters

in	<i>callback</i>	Response callback with list of bridge configurations
----	-----------------	--

##### Returns

Status of requestBridgeInfo request i.e. success or suitable status code.

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.25.1.1.2.6 `virtual telux::common::Status telux::data::net::IBridgeManager::removeBridge ( std::string ifaceName, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Delete a software bridge configuration for an interface.

##### Parameters

in	<i>ifaceName</i>	Name of the interface whose configuration needs to be deleted
in	<i>callback</i>	Optional callback to get the response for removeBridge

##### Returns

Status of removeBridge request i.e. success or suitable status code.

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.25.1.1.2.7 `virtual telux::common::Status telux::data::net::IBridgeManager::registerListener ( std::weak_ptr< IBridgeListener > listener ) [pure virtual]`

Register Bridge Manager as listener for Data Service health events like data service available or data service not available.

##### Parameters

in	<i>listener</i>	pointer of <a href="#">IBridgeListener</a> object that processes the notification
----	-----------------	---

**Returns**

Status of registerListener success or suitable status code

**5.25.1.1.2.8** virtual telux::common::Status telux::data::net::IBridgeManager::deregisterListener ( std::weak\_ptr< IBridgeListener > *listener* ) [pure virtual]

Removes a previously added listener.

**Parameters**

in	<i>listener</i>	pointer of <a href="#">IBridgeListener</a> object that needs to be removed
----	-----------------	--

**Returns**

Status of deregisterListener success or suitable status code

**5.25.1.2 class telux::data::net::IBridgeListener**

Interface for Bridge listener object. Client needs to implement this interface to get access to Bridge services notifications like onServiceStatusChange.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

**Public member functions**

- virtual void [onServiceStatusChange](#) (telux::common::ServiceStatus status)
- virtual [~IBridgeListener](#) ()

**5.25.1.2.1 Constructors and Destructors**

**5.25.1.2.1.1** virtual telux::data::net::IBridgeListener::~~IBridgeListener ( ) [virtual]

Destructor for [IBridgeListener](#)

**5.25.1.2.2 Member Function Documentation**

**5.25.1.2.2.1** virtual void telux::data::net::IBridgeListener::onServiceStatusChange ( telux::common::ServiceStatus *status* ) [virtual]

This function is called when service status changes.

**Parameters**

in	<i>status</i>	- <a href="#">ServiceStatus</a>
----	---------------	---------------------------------



### 5.25.1.3 class telux::data::net::IFirewallManager

IFirewallManager is a primary interface that filters and controls the network traffic on a pre-configured set of rules. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready.

#### Public member functions

- virtual bool `isSubsystemReady ()=0`
- virtual `std::future< bool > onSubsystemReady ()=0`
- virtual `telux::common::Status setFirewallConfig (FirewallConfig fwConfig, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status requestFirewallConfig (BackhaulInfo bhInfo, FirewallConfigCb callback)=0`
- virtual `telux::common::Status addFirewallEntry (FirewallEntryInfo entry, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status requestFirewallEntries (BackhaulInfo bhInfo, FirewallEntryInfoCb callback)=0`
- virtual `telux::common::Status removeFirewallEntry (BackhaulInfo bhInfo, uint32_t handle, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status enableDmz (DmzConfig config, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status disableDmz (BackhaulInfo bhInfo, const IpFamilyType ipType, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status requestDmzEntry (BackhaulInfo bhInfo, DmzEntryInfoCb callback)=0`
- virtual `telux::common::Status registerListener (std::weak_ptr< IFirewallListener > listener)=0`
- virtual `telux::common::Status deregisterListener (std::weak_ptr< IFirewallListener > listener)=0`
- virtual `telux::data::OperationType getOperationType ()=0`
- virtual `telux::common::Status setFirewall (int profileId, bool enable, bool allowPackets, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status requestFirewallStatus (int profileId, FirewallStatusCb callback)=0`
- virtual `telux::common::Status addFirewallEntry (int profileId, std::shared_ptr< IFirewallEntry > entry, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status requestFirewallEntries (int profileId, FirewallEntriesCb callback)=0`
- virtual `telux::common::Status removeFirewallEntry (int profileId, uint32_t handle, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status enableDmz (int profileId, const std::string ipAddr, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status disableDmz (int profileId, const telux::data::IpFamilyType ipType, telux::common::ResponseCallback callback=nullptr)=0`

- virtual `telux::common::Status requestDmzEntry` (int profileId, `DmzEntriesCb` dmzCb)=0
- virtual `~IFirewallManager` ()

### 5.25.1.3.1 Constructors and Destructors

**5.25.1.3.1.1** virtual `telux::data::net::IFirewallManager::~IFirewallManager` ( ) [virtual]

Destructor for `IFirewallManager`

### 5.25.1.3.2 Member Function Documentation

**5.25.1.3.2.1** virtual `bool telux::data::net::IFirewallManager::isSubsystemReady` ( ) [pure virtual]

Checks if the data subsystem is ready.

#### Returns

True if Firewall Manager is ready for service, otherwise returns false.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.25.1.3.2.2** virtual `std::future<bool> telux::data::net::IFirewallManager::onSubsystemReady` ( ) [pure virtual]

Wait for data subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when firewall manager is ready.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.25.1.3.2.3** virtual `telux::common::Status telux::data::net::IFirewallManager::setFirewallConfig` ( `FirewallConfig fwConfig`, `telux::common::ResponseCallback callback = nullptr` ) [pure virtual]

Sets firewall configuration to enable or disable firewall and update configuration to drop or accept the packets matching the rules on slot ID, profile ID and backhaul type.

On platforms with Access control enabled, Caller needs to have `TELUX_DATA_NETWORK_CONFIG` permission to invoke this API successfully.

in	<i>config</i>	Firewall configuration telux::data::FirewallConfig.
----	---------------	---

**Returns**

Status of setFirewallConfig i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.25.1.3.2.4 virtual telux::common::Status telux::data::net::IFirewallManager::requestFirewallConfig ( BackhaulInfo *bhInfo*, FirewallConfigCb *callback* ) [pure virtual]

Request status of firewall settings on specific backhaul

**Parameters**

in	<i>bhInfo</i>	Backhaul Information to request firewall status for.
in	<i>callback</i>	callback to get the response of requestFirewallConfig

**Returns**

Status of requestFirewallConfig i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.25.1.3.2.5 virtual telux::common::Status telux::data::net::IFirewallManager::addFirewallEntry ( FirewallEntryInfo *entry*, telux::common::ResponseCallback *callback = nullptr* ) [pure virtual]

Adds the firewall rule to specific backhaul

On platforms with Access control enabled, Caller needs to have TELUX\_DATA\_NETWORK\_CONFIG permission to invoke this API successfully.

**Parameters**

in	<i>entries</i>	Firewall rules entries settings.
in	<i>callback</i>	optional callback to get the response addFirewallEntry

**Returns**

Status of addFirewallEntry i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.25.1.3.2.6 virtual telux::common::Status telux::data::net::IFirewallManager::requestFirewallEntries ( BackhaulInfo *bhInfo*, FirewallEntryInfoCb *callback* ) [pure virtual]

Request Firewall rules for specific backhaul

**Parameters**

in	<i>bhInfo</i>	Backhaul Information to request firewall entries for.
in	<i>callback</i>	callback to get the response requestFirewallEntries.

**Returns**

Status of requestFirewallEntries i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.25.1.3.2.7 virtual telux::common::Status telux::data::net::IFirewallManager::removeFirewallEntry ( BackhaulInfo *bhInfo*, uint32\_t *handle*, telux::common::ResponseCallback *callback* = *nullptr* ) [pure virtual]

Remove firewall entry set on particular backhaul

**Parameters**

in	<i>bhInfo</i>	Backhaul information to remove firewall entries from.
in	<i>handle</i>	handle of Firewall entry to be removed. To retrieve the handle, first use <a href="#">requestFirewallEntries()</a> to get the list of entries added in the system. And then use <a href="#">IFirewallEntry::getHandle()</a>
in	<i>callback</i>	callback to get the response removeFirewallEntry

**Returns**

Status of removeFirewallEntry i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.25.1.3.2.8 `virtual telux::common::Status telux::data::net::IFirewallManager::enableDmz ( DmzConfig config, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Enable demilitarized zone (DMZ) on particular backhaul

#### Parameters

in	<i>config</i>	DMZ configuration to be enabled
in	<i>callback</i>	optional callback to get the response addDmz

#### Returns

Status of enableDmz i.e. success or suitable status code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.25.1.3.2.9 `virtual telux::common::Status telux::data::net::IFirewallManager::disableDmz ( BackhaulInfo bhInfo, const IpFamilyType ipType, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Disable demilitarized zone (DMZ) on particular backhaul

#### Parameters

in	<i>bhInfo</i>	Backhaul on which DMZ will be disabled.
in	<i>ipType</i>	Specify IP type of the DMZ to be disabled
in	<i>callback</i>	optional callback to get the response removeDmz

#### Returns

Status of disableDmz i.e. success or suitable status code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.25.1.3.2.10 `virtual telux::common::Status telux::data::net::IFirewallManager::requestDmzEntry ( BackhaulInfo bhInfo, DmzEntryInfoCb callback ) [pure virtual]`

Request DMZ entry on particular backhaul that was previously set using enableDmz API

#### Parameters

in	<i>bhInfo</i>	Backhaul info on which DMZ entries are requested.
in	<i>callback</i>	callback to get the response requestDmzEntry

**Returns**

Status of requestDmzEntry i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.25.1.3.2.11 `virtual telux::common::Status telux::data::net::IFirewallManager::registerListener ( std::weak_ptr< IFirewallListener > listener ) [pure virtual]`

Register Firewall Manager as listener for Data Service health events like data service available or data service not available.

**Parameters**

in	<i>listener</i>	pointer of <a href="#">IFirewallListener</a> object that processes the notification
----	-----------------	---

**Returns**

Status of registerListener success or suitable status code

### 5.25.1.3.2.12 `virtual telux::common::Status telux::data::net::IFirewallManager::deregisterListener ( std::weak_ptr< IFirewallListener > listener ) [pure virtual]`

Removes a previously added listener.

**Parameters**

in	<i>listener</i>	pointer of <a href="#">IFirewallListener</a> object that needs to be removed
----	-----------------	--

**Returns**

Status of deregisterListener success or suitable status code

### 5.25.1.3.2.13 `virtual telux::data::OperationType telux::data::net::IFirewallManager::getOperationType ( ) [pure virtual]`

Get the associated operation type for this instance.

**Returns**

OperationType of getOperationType i.e. LOCAL or REMOTE.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.25.1.3.2.14** `virtual telux::common::Status telux::data::net::IFirewallManager::setFirewall ( int profileId, bool enable, bool allowPackets, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Sets firewall configuration to enable or disable and update configuration to drop or accept the packets matching the rules.

#### Parameters

in	<i>profileId</i>	Profile identifier corresponding to the WWAN network interface. Firewall will be configured on this WWAN network interface.
in	<i>enable</i>	Indicates whether the firewall is enabled
in	<i>allowPackets</i>	Indicates whether to accept or drop packets matching the rules
in	<i>callback</i>	optional callback to get the response setFirewall

#### Returns

Status of setFirewall i.e. success or suitable status code.

**Deprecated** Use telux::data::setFirewallConfig API to set firewall on any backhaul

**5.25.1.3.2.15** `virtual telux::common::Status telux::data::net::IFirewallManager::requestFirewallStatus ( int profileId, FirewallStatusCb callback ) [pure virtual]`

Request status of firewall

#### Parameters

in	<i>profileId</i>	Profile identifier for which firewall status is requested.
in	<i>callback</i>	callback to get the response of requestFirewallStatus

#### Returns

Status of requestFirewallStatus i.e. success or suitable status code.

**Deprecated** Use telux::data::requestFirewallConfig API to request firewall status on any backhaul

**5.25.1.3.2.16** `virtual telux::common::Status telux::data::net::IFirewallManager::addFirewallEntry ( int profileId, std::shared_ptr< IFirewallEntry > entry, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Adds the firewall rule

#### Parameters

in	<i>profileId</i>	Profile identifier on which firewall rule will be added.
in	<i>entry</i>	Firewall entry based on protocol type

in	<i>callback</i>	optional callback to get the response addFirewallEntry
----	-----------------	--

**Returns**

Status of addFirewallEntry i.e. success or suitable status code.

**Deprecated** Use telux::data::addFirewallEntry API to add firewall rule on any backhaul

**5.25.1.3.2.17** virtual telux::common::Status telux::data::net::IFirewallManager::requestFirewallEntries ( int *profileId*, FirewallEntriesCb *callback* ) [pure virtual]

Request Firewall rules

**Parameters**

in	<i>profileId</i>	Profile identifier on which firewall entries are retrieved.
in	<i>callback</i>	callback to get the response requestFirewallEntries.

**Returns**

Status of requestFirewallEntries i.e. success or suitable status code.

**Deprecated** Use telux::data::requestFirewallEntries API to request firewall rules on any backhaul

**5.25.1.3.2.18** virtual telux::common::Status telux::data::net::IFirewallManager::removeFirewallEntry ( int *profileId*, uint32\_t *handle*, telux::common::ResponseCallback *callback* = nullptr ) [pure virtual]

Remove firewall entry

**Parameters**

in	<i>profileId</i>	Profile identifier on which firewall entry will be removed.
in	<i>handle</i>	handle of Firewall entry to be removed. To retrieve the handle, first use <a href="#">requestFirewallEntries()</a> to get the list of entries added in the system. And then use <a href="#">IFirewallEntry::getHandle()</a>
in	<i>callback</i>	callback to get the response removeFirewallEntry

**Returns**

Status of removeFirewallEntry i.e. success or suitable status code.

**Deprecated** Use telux::data::removeFirewallEntry API to remove firewall rule from any backhaul



**5.25.1.3.2.19** `virtual telux::common::Status telux::data::net::IFirewallManager::enableDmz ( int profileId, const std::string ipAddr, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Enable demilitarized zone (DMZ)

#### Parameters

in	<i>profileId</i>	Profile identifier on which DMZ will be enabled.
in	<i>ipAddr</i>	IP address for which DMZ will be enabled
in	<i>callback</i>	optional callback to get the response addDmz

#### Returns

Status of enableDmz i.e. success or suitable status code.

**Deprecated** Use telux::data::enableDmz API to enable DMZ on any backhaul

**5.25.1.3.2.20** `virtual telux::common::Status telux::data::net::IFirewallManager::disableDmz ( int profileId, const telux::data::IpFamilyType ipType, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Disable demilitarized zone (DMZ)

#### Parameters

in	<i>profileId</i>	Profile identifier on which DMZ will be disabled.
in	<i>ipType</i>	Specify IP type of the DMZ to be disabled
in	<i>callback</i>	optional callback to get the response removeDmz

#### Returns

Status of disableDmz i.e. success or suitable status code.

**Deprecated** Use telux::data::disableDmz API to Disable DMZ on any backhaul

**5.25.1.3.2.21** `virtual telux::common::Status telux::data::net::IFirewallManager::requestDmzEntry ( int profileId, DmzEntriesCb dmzCb ) [pure virtual]`

Request DMZ entry that was previously set using enableDmz API

#### Parameters

in	<i>profileId</i>	Profile identifier on which DMZ entries are requested.
in	<i>dmzCb</i>	callback to get the response requestDmzEntry

**Returns**

Status of requestDmzEntry i.e. success or suitable status code.

**Deprecated** Use telux::data::requestDmzEntry API to request DMZ on any backhaul

**5.25.1.4 class telux::data::net::IFirewallEntry**

Firewall entry class is used for configuring firewall rules.

**Public member functions**

- virtual std::shared\_ptr< IIPFilter > getIPProtocolFilter ()=0
- virtual telux::data::Direction getDirection ()=0
- virtual telux::data::IpFamilyType getIpFamilyType ()=0
- virtual uint32\_t getHandle ()=0
- virtual ~IFirewallEntry ()

**Static Public Attributes**

- static const uint32\_t INVALID\_HANDLE = 0

**5.25.1.4.1 Constructors and Destructors**

**5.25.1.4.1.1** virtual telux::data::net::IFirewallEntry::~IFirewallEntry ( ) [virtual]

Destructor for [IFirewallEntry](#)

**5.25.1.4.2 Member Function Documentation**

**5.25.1.4.2.1** virtual std::shared\_ptr<IIPFilter> telux::data::net::IFirewallEntry::getIPProtocolFilter ( ) [pure virtual]

Get IPProtocol filter type

**Returns**

[telux::data::IIPFilter](#).

**5.25.1.4.2.2** virtual telux::data::Direction telux::data::net::IFirewallEntry::getDirection ( ) [pure virtual]

Get firewall direction

**Returns**

[telux::data::Direction](#).

**5.25.1.4.2.3** `virtual telux::data::IpFamilyType telux::data::net::IFirewallEntry::getIpFamilyType ( )`  
`[pure virtual]`

Get Ip FamilyType

#### Returns

[telux::data::IpFamilyType](#).

**5.25.1.4.2.4** `virtual uint32_t telux::data::net::IFirewallEntry::getHandle ( )` `[pure virtual]`

Get the unique handle identifying this Firewall entry in the system

#### Returns

uint32\_t handle if initialized or INVALID\_HANDLE otherwise

### 5.25.1.4.3 Field Documentation

**5.25.1.4.3.1** `const uint32_t telux::data::net::IFirewallEntry::INVALID_HANDLE = 0` `[static]`

## 5.25.1.5 class telux::data::net::IFirewallListener

Interface for Firewall listener object. Client needs to implement this interface to get access to Firewall services notifications like onServiceStatusChange.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

#### Public member functions

- virtual void [onServiceStatusChange](#) ([telux::common::ServiceStatus](#) status)
- virtual [~IFirewallListener](#) ()

### 5.25.1.5.1 Constructors and Destructors

**5.25.1.5.1.1** `virtual telux::data::net::IFirewallListener::~IFirewallListener ( )` `[virtual]`

Destructor for [IFirewallListener](#)

### 5.25.1.5.2 Member Function Documentation

**5.25.1.5.2.1** `virtual void telux::data::net::IFirewallListener::onServiceStatusChange ( telux::common↔  
 ::ServiceStatus status )` `[virtual]`

This function is called when service status changes.

in	status	- <a href="#">ServiceStatus</a>
----	--------	---------------------------------

### 5.25.1.6 struct telux::data::net::L2tpSessionConfig

L2TP tunnel sessions configuration

#### Data fields

Type	Field	Description
uint32_t	locId	Local session id
uint32_t	peerId	Peer session id

### 5.25.1.7 struct telux::data::net::L2tpTunnelConfig

L2TP tunnel configuration

#### Data fields

Type	Field	Description
<a href="#">L2tpProtocol</a>	prot	Encapsulation protocols
uint32_t	locId	Local tunnel id
uint32_t	peerId	Peer tunnel id
uint32_t	localUdpPort	Local udp port - if UDP encapsulation is used
uint32_t	peerUdpPort	Peer udp port - if IP encapsulation is used
string	peerIpv6Addr	Peer IPv6 Address - for Ipv6 tunnels
string	peerIpv4Addr	Peer IPv4 Address - for Ipv4 tunnels
string	locIface	interface name to create L2TP tunnel on
<a href="#">IpFamilyType</a>	ipType	Ip family type <a href="#">telux::data::IpFamilyType</a>
vector< <a href="#">L2tp↔ SessionConfig</a> >	sessionConfig	List of L2tp tunnel sessions

### 5.25.1.8 struct telux::data::net::L2tpSysConfig

L2TP Configuration

#### Data fields

Type	Field	Description
vector< <a href="#">L2tp↔ TunnelConfig</a> >	configList	List of L2tp tunnel configurations
bool	enableMtu	Enable MTU size setting on underlying interfaces to avoid segmentation
bool	enableTcpMss	Enable TCP MSS clamping on L2TP interfaces to avoid segmentation

### 5.25.1.9 class telux::data::net::IL2tpManager

L2tpManager is a primary interface for configuring L2TP Service. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready.

#### Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status setConfig](#) (bool enable, bool enableMss, bool enableMtu, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status addTunnel](#) (const [L2tpTunnelConfig](#) &l2tpTunnelConfig, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status requestConfig](#) ([L2tpConfigCb](#) l2tpConfigCb)=0
- virtual [telux::common::Status removeTunnel](#) (uint32\_t tunnelId, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status registerListener](#) (std::weak\_ptr< [IL2tpListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak\_ptr< [IL2tpListener](#) > listener)=0
- virtual [~IL2tpManager](#) ()

#### 5.25.1.9.1 Constructors and Destructors

**5.25.1.9.1.1** virtual [telux::data::net::IL2tpManager::~~IL2tpManager](#) ( ) [**virtual**]

Destructor for [IL2tpManager](#)

#### 5.25.1.9.2 Member Function Documentation

**5.25.1.9.2.1** virtual bool [telux::data::net::IL2tpManager::isSubsystemReady](#) ( ) [**pure virtual**]

Checks if the L2tp manager subsystem is ready.

#### Returns

True if L2tp Manager is ready for service, otherwise returns false.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.25.1.9.2.2** virtual std::future<bool> [telux::data::net::IL2tpManager::onSubsystemReady](#) ( ) [**pure virtual**]

Wait for L2tp manager subsystem to be ready.

**Returns**

A future that caller can wait on to be notified when L2tp manager is ready.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

```
5.25.1.9.2.3 virtual telux::common::Status telux::data::net::l2tpManager::setConfig ( bool enable,
bool enableMss, bool enableMtu, telux::common::ResponseCallback callback = nullptr
) [pure virtual]
```

Enable L2TP for unmanaged Tunnel State

**Parameters**

in	<i>enable</i>	Enable/Disable L2TP for unmanaged tunnels.
in	<i>enableMss</i>	Enable/Disable TCP MSS to be clamped on L2TP interfaces to avoid Segmentation
in	<i>enableMtu</i>	Enable/Disable MTU size to be set on underlying interfaces to avoid fragmentation
in	<i>callback</i>	optional callback to get the response setConfig

**Returns**

Status of setConfig i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

```
5.25.1.9.2.4 virtual telux::common::Status telux::data::net::l2tpManager::addTunnel ( const L2tp↔
TunnelConfig & l2tpTunnelConfig, telux::common::ResponseCallback callback = nullptr
) [pure virtual]
```

Set L2TP Configuration for one tunnel

**Parameters**

in	<i>l2tpTunnelConfig</i>	Configuration to be set telux::net::L2tpTunnelConfig
in	<i>callback</i>	Optional callback to get the response addTunnel

**Returns**

Status of addTunnel i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.25.1.9.2.5 virtual telux::common::Status telux::data::net::IL2tpManager::requestConfig ( L2tpConfigCb l2tpConfigCb ) [pure virtual]

Get Current L2TP Configuration

**Parameters**

in	<i>l2tpConfigCb</i>	Asynchronous callback to get current L2TP configurations
----	---------------------	--

**Returns**

Status of requestConfig i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.25.1.9.2.6 virtual telux::common::Status telux::data::net::IL2tpManager::removeTunnel ( uint32\_t tunnelId, telux::common::ResponseCallback callback = nullptr ) [pure virtual]

Remove L2TP Tunnel

**Parameters**

in	<i>tunnelId</i>	Tunnel ID to be removed
in	<i>callback</i>	optional callback to get the response removeConfig

**Returns**

Status of removeTunnel i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.25.1.9.2.7 virtual telux::common::Status telux::data::net::IL2tpManager::registerListener ( std::weak\_ptr< IL2tpListener > listener ) [pure virtual]

Register L2TP Manager as listener for Data Service health events like data service available or data service not available.

in	<i>listener</i>	pointer of <a href="#">IL2tpListener</a> object that processes the notification
----	-----------------	---

**Returns**

Status of registerListener success or suitable status code

**5.25.1.9.2.8** virtual telux::common::Status telux::data::net::IL2tpManager::deregisterListener ( std::weak\_ptr< IL2tpListener > *listener* ) [pure virtual]

Removes a previously added listener.

**Parameters**

in	<i>listener</i>	pointer of <a href="#">IL2tpListener</a> object that needs to be removed
----	-----------------	--

**Returns**

Status of deregisterListener success or suitable status code

**5.25.1.10 class telux::data::net::IL2tpListener**

Interface for L2TP listener object. Client needs to implement this interface to get access to L2TP services notifications like onServiceStatusChange.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

**Public member functions**

- virtual void [onServiceStatusChange](#) (telux::common::ServiceStatus status)
- virtual [~IL2tpListener](#) ()

**5.25.1.10.1 Constructors and Destructors**

**5.25.1.10.1.1** virtual telux::data::net::IL2tpListener::~~IL2tpListener ( ) [virtual]

Destructor for [IL2tpListener](#)

**5.25.1.10.2 Member Function Documentation**

**5.25.1.10.2.1** virtual void telux::data::net::IL2tpListener::onServiceStatusChange ( telux::common::↔ ServiceStatus *status* ) [virtual]

This function is called when service status changes.



**Parameters**

in	status	- <a href="#">ServiceStatus</a>
----	--------	---------------------------------

**5.25.1.11 struct telux::data::net::NatConfig**

Structure represents Network Address Translation (NAT) configuration

**Data fields**

Type	Field	Description
string	addr	Private IP address
uint16_t	port	Private port
uint16_t	globalPort	Global port
<a href="#">IpProtocol</a>	proto	IP protocol telux::net::IpProtocol

**5.25.1.12 class telux::data::net::INatManager**

NatManager is a primary interface for configuring static network address translation(SNAT) and DMZ (demilitarized zone). It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready.

**Public member functions**

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status addStaticNatEntry](#) (int profileId, const [NatConfig](#) &snatConfig, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status removeStaticNatEntry](#) (int profileId, const [NatConfig](#) &snatConfig, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status requestStaticNatEntries](#) (int profileId, [StaticNatEntriesCb](#) snatEntriesCb)=0
- virtual [telux::common::Status registerListener](#) (std::weak\_ptr< [INatListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak\_ptr< [INatListener](#) > listener)=0
- virtual [telux::data::OperationType getOperationType](#) ()=0
- virtual [~INatManager](#) ()

**5.25.1.12.1 Constructors and Destructors**

**5.25.1.12.1.1 virtual telux::data::net::INatManager::~INatManager ( ) [virtual]**

Destructor for [INatManager](#)

## 5.25.1.12.2 Member Function Documentation

### 5.25.1.12.2.1 virtual bool telux::data::net::INatManager::isSubsystemReady ( ) [pure virtual]

Checks if the NAT manager subsystem is ready.

#### Returns

True if NAT Manager is ready for service, otherwise returns false.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.25.1.12.2.2 virtual std::future<bool> telux::data::net::INatManager::onSubsystemReady ( ) [pure virtual]

Wait for NAT manager subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when NAT manager is ready.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.25.1.12.2.3 virtual telux::common::Status telux::data::net::INatManager::addStaticNatEntry ( int *profileId*, const NatConfig & *snatConfig*, telux::common::ResponseCallback *callback* = nullptr ) [pure virtual]

Adds a static Network Address Translation (NAT) entry in the NAT table, these entries are persistent across object, connection and reboot lifetimes. To remove an entry it needs a explicit call to [removeStaticNatEntry\(\)](#) API, it supports both IPv4 and IPv6

#### Parameters

in	<i>profileId</i>	Profile identifier to which static entry will be mapped to.
in	<i>snatConfig</i>	snatConfiguration telux::net::NatConfig
in	<i>callback</i>	optional callback to get the response addStaticNatEntry

#### Returns

Status of addStaticNatEntry i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.25.1.12.2.4 virtual telux::common::Status telux::data::net::INatManager::removeStaticNatEntry ( int *profileId*, const NatConfig & *snatConfig*, telux::common::ResponseCallback *callback* = nullptr ) [pure virtual]**

Removes a static Network Address Translation (NAT) entry in the NAT table, it supports both IPv4 and IPv6

**Parameters**

in	<i>profileId</i>	Profile identifier to which static entry will be removed from.
in	<i>snatConfig</i>	snatConfiguration telux::net::NatConfig
in	<i>callback</i>	optional callback to get the response removeStaticNatEntry

**Returns**

Status of removeStaticNatEntry i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.25.1.12.2.5 virtual telux::common::Status telux::data::net::INatManager::requestStaticNatEntries ( int *profileId*, StaticNatEntriesCb *snatEntriesCb* ) [pure virtual]**

Request list of static nat entries available in the NAT table

**Parameters**

in	<i>profileId</i>	Profile identifier to which static entries will be retrieved.
in	<i>snatEntriesCb</i>	Asynchronous callback to get the list of static Network Address Translation (NAT) entries

**Returns**

Status of requestStaticNatEntries i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.25.1.12.2.6** `virtual telux::common::Status telux::data::net::INatManager::registerListener ( std::weak_ptr< INatListener > listener ) [pure virtual]`

Register Nat Manager as listener for Data Service health events like data service available or data service not available.

#### Parameters

in	<i>listener</i>	pointer of <a href="#">INatListener</a> object that processes the notification
----	-----------------	--

#### Returns

Status of registerListener success or suitable status code

**5.25.1.12.2.7** `virtual telux::common::Status telux::data::net::INatManager::deregisterListener ( std::weak_ptr< INatListener > listener ) [pure virtual]`

Removes a previously added listener.

#### Parameters

in	<i>listener</i>	pointer of <a href="#">INatListener</a> object that needs to be removed
----	-----------------	---

#### Returns

Status of deregisterListener success or suitable status code

**5.25.1.12.2.8** `virtual telux::data::OperationType telux::data::net::INatManager::getOperationType ( ) [pure virtual]`

Get the associated operation type for this instance.

#### Returns

OperationType of getOperationType i.e. LOCAL or REMOTE.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.25.1.13 class telux::data::net::INatListener

Interface for Nat listener object. Client needs to implement this interface to get access to Nat services notifications like onServiceStatusChange.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

**Public member functions**

- virtual void [onServiceStatusChange](#) ([telux::common::ServiceStatus](#) status)
- virtual [~INatListener](#) ()

**5.25.1.13.1 Constructors and Destructors**

**5.25.1.13.1.1** virtual [telux::data::net::INatListener::~INatListener](#) ( ) [[virtual](#)]

Destructor for [INatListener](#)

**5.25.1.13.2 Member Function Documentation**

**5.25.1.13.2.1** virtual void [telux::data::net::INatListener::onServiceStatusChange](#) ( [telux::common::ServiceStatus](#) *status* ) [[virtual](#)]

This function is called when service status changes.

**Parameters**

<i>in</i>	<i>status</i>	- <a href="#">ServiceStatus</a>
-----------	---------------	---------------------------------

**5.25.1.14 class telux::data::net::ISocksManager**

SocksManager is a primary interface for configuring legacy Socks proxy server. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready.

**Public member functions**

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status enableSocks](#) (bool enable, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status registerListener](#) (std::weak\_ptr< [ISocksListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak\_ptr< [ISocksListener](#) > listener)=0
- virtual [telux::data::OperationType getOperationType](#) ()=0
- virtual [~ISocksManager](#) ()

**5.25.1.14.1 Constructors and Destructors**

**5.25.1.14.1.1** virtual [telux::data::net::ISocksManager::~~ISocksManager](#) ( ) [[virtual](#)]

Destructor for Socks Manager

### 5.25.1.14.2 Member Function Documentation

#### 5.25.1.14.2.1 virtual bool telux::data::net::ISocksManager::isSubsystemReady ( ) [pure virtual]

Checks if the SocksManager subsystem is ready.

#### Returns

True if SocksManager is ready for service, otherwise returns false.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.25.1.14.2.2 virtual std::future<bool> telux::data::net::ISocksManager::onSubsystemReady ( ) [pure virtual]

Wait for SocksManager subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when SocksManager is ready.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.25.1.14.2.3 virtual telux::common::Status telux::data::net::ISocksManager::enableSocks ( bool *enable*, telux::common::ResponseCallback *callback* = nullptr ) [pure virtual]

Enable or Disable Socks proxy service.

#### Parameters

in	<i>enable</i>	true: enable proxy, false: disable proxy
in	<i>callback</i>	optional callback to get the operation error code if any

#### Returns

Status of proxy enablement i.e. success or suitable status code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.25.1.14.2.4 virtual telux::common::Status telux::data::net::ISocksManager::registerListener ( std::weak\_ptr< ISocksListener > *listener* ) [pure virtual]

Register Socks Manager as listener for Data Service health events like data service available or data service not available.

##### Parameters

in	<i>listener</i>	pointer of <a href="#">ISocksListener</a> object that processes the notification
----	-----------------	--

##### Returns

Status of registerListener success or suitable status code

#### 5.25.1.14.2.5 virtual telux::common::Status telux::data::net::ISocksManager::deregisterListener ( std::weak\_ptr< ISocksListener > *listener* ) [pure virtual]

Removes a previously added listener.

##### Parameters

in	<i>listener</i>	pointer of <a href="#">ISocksListener</a> object that needs to be removed
----	-----------------	---

##### Returns

Status of deregisterListener success or suitable status code

#### 5.25.1.14.2.6 virtual telux::data::OperationType telux::data::net::ISocksManager::getOperationType ( ) [pure virtual]

Get the associated operation type for this instance.

##### Returns

OperationType of getOperationType i.e. LOCAL or REMOTE.

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.25.1.15 class telux::data::net::ISocksListener

Interface for Socks listener object. Client needs to implement this interface to get access to Socks services notifications like onServiceStatusChange.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

**Public member functions**

- virtual void [onServiceStatusChange](#) ([telux::common::ServiceStatus](#) status)
- virtual [~ISocksListener](#) ()

**5.25.1.15.1 Constructors and Destructors**

**5.25.1.15.1.1** virtual [telux::data::net::ISocksListener::~~ISocksListener](#) ( ) [[virtual](#)]

Destructor for [ISocksListener](#)

**5.25.1.15.2 Member Function Documentation**

**5.25.1.15.2.1** virtual void [telux::data::net::ISocksListener::onServiceStatusChange](#) ( [telux::common::ServiceStatus](#) *status* ) [[virtual](#)]

This function is called when service status changes.

**Parameters**

in	<i>status</i>	- <a href="#">ServiceStatus</a>
----	---------------	---------------------------------

**5.25.1.16 class [telux::data::net::IVlanManager](#)**

[VlanManager](#) is a primary interface for configuring VLAN (Virtual Local Area Network). it provide APIs for create, query, remove VLAN interfaces and associate or disassociate with profile IDs. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready.

**Public member functions**

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status](#) [createVlan](#) (const [VlanConfig](#) &vlanConfig, [CreateVlanCb](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [removeVlan](#) (int16\_t vlanId, [InterfaceType](#) ifaceType, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [queryVlanInfo](#) ([QueryVlanResponseCb](#) callback)=0
- virtual [telux::common::Status](#) [bindToBackhaul](#) ([VlanBindConfig](#) vlanBindConfig, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [unbindFromBackhaul](#) ([VlanBindConfig](#) vlanBindConfig, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [queryVlanToBackhaulBindings](#) ([BackhaulType](#) backhaul, [VlanBindingsResponseCb](#) callback)=0



- virtual [telux::common::Status registerListener](#) (std::weak\_ptr< [IVlanListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak\_ptr< [IVlanListener](#) > listener)=0
- virtual [telux::data::OperationType getOperationType](#) ()=0
- virtual [telux::common::Status bindWithProfile](#) (int profileId, int vlanId, [telux::common::ResponseCallback](#) callback)=0
- virtual [telux::common::Status unbindFromProfile](#) (int profileId, int vlanId, [telux::common::ResponseCallback](#) callback)=0
- virtual [telux::common::Status queryVlanMappingList](#) ([VlanMappingResponseCb](#) callback)=0
- virtual [~IVlanManager](#) ()

### 5.25.1.16.1 Constructors and Destructors

5.25.1.16.1.1 virtual [telux::data::net::IVlanManager::~~IVlanManager](#) ( ) [[virtual](#)]

Destructor for [IVlanManager](#)

### 5.25.1.16.2 Member Function Documentation

5.25.1.16.2.1 virtual bool [telux::data::net::IVlanManager::isSubsystemReady](#) ( ) [[pure virtual](#)]

Checks if the data subsystem is ready.

#### Returns

True if VLAN Manager is ready for service, otherwise returns false.

5.25.1.16.2.2 virtual std::future<bool> [telux::data::net::IVlanManager::onSubsystemReady](#) ( ) [[pure virtual](#)]

Wait for data subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when VLAN manager is ready.

5.25.1.16.2.3 virtual [telux::common::Status telux::data::net::IVlanManager::createVlan](#) ( const [VlanConfig](#) & *vlanConfig*, [CreateVlanCb](#) *callback = nullptr* ) [[pure virtual](#)]

Create a VLAN associated with multiple interfaces

#### Note

if interface configured as VLAN for the first time, it may trigger auto reboot.

in	<i>vlanConfig</i>	vlan configuration
out	<i>callback</i>	optional callback to get the response createVlan

**Returns**

Immediate status of [createVlan\(\)](#) request sent i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.25.1.16.2.4 virtual telux::common::Status telux::data::net::IVlanManager::removeVlan ( int16\_t *vlanId*, InterfaceType *ifaceType*, telux::common::ResponseCallback *callback* = nullptr ) [pure virtual]**

Remove VLAN configuration

**Note**

This will delete all clients associated with interface

**Parameters**

in	<i>vlanId</i>	VLAN ID
in	<i>ifaceType</i>	telux::net::InterfaceType
out	<i>callback</i>	optional callback to get the response removeVlan

**Returns**

Immediate status of [removeVlan\(\)](#) request sent i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.25.1.16.2.5 virtual telux::common::Status telux::data::net::IVlanManager::queryVlanInfo ( QueryVlanResponseCb *callback* ) [pure virtual]**

Query information about all the VLANs in the system

**Parameters**

out	<i>callback</i>	Response callback with list of configured VLANs
-----	-----------------	---

**Returns**

Immediate status of [queryVlanInfo\(\)](#) request sent i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.25.1.16.2.6** `virtual telux::common::Status telux::data::net::IVlanManager::bindToBackhaul ( VlanBindConfig vlanBindConfig, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Bind a VLAN with a particular profile id. When a WWAN network interface is brought up using [IDataConnectionManager::startDataCall](#) on that profile id, that interface will be accessible from this VLAN. The behavior of this API is dependent on platform/system configuration. If the platform is configured to allow multiple VLANs to be bound to the same profile id then:

- Binding multiple VLANs to any profile id can be achieved by calling this API with each VLAN id. Each VLAN will be associated with its own bridge.
- Reboot is not triggered with any bind operation. If the platform is not configured to allow multiple VLANs to be bound to the same profile id then:
- Binding VLAN to default profile id will associate it with bridge0 and trigger automatic reboot.
- Binding VLAN to any other profile id will associate it with own bridge.
- Multiple VLAN binding attempt to any profile id will result in error [telux::common::ErrorCode::INVALID\\_OPERATION](#). This setting will be persistent across multiple boots.

**Parameters**

in	<i>vlanBindConfig</i>	Backhaul information and vlan id to bind it to. <a href="#">telux::data::net::VlanBindConfig</a>
out	<i>callback</i>	Callback to get the response of bindToBackhaul API

**Returns**

Immediate status of [bindToBackhaul\(\)](#) request sent i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.25.1.16.2.7** `virtual telux::common::Status telux::data::net::IVlanManager::unbindFromBackhaul ( VlanBindConfig vlanBindConfig, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Unbind VLAN from particular backhaul. This API will stop vlan traffic flow to/from specified backhaul type. Slot ID and profile ID are relevant only for WWAN backhaul. For all other backhaul types, values are don't care.

in	<i>vlanBindConfig</i>	Backhaul information and vlan id to unbind it from. <a href="#">telux::data::net::VlanBindConfig</a>
in	<i>callback</i>	Callback to get the response of unbindFromBackhaul API

**Returns**

Immediate status of [unbindFromBackhaul\(\)](#) request sent i.e. success or suitable status code

**Note**

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

**5.25.1.16.2.8** `virtual telux::common::Status telux::data::net::IVlanManager::queryVlanToBackhaul(BackhaulType backhaul, VlanBindingsResponseCb callback) [pure virtual]`

Query VLAN to backhaul binding configurations

**Parameters**

in	<i>backhaul</i>	Backhaul to query vlan binding for.
in	<i>callback</i>	callback to get the response of queryVlanToBackhaulBindings API

**Returns**

Immediate status of [queryVlanToBackhaulBindings\(\)](#) request sent i.e. success or suitable status code

**Note**

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

**5.25.1.16.2.9** `virtual telux::common::Status telux::data::net::IVlanManager::registerListener (std::weak_ptr< IVlanListener > listener) [pure virtual]`

Register Vlan Manager as a listener for Data Service health events like data service available or data service not available.

**Parameters**

in	<i>listener</i>	pointer of <a href="#">IVlanListener</a> object that processes the notification
----	-----------------	---

**Returns**

Status of registerListener success or suitable status code

**5.25.1.16.2.10** virtual telux::common::Status telux::data::net::IVlanManager::deregisterListener ( std::weak\_ptr< IVlanListener > *listener* ) [pure virtual]

Removes a previously added listener.

#### Parameters

in	<i>listener</i>	pointer of <a href="#">IVlanListener</a> object that needs to be removed
----	-----------------	--

#### Returns

Status of deregisterListener success or suitable status code

**5.25.1.16.2.11** virtual telux::data::OperationType telux::data::net::IVlanManager::getOperationType ( ) [pure virtual]

Get the associated operation type for this instance.

#### Returns

OperationType of getOperationType i.e. LOCAL or REMOTE.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.25.1.16.2.12** virtual telux::common::Status telux::data::net::IVlanManager::bindWithProfile ( int *profileId*, int *vlanId*, telux::common::ResponseCallback *callback* ) [pure virtual]

Bind a Vlan with a particular profile ID. When a WWAN network interface is brought up using [IDataConnectionManager::startDataCall](#) on that profile ID, that interface will be accessible from this Vlan

#### Parameters

in	<i>profileId</i>	profile id for vlan association
in	<i>vlanId</i>	sets vlan id
out	<i>callback</i>	callback to get the response of associateWithProfileId API

#### Returns

Immediate status of associateWithProfileId() request sent i.e. success or suitable status code.

**Deprecated** Use [bindToBackhaul\(\)](#) API below to bind VLAN to backhaul

**5.25.1.16.2.13** virtual telux::common::Status telux::data::net::IVlanManager::unbindFromProfile ( int *profileId*, int *vlanId*, telux::common::ResponseCallback *callback* ) [pure virtual]

Unbind VLAN id with given profile id

#### Parameters

in	<i>profileId</i>	profile id for vlan association
in	<i>vlanId</i>	vlan id
in	<i>callback</i>	callback to get the response of associateWithProfileId API

#### Returns

Immediate status of disassociateFromProfileId() request sent i.e. success or suitable status code

**Deprecated** Use [unbindFromBackhaul\(\)](#) API below to unbind VLAN to backhaul

**5.25.1.16.2.14** virtual telux::common::Status telux::data::net::IVlanManager::queryVlanMappingList ( VlanMappingResponseCb *callback* ) [pure virtual]

Query VLAN mapping list with associated profile id and vlan id

#### Parameters

in	<i>callback</i>	callback to get the response of queryVlanMappingList API
----	-----------------	--

#### Returns

Immediate status of [queryVlanMappingList\(\)](#) request sent i.e. success or suitable status code

**Deprecated** Use [queryVlanToBackhaulBindings\(\)](#) API below to request VLAN to backhaul mapping

### 5.25.1.17 class telux::data::net::IVlanListener

Interface for Vlan listener object. Client needs to implement this interface to get access to Socks services notifications like onServiceStatusChange.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

#### Public member functions

- virtual void [onServiceStatusChange](#) (telux::common::ServiceStatus status)
- virtual void [onHwAccelerationChanged](#) (const ServiceState state)
- virtual [~IVlanListener](#) ()

### 5.25.1.17.1 Constructors and Destructors

5.25.1.17.1.1 `virtual telux::data::net::IVlanListener::~~IVlanListener ( ) [virtual]`

Destructor for [IVlanListener](#)

### 5.25.1.17.2 Member Function Documentation

5.25.1.17.2.1 `virtual void telux::data::net::IVlanListener::onServiceStatusChange ( telux::common::↔ ServiceStatus status ) [virtual]`

This function is called when service status changes.

#### Parameters

in	<i>status</i>	- <a href="#">ServiceStatus</a>
----	---------------	---------------------------------

5.25.1.17.2.2 `virtual void telux::data::net::IVlanListener::onHwAccelerationChanged ( const ServiceState state ) [virtual]`

This function is called when there is a change in IPA Connection Manager daemon state.

#### Parameters

in	<i>state</i>	New state of IPA connection Manager daemon Active/Inactive
----	--------------	--

## 5.25.2 Enumeration Type Documentation

5.25.2.1 `enum telux::data::net::L2tpProtocol [strong]`

L2TP encapsulation protocols

#### Enumerator

***NONE***

***IP*** IP Protocol used for encapsulation

***UDP*** UDP Protocol used for encapsulation

## 5.26 Telematics\_platform\_deviceinfo

### 5.26.1 Data Structure Documentation

#### 5.26.1.1 class telux::platform::IDeviceInfoListener

Listener class for getting device info related notifications . The client needs to implement these methods as briefly as possible and avoid blocking calls in it. The methods in this class can be invoked from multiple different threads. Client needs to make sure that the implementation is thread-safe.

##### Public member functions

- virtual [~IDeviceInfoListener](#) ()

##### 5.26.1.1.1 Constructors and Destructors

###### 5.26.1.1.1.1 virtual telux::platform::IDeviceInfoListener::~~IDeviceInfoListener ( ) [virtual]

Destructor of [IDeviceInfoListener](#)

#### 5.26.1.2 struct telux::platform::PlatformVersion

Structure contains the version of the platform software

##### Data fields

Type	Field	Description
string	meta	Meta Version, for example: SA2150P_SA515M.LE_LE.1-3_2-1-00297-STD.INT-1
string	modem	Modem Version, for example: MPSS.HI.3.1.c3-00114-SDX55_GENAUTO_TEST-1
string	externalApp	External App Version, for example: LE.UM.3.2.3-72102-SA2150p.Int-1
string	integratedApp	Integrated App MDM Version, for example: LE.UM.4.1.1-71802-sa515m.Int-1

#### 5.26.1.3 class telux::platform::IDeviceInfoManager

[IDeviceInfoManager](#) provides interface to to retrieve IMEI and platform version operations.

##### Public member functions

- virtual [telux::common::ServiceStatus getServiceStatus](#) ()=0
- virtual [telux::common::Status registerListener](#) (std::weak\_ptr< [IDeviceInfoListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak\_ptr< [IDeviceInfoListener](#) > listener)=0
- virtual [telux::common::Status getPlatformVersion](#) ([PlatformVersion](#) &pv)=0
- virtual [telux::common::Status getIMEI](#) (std::string &imei)=0



- virtual `~IDeviceInfoManager ()`

### 5.26.1.3.1 Constructors and Destructors

#### 5.26.1.3.1.1 virtual `telux::platform::IDeviceInfoManager::~IDeviceInfoManager ( ) [virtual]`

Destructor of `IDeviceInfoManager`

### 5.26.1.3.2 Member Function Documentation

#### 5.26.1.3.2.1 virtual `telux::common::ServiceStatus telux::platform::IDeviceInfoManager::getServiceStatus ( ) [pure virtual]`

This status indicates whether the object is in a usable state.

#### Returns

`telux::common::ServiceStatus` indicating the current status of the device info service.

#### 5.26.1.3.2.2 virtual `telux::common::Status telux::platform::IDeviceInfoManager::registerListener ( std::weak_ptr< IDeviceInfoListener > listener ) [pure virtual]`

Registers the listener for FileSystem Manager indications.

#### Parameters

in	<i>listener</i>	- pointer to implemented listener.
----	-----------------	------------------------------------

#### Returns

status of the registration request.

#### 5.26.1.3.2.3 virtual `telux::common::Status telux::platform::IDeviceInfoManager::deregisterListener ( std::weak_ptr< IDeviceInfoListener > listener ) [pure virtual]`

Deregisters the previously registered listener.

#### Parameters

in	<i>listener</i>	- pointer to registered listener that needs to be removed.
----	-----------------	--

#### Returns

status of the deregistration request.

#### 5.26.1.3.2.4 virtual telux::common::Status telux::platform::DeviceInfoManager::getPlatformVersion ( PlatformVersion & pv ) [pure virtual]

Get the platform version. Need obtain required permissions from telux\_allow\_version.

##### Parameters

out	<i>pv</i>	- <a href="#">telux::platform::PlatformVersion</a>
-----	-----------	--

##### Returns

- [telux::common::Status](#)

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.26.1.3.2.5 virtual telux::common::Status telux::platform::DeviceInfoManager::getIMEI ( std::string & imei ) [pure virtual]

Get the international mobile equipment identity.

##### Parameters

out	<i>imei</i>	- std::string
-----	-------------	---------------

##### Returns

- [telux::common::Status](#)

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

## 5.27 Telematics\_platform

### 5.27.1 Data Structure Documentation

#### 5.27.1.1 class telux::platform::PlatformFactory

[PlatformFactory](#) allows creation of Platform services related classes.

##### Public member functions

- virtual std::shared\_ptr< [IDeviceInfoManager](#) > [getDeviceInfoManager](#) (telux::common::InitResponseCb callback=nullptr)=0

##### Static Public Member Functions

- static [PlatformFactory](#) & [getInstance](#) ()

##### Protected Member Functions

- [PlatformFactory](#) ()
- virtual [~PlatformFactory](#) ()

#### 5.27.1.1.1 Constructors and Destructors

5.27.1.1.1.1 [telux::platform::PlatformFactory::PlatformFactory](#) ( ) [[protected](#)]

5.27.1.1.1.2 [virtual telux::platform::PlatformFactory::~~PlatformFactory](#) ( ) [[protected](#)], [[virtual](#)]

#### 5.27.1.1.2 Member Function Documentation

5.27.1.1.2.1 [static PlatformFactory& telux::platform::PlatformFactory::getInstance](#) ( ) [[static](#)]

Get instance of platform Factory

5.27.1.1.2.2 [virtual std::shared\\_ptr<IDeviceInfoManager> telux::platform::PlatformFactory::get↔ DeviceInfoManager](#) ( [telux::common::InitResponseCb callback = nullptr](#) ) [[pure virtual](#)]

Get instance of device info manager ([IDeviceInfoManager](#)). The device info manager supports device info request like retrieving IMEI and platform version.

##### Parameters

in	<i>callback</i>	Optional callback to get the initialization status of FsManager. <a href="#">telux::common::InitResponseCb</a>
----	-----------------	--

**Returns**

pointer of [IDeviceInfoManager](#) object.

## 5.28 Telematics\_cellbroadcast

### 5.28.1 Data Structure Documentation

#### 5.28.1.1 struct telux::tel::CellBroadcastFilter

Defines cellbroadcast message filter. Refer spec 3GPP TS 23.041 9.4.1.2.2 for message identifier. Eg: If user want to receive from 0x1112 to 0x1116 then, startMessageId is 0x1112 and endMessageId is 0x1116. If user want to receive only 0x1112, then both startMessageId and endMessageId is 0x1112.

##### Data fields

Type	Field	Description
int	startMessageId	Intended to receive start from which MessageType
int	endMessageId	Intended to receive upto which MessageType

### 5.28.2 Enumeration Type Documentation

#### 5.28.2.1 enum telux::tel::GeographicalScope [strong]

Defines geographical scope of cell broadcast.

##### Enumerator

**CELL\_WIDE\_IMMEDIATE** Cell wide geographical scope with immediate display

**PLMN\_WIDE** PLMN wide geographical scope

**LA\_WIDE**

**CELL\_WIDE** Location / Service/ Tracking area wide geographical scope (GSM/UMTS/E-UTRAN/NG-RAN). Cell wide geographical scope

#### 5.28.2.2 enum telux::tel::MessagePriority [strong]

Defines priority for cell broadcast message.

##### Enumerator

**UNKNOWN** Unknown message priority

**NORMAL** Normal message priority

**EMERGENCY** Emergency message priority

#### 5.28.2.3 enum telux::tel::MessageType [strong]

Defines message type for cell broadcast message.

##### Enumerator

**UNKNOWN** Unknown message type

**ETWS** Earthquake and Tsunami Warning System

**CMAS** Commercial Mobile Alert System

**5.28.2.4 enum telux::tel::EtwsWarningType [strong]**

Defines warning type for ETWS cell broadcast message.

**Enumerator**

**UNKNOWN** Unknown ETWS warning type  
**EARTHQUAKE** ETWS warning type for earthquake  
**TSUNAMI** ETWS warning type for tsunami  
**EARTHQUAKE\_AND\_TSUNAMI** ETWS warning type for earthquake and tsunami  
**TEST\_MESSAGE** ETWS warning type for test messages  
**OTHER\_EMERGENCY** ETWS warning type for other emergency types

**5.28.2.5 enum telux::tel::CmasMessageClass [strong]**

Defines message class for CMAS cell broadcast message.

**Enumerator**

**UNKNOWN** CMAS category for warning types that are reserved for future extension  
**PRESIDENTIAL\_LEVEL\_ALERT** Presidential-level alert (Korean Public Alert System Class 0 message)  
**EXTREME\_THREAT** Extreme threat to life and property (Korean Public Alert System Class 1 message)  
**SEVERE\_THREAT** Severe threat to life and property (Korean Public Alert System Class 1 message).  
**CHILD\_ABDUCTION\_EMERGENCY** Child abduction emergency (AMBER Alert)  
**REQUIRED\_MONTHLY\_TEST** CMAS test message  
**CMAS\_EXERCISE** CMAS exercise  
**OPERATOR\_DEFINED\_USE** CMAS category for operator defined use

**5.28.2.6 enum telux::tel::CmasSeverity [strong]**

Defines severity type for CMAS cell broadcast message.

**Enumerator**

**UNKNOWN** CMAS alert severity is unknown. The severity is available for all GSM/UMTS alerts except for the Presidential-level alert class (Korean Public Alert System Class 0).  
**EXTREME** Extraordinary threat to life or property  
**SEVERE** Significant threat to life or property

**5.28.2.7 enum telux::tel::CmasUrgency [strong]**

Defines urgency type for CMAS cell broadcast message.

**Enumerator**

**UNKNOWN** CMAS alert urgency is unknown. The urgency is available for all GSM/UMTS alerts except for the Presidential-level alert class (Korean Public Alert System Class 0).  
**IMMEDIATE** Responsive action should be taken immediately  
**EXPECTED** Responsive action should be taken within the next hour

### 5.28.2.8 enum telux::tel::CmasCertainty [strong]

Defines certainty type for CMAS cell broadcast message.

#### Enumerator

**UNKNOWN** CMAS alert certainty is unknown. The certainty is available for all GSM/UMTS alerts except for the Presidential-level alert class (Korean Public Alert System Class 0).

**OBSERVED** Determined to have occurred or to be ongoing.

**LIKELY** Likely (probability > ~50%)

### 5.28.2.9 enum telux::tel::GeometryType [strong]

Defines geometry type specified in wireless emergency alert.

#### Enumerator

**UNKNOWN** Unknown geometry type

**POLYGON** Polygon geometry type

**CIRCLE** Circle geometry type

## 5.29 Telematics\_ecall

### 5.29.1 Data Structure Documentation

#### 5.29.1.1 class telux::tel::IEcallManager

[IEcallManager](#) allows operations related to automotive emergency call management and its related configurations.

##### Public member functions

- virtual [telux::common::ServiceStatus](#) getServiceStatus ()=0
- virtual [telux::common::Status](#) setConfig ([EcallConfig](#) config)=0
- virtual [telux::common::Status](#) getConfig ([EcallConfig](#) &config)=0
- virtual [telux::common::Status](#) registerListener (std::weak\_ptr< [IEcallListener](#) > listener)=0
- virtual [telux::common::Status](#) deregisterListener (std::weak\_ptr< [IEcallListener](#) > listener)=0
- virtual [~IEcallManager](#) ()

##### 5.29.1.1.1 Constructors and Destructors

5.29.1.1.1 virtual [telux::tel::IEcallManager::~IEcallManager](#) ( ) [[virtual](#)]

##### 5.29.1.1.2 Member Function Documentation

5.29.1.1.2.1 virtual [telux::common::ServiceStatus](#) [telux::tel::IEcallManager::getServiceStatus](#) ( )  
[[pure virtual](#)]

Checks the status of [IEcallManager](#) sub-system and returns the result.

##### Returns

the status of [IEcallManager](#) sub-system status [telux::common::ServiceStatus](#)

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.29.1.1.2.2 virtual [telux::common::Status](#) [telux::tel::IEcallManager::setConfig](#) ( [EcallConfig](#) *config* )  
[[pure virtual](#)]

Set the configuration related to emergency call. The configuration is persistent and takes effect when the next emergency call is dialed.



in	<i>config</i>	eCall configuration to be set <a href="#">EcallConfig</a>
----	---------------	---

**Returns**

Status of setConfig i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.29.1.1.2.3 virtual telux::common::Status telux::tel::IEcallManager::getConfig ( EcallConfig & *config* ) [pure virtual]

Get the configuration related to emergency call.

**Parameters**

out	<i>config</i>	Parameter to hold the fetched eCall configuration <a href="#">EcallConfig</a>
-----	---------------	---

**Returns**

Status of getConfig i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.29.1.1.2.4 virtual telux::common::Status telux::tel::IEcallManager::registerListener ( std::weak\_ptr< IEcallListener > *listener* ) [pure virtual]

Register a listener for notifications from the EcallManager.

**Parameters**

in	<i>listener</i>	Pointer to <a href="#">IEcallListener</a> object that processes the notification
----	-----------------	--

**Returns**

Status of registerListener i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.29.1.1.2.5** `virtual telux::common::Status telux::tel::IEcallManager::deregisterListener ( std::weak_ptr< IEcallListener > listener ) [pure virtual]`

Deregister a previously registered listener.

#### Parameters

in	<i>listener</i>	Pointer to <a href="#">IEcallListener</a> object that needs to be deregistered.
----	-----------------	---

#### Returns

Status of deregisterListener i.e. success or suitable error code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.29.1.2 class telux::tel::IEcallListener

Listener class to notify service status change notifications. The listener method can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe.

#### Public member functions

- virtual [~IEcallListener](#) ()

#### 5.29.1.2.1 Constructors and Destructors

**5.29.1.2.1.1** `virtual telux::tel::IEcallListener::~~IEcallListener ( ) [virtual]`

Destructor of [IEcallListener](#)

## 5.30 Telematics\_ims\_settings

### 5.30.1 Data Structure Documentation

#### 5.30.1.1 struct telux::tel::ImsServiceConfig

Defines the selected IMS service configuration parameters and their corresponding value

##### Data fields

Type	Field	Description
<a href="#">ImsService</a> ↔ <a href="#">ConfigValidity</a>	config↔ ValidityMask	Indicates the configurations type. Bit set to 1 denotes the config is valid.
bool	imsService↔ Enabled	Enable/Disable IMS service
bool	voImsEnabled	Enable/Disable VOIMS service

#### 5.30.1.2 class telux::tel::ImsSettingsManager

ImsSettingsManager allows IMS settings. For example enabling or disabling IMS service, VOIMS service.

##### Public member functions

- virtual [telux::common::ServiceStatus](#) getServiceStatus ()=0
- virtual [telux::common::Status](#) requestServiceConfig (SlotId slotId, [ImsServiceConfigCb](#) callback)=0
- virtual [telux::common::Status](#) setServiceConfig (SlotId slotId, [ImsServiceConfig](#) config, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) registerListener (std::weak\_ptr< [ImsSettingsListener](#) > listener)=0
- virtual [telux::common::Status](#) deregisterListener (std::weak\_ptr< [ImsSettingsListener](#) > listener)=0
- virtual [~ImsSettingsManager](#) ()

#### 5.30.1.2.1 Constructors and Destructors

5.30.1.2.1.1 virtual [telux::tel::ImsSettingsManager::~ImsSettingsManager](#) ( ) [[virtual](#)]

#### 5.30.1.2.2 Member Function Documentation

5.30.1.2.2.1 virtual [telux::common::ServiceStatus](#) [telux::tel::ImsSettingsManager::getServiceStatus](#) ( ) [[pure virtual](#)]

This status indicates whether the [ImsSettingsManager](#) object is in a usable state.

##### Returns

[SERVICE\\_AVAILABLE](#) - If IMS settings manager is ready for service. [SERVICE\\_UNAVAILABLE](#) - If IMS settings manager is temporarily unavailable. [SERVICE\\_FAILED](#) - If IMS settings manager encountered an irrecoverable failure.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.30.1.2.2.2 virtual telux::common::Status telux::tel::ImsSettingsManager::requestServiceConfig ( SlotId *slotId*, ImsServiceConfigCb *callback* ) [pure virtual]

Request the IMS service configurations

**Parameters**

in	<i>slotId</i>	Slot for which the IMS service configurations is requested.
in	<i>callback</i>	Callback function to get the response of request IMS service configurations.

**Returns**

Status of requestServiceConfig i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.30.1.2.2.3 virtual telux::common::Status telux::tel::ImsSettingsManager::setServiceConfig ( SlotId *slotId*, ImsServiceConfig *config*, common::ResponseCallback *callback* = *nullptr* ) [pure virtual]

To configure the IMS service configurations. Also specify whether configuration needs to be enabled or disabled.

**Parameters**

in	<i>slotId</i>	Slot for which the IMS service configuration is intended.
in	<i>config</i>	Indicates which configuration are configured currently and whether the config is enabled or disabled. <a href="#">ImsServiceConfig</a> .
in	<i>callback</i>	Callback function to get the response of set IMS service configuration request.

**Returns**

Status of setServiceConfig i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.30.1.2.2.4 virtual telux::common::Status telux::tel::ImsSettingsManager::registerListener ( std::weak\_ptr< ImsSettingsListener > *listener* ) [pure virtual]

Register a listener for specific events in the IMS settings subsystem.

##### Parameters

in	<i>listener</i>	Pointer to <a href="#">ImsSettingsListener</a> object that processes the notification
----	-----------------	---

##### Returns

Status of registerListener i.e. success or suitable error code.

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.30.1.2.2.5 virtual telux::common::Status telux::tel::ImsSettingsManager::deregisterListener ( std::weak\_ptr< ImsSettingsListener > *listener* ) [pure virtual]

Deregister the previously added listener.

##### Parameters

in	<i>listener</i>	Pointer to <a href="#">ImsSettingsListener</a> object that needs to be deregistered.
----	-----------------	--

##### Returns

Status of deregisterListener i.e. success or suitable error code.

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.30.1.3 class telux::tel::ImsSettingsListener

Listener class for getting IMS service configuration change notifications. The listener method can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe.

##### Public member functions

- virtual void [onImsServiceConfigsChange](#) (SlotId slotId, [ImsServiceConfig](#) config)
- virtual void [onServiceStatusChange](#) (telux::common::ServiceStatus status)
- virtual [~ImsSettingsListener](#) ()

### 5.30.1.3.1 Constructors and Destructors

#### 5.30.1.3.1.1 virtual telux::tel::ImsSettingsListener::~ImsSettingsListener ( ) [virtual]

Destructor of [ImsSettingsListener](#)

### 5.30.1.3.2 Member Function Documentation

#### 5.30.1.3.2.1 virtual void telux::tel::ImsSettingsListener::onImsServiceConfigsChange ( SlotId *slotId*, ImsServiceConfig *config* ) [virtual]

This function is called whenever any IMS service configuration is changed.

#### Parameters

in	<i>slotId</i>	SIM corresponding to slot identifier for which the IMS service configuration has changed.
in	<i>config</i>	Indicates which configuration is valid and whether the config is enabled or disabled. <a href="#">ImsServiceConfig</a> .

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.30.1.3.2.2 virtual void telux::tel::ImsSettingsListener::onServiceStatusChange ( telux::common::↔ ServiceStatus *status* ) [virtual]

This function is called when [ImsSettingsManager](#) service status changes.

#### Parameters

in	<i>status</i>	- ServiceStatus
----	---------------	-----------------

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

## 5.30.2 Enumeration Type Documentation

### 5.30.2.1 enum telux::tel::ImsServiceConfigType

Defines the IMS service configuration parameters

#### Enumerator

**IMSSETTINGS\_VOIMS** Voice calling support on LTE

**IMSSETTINGS\_IMS\_SERVICE** IMS Normal Registration configuration

## 5.31 Telematics\_multi\_sim

### 5.31.1 Data Structure Documentation

#### 5.31.1.1 struct telux::tel::SlotStatus

Represents status of a physical SIM slot

##### Data fields

Type	Field	Description
<a href="#">SlotState</a>	slotState	State of the physical SIM slot
<a href="#">CardState</a>	cardState	Status of the card in the physical slot
<a href="#">CardError</a>	cardError	Indicates the reason for the card error, and is valid only when the card state is CARDSTATE_ERROR.

#### 5.31.1.2 class telux::tel::IMultiSimManager

MultiSimManager allows to perform operation pertaining to devices which have more than one SIM/UICC card. Clients should check if the subsystem is ready before invoking any of the APIs as follows.

```
bool isReady = MultiSimManager->isSubsystemReady();
```

##### Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status](#) [getSlotCount](#) (int &count)=0
- virtual [telux::common::Status](#) [requestHighCapability](#) ([HighCapabilityCallback](#) callback)=0
- virtual [telux::common::Status](#) [setHighCapability](#) (int slotId, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [switchActiveSlot](#) (SlotId slotId, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [requestSlotStatus](#) ([SlotStatusCallback](#) callback)=0
- virtual [telux::common::Status](#) [registerListener](#) (std::weak\_ptr< [IMultiSimListener](#) > listener)=0
- virtual [telux::common::Status](#) [deregisterListener](#) (std::weak\_ptr< [IMultiSimListener](#) > listener)=0
- virtual [~IMultiSimManager](#) ()

#### 5.31.1.2.1 Constructors and Destructors

```
5.31.1.2.1.1 virtual telux::tel::IMultiSimManager::~IMultiSimManager ( ) [virtual]
```

### 5.31.1.2.2 Member Function Documentation

#### 5.31.1.2.2.1 `virtual bool telux::tel::IMultiSimManager::isSubsystemReady ( ) [pure virtual]`

Checks the status of Multi SIM subsystem and returns the result.

#### Returns

If true MultiSimManager is ready.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.31.1.2.2.2 `virtual std::future<bool> telux::tel::IMultiSimManager::onSubsystemReady ( ) [pure virtual]`

Wait for Multi SIM subsystem to be ready.

#### Returns

A future that caller can wait on to be notified when Multi SIM subsystem is ready.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.31.1.2.2.3 `virtual telux::common::Status telux::tel::IMultiSimManager::getSlotCount ( int & count ) [pure virtual]`

Get SIM slot count. The count can be used to determine whether the device supports multi SIM.

#### Parameters

out	<i>count</i>	Slot count.
-----	--------------	-------------

#### Returns

Status of getSlotCount i.e. success or suitable error code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.



#### 5.31.1.2.2.4 **virtual telux::common::Status telux::tel::IMultiSimManager::requestHighCapability ( HighCapabilityCallback *callback* ) [pure virtual]**

Request to find out which SIM/slot is allowed to use advance Radio Technology like 5G at a time. For example SIM/slot with high capability may allowed to use RAT capabilities like 5G/4G/3G/2G while the SIM/slot with low capability may be allowed to use RAT capabilities like 4G/2G.

##### Parameters

in	<i>callback</i>	Callback function to get the response of request high capability.
----	-----------------	---

##### Returns

Status of requestHighCapability i.e. success or suitable error code.

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.31.1.2.2.5 **virtual telux::common::Status telux::tel::IMultiSimManager::setHighCapability ( int *slotId*, common::ResponseCallback *callback* = nullptr ) [pure virtual]**

Set SIM/slot with high capability asynchronously. On dual SIM devices, only one SIM may be allowed to use advanced Radio technology like 5G at a time. This API sets the SIM/slot that should be allowed the highest RAT capability. The other SIM/slot will be given lower RAT capabilities. For example, SIM in slot1 will be allowed 2G/3G/4G/5G and the SIM in slot2 will be allowed only 2G/4G.

##### Parameters

in	<i>slotId</i>	Slot set with high capability.
in	<i>callback</i>	Callback function to get the response of set high capability request.

##### Returns

Status of setHighCapability i.e. success or suitable error code.

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.31.1.2.2.6 **virtual telux::common::Status telux::tel::IMultiSimManager::switchActiveSlot ( SlotId *slotId*, common::ResponseCallback *callback* = nullptr ) [pure virtual]**

Choose the physical SIM slot to be used by modem on Single-SIM TCU platforms. After switching the slot, only the SIM on chosen physical slot can be used for WWAN functionality.

in	<i>slotId</i>	physical slot to be made active
in	<i>callback</i>	Callback function to get the response of slot switch request

**Returns**

Status of switchActiveSlot i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.31.1.2.2.7 virtual telux::common::Status telux::tel::IMultiSimManager::requestSlotStatus ( SlotStatusCallback *callback* ) [pure virtual]

Request the status of physical slots.

**Parameters**

in	<i>callback</i>	Callback function to get the response of slot status request
----	-----------------	--

**Returns**

Status of requestSlotStatus i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 5.31.1.2.2.8 virtual telux::common::Status telux::tel::IMultiSimManager::registerListener ( std::weak\_ptr< IMultiSimListener > *listener* ) [pure virtual]

Register a listener for specific events in the Multi SIM subsystem.

**Parameters**

in	<i>listener</i>	Pointer to <a href="#">IMultiSimListener</a> object that processes the notification
----	-----------------	---

**Returns**

Status of registerListener i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**5.31.1.2.2.9** `virtual telux::common::Status telux::tel::IMultiSimManager::deregisterListener ( std::weak_ptr< IMultiSimListener > listener ) [pure virtual]`

Deregister the previously added listener.

#### Parameters

in	<i>listener</i>	Pointer to <a href="#">IMultiSimListener</a> object that needs to be deregistered.
----	-----------------	--

#### Returns

Status of deregisterListener i.e. success or suitable error code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.31.1.3 class telux::tel::IMultiSimListener

Listener class for getting high capability change notification. The listener method can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe.

#### Public member functions

- virtual void [onHighCapabilityChanged](#) (int slotId)
- virtual void [onSlotStatusChanged](#) (std::map< SlotId, [SlotStatus](#) > slotStatus)
- virtual [~IMultiSimListener](#) ()

#### 5.31.1.3.1 Constructors and Destructors

**5.31.1.3.1.1** `virtual telux::tel::IMultiSimListener::~~IMultiSimListener ( ) [virtual]`

Destructor of [IMultiSimListener](#)

#### 5.31.1.3.2 Member Function Documentation

**5.31.1.3.2.1** `virtual void telux::tel::IMultiSimListener::onHighCapabilityChanged ( int slotId ) [virtual]`

This function is called whenever there is change in high capability for SIM/slot.

#### Parameters

in	<i>slotId</i>	SIM corresponding to slot identifier has high capability now.
----	---------------	---

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 5.31.1.3.2 virtual void telux::tel::IMultiSimListener::onSlotStatusChanged ( std::map< SlotId, SlotStatus > slotStatus ) [virtual]

This function is called whenever there is change in physical SIM slots status.

**Parameters**

in	<i>slotStatus</i>	list of slots status <a href="#">SlotStatus</a>
----	-------------------	---

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

## 5.31.2 Enumeration Type Documentation

### 5.31.2.1 enum telux::tel::SlotState [strong]

Represents state of the physical SIM slot

**Enumerator**

**UNKNOWN**

**INACTIVE** Slot is inactive

**ACTIVE** Slot is active

# 6 Namespace Documentation

---

## 6.1 telux Namespace Reference

DeviceConfig provides utility functions to get device configuration details such as multi SIM support.

### Namespaces

- [audio](#)
- [common](#)
- [config](#)
- [cv2x](#)
- [data](#)
- [loc](#)
- [platform](#)
- [power](#)
- [tel](#)
- [therm](#)
- [wlan](#)

DeviceConfig provides utility functions to get device configuration details such as multi SIM support.

## 6.2 telux::audio Namespace Reference

### Data Structures

- struct [AmrwbpParams](#)
- class [AudioFactory](#)  
*AudioFactory allows creation of audio manager. [More...](#)*
- struct [ChannelVolume](#)
- struct [DtmfTone](#)
- struct [FormatInfo](#)
- struct [FormatParams](#)

- class [IAudioBuffer](#)

*Stream Buffer manages the buffer to be used for read and write operations on Audio Streams. For write operations, applications should request a stream buffer, populate it with the data and then pass it to the write operation and set the dataSize that is to be written to the stream. Similarly for read operations, the application should request a stream buffer and use that in the read operation. At the end of the read, the stream buffer will contain the data read. Once an operation (read/write) has completed, the stream buffer could be reused for a subsequent read/write operation, provided `reset()` API called on stream buffer between subsequent calls. [More...](#)*
- class [IAudioCaptureStream](#)

*IAudioCaptureStream represents single audio capture stream. [More...](#)*
- class [IAudioDevice](#)

*Audio device and it's characteristics like Direction (Sink or Source), type. [More...](#)*
- class [IAudioListener](#)
- class [IAudioLoopbackStream](#)

*IAudioLoopbackStream represents audio loopback stream. [More...](#)*
- class [IAudioManager](#)

*Audio Manager is a primary interface for audio operations. It provide APIs to manage Streams ( like voice, play, record etc) and sound cards. [More...](#)*
- class [IAudioPlayStream](#)

*IAudioPlayStream represents single audio playback stream. [More...](#)*
- class [IAudioStream](#)

*IAudioStream represents single audio stream with base properties. [More...](#)*
- class [IAudioToneGeneratorStream](#)

*IAudioToneGeneratorStream represents tone generator stream. [More...](#)*
- class [IAudioVoiceStream](#)

*IAudioVoiceStream represents single voice stream. [More...](#)*
- class [IPlayListener](#)
- class [IStreamBuffer](#)
- class [ITranscodeListener](#)
- class [ITranscoder](#)

*ITranscoder is used to convert one audio format to another audio format using the transcoding operation. [More...](#)*
- class [IVoiceListener](#)

*Listener class for getting notifications related to DTMF tone detection. The client needs to implement these methods as briefly as possible and avoid blocking calls in it. The methods in this class can be invoked from multiple different threads. Client needs to make sure that the implementation is thread-safe. [More...](#)*
- struct [StreamBuffer](#)
- struct [StreamConfig](#)

- struct [StreamMute](#)
- struct [StreamVolume](#)

## 6.2.1 Variable Documentation

### 6.2.1.1 `const uint16_t telux::audio::INFINITE_DTMF_DURATION = 0xFFFF`

Duration to play DTMF tone for infinite time

### 6.2.1.2 `const uint16_t telux::audio::INFINITE_TONE_DURATION = 0xFFFF`

## 6.3 telux::common Namespace Reference

### Data Structures

- class [DeviceConfig](#)
- class [ICommandCallback](#)
- class [ICommandResponseCallback](#)

*General command response callback for most of the requests, client needs to implement this interface to get single shot response. [More...](#)*

- class [IServiceStatusListener](#)
- class [Log](#)
- struct [SdkVersion](#)
- class [Version](#)

*Provides version of SDK. [More...](#)*

## 6.3.1 Typedef Documentation

### 6.3.1.1 `using telux::common::ResponseCallback = typedef std::function<void(telux↔::common::ErrorCode errorCode)>`

General response callback for most of the requests, client needs to implement this function to get the asynchronous response.

The methods in callback can be invoked from multiple different threads. The implementation should be thread safe.

#### Parameters

in	<i>errorCode</i>	<a href="#">ErrorCode</a>
----	------------------	---------------------------

### 6.3.1.2 `using telux::common::InitResponseCb = typedef std::function<void(telux↔::common::ServiceStatus status)>`

This API is invoked when the initialization of an object completes.

in	status	- Service status
----	--------	------------------

## 6.4 telux::config Namespace Reference

### Data Structures

- class [ConfigFactory](#)

*ConfigFactory* allows creation of config related classes. [More...](#)

- struct [ConfigInfo](#)

- class [IModemConfigListener](#)

*Listener class for getting notifications related to configuration change detection. The client needs to implement these methods as briefly as possible and avoid blocking calls in it. The methods in this class can be invoked from multiple different threads. Client needs to make sure that the implementation is thread-safe. [More...](#)*

- class [IModemConfigManager](#)

*IModemConfigManager* provides interface to list config files present in modem's storage. load a new config file in modem, activate a config file, get active config file information, deactivate a config file, delete config file from the modem's storage, get and set mode of config auto selection, register and deregister listener for config update in modem. The config files are also referred to as MBNs. [More...](#)

## 6.5 telux::cv2x Namespace Reference

### Data Structures

- struct [ConfigEventInfo](#)

- class [Cv2xFactory](#)

*Cv2xFactory* is the factory that creates the Cv2x Radio. [More...](#)

- struct [Cv2xPoolStatus](#)

- struct [Cv2xRadioCapabilities](#)

- struct [Cv2xStatus](#)

- struct [Cv2xStatusEx](#)

- struct [DataSessionSettings](#)

- struct [EventFlowInfo](#)

- class [ICv2xConfig](#)

*Cv2xConfig* provide operations to update or request cv2x configuration. [More...](#)

- class [ICv2xConfigListener](#)

*Listeners for ICv2xConfig* must implement this interface. [More...](#)

- class [ICv2xListener](#)

*Cv2x Radio Manager* listeners implement this interface.



- class [ICv2xRadio](#)
- class [ICv2xRadioListener](#)  
*Listeners for Cv2xRadio must implement this interface. [More...](#)*
- class [ICv2xRadioManager](#)  
*Cv2xRadioManager manages instances of Cv2xRadio. [More...](#)*
- class [ICv2xRxSubscription](#)
- class [ICv2xThrottleManager](#)  
*ThrottleManager provides throttle manager client interface. [More...](#)*
- class [ICv2xThrottleManagerListener](#)  
*Listener class for getting filter rate update notification. [More...](#)*
- class [ICv2xTxFlow](#)
- class [ICv2xTxRxSocket](#)
- class [ICv2xTxStatusReportListener](#)  
*Listeners for CV2X Tx status report must implement this interface. [More...](#)*
- struct [IPv6Address](#)
- struct [MacDetails](#)
- struct [RFTxInfo](#)
- struct [SocketInfo](#)
- struct [SpsFlowInfo](#)
- struct [SpsSchedulingInfo](#)
- struct [TrustedUEInfo](#)
- struct [TrustedUEInfoList](#)
- struct [TxPoolIdInfo](#)
- struct [TxStatusReport](#)

## 6.5.1 Typedef Documentation

**6.5.1.1** `using telux::cv2x::CreateRxSubscriptionCallback = typedef std::function<void (std::shared_ptr<ICv2xRxSubscription> rxSub, telux::common::ErrorCode error)>`

This function is called as a response to createRxSubscription.

**Parameters**

in	<i>rxSub</i>	- Rx Subscription
in	<i>error</i>	- Indicates whether socket creation succeeded <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> </ul>

**6.5.1.2 using telux::cv2x::CreateTxSpsFlowCallback = typedef std::function<void (std::shared\_ptr<ICv2xTxFlow> txSpsFlow, std::shared\_ptr<ICv2xTxFlow> txEventFlow, telux::common::ErrorCode spsError, telux::common::ErrorCode eventError)>**

This function is called as a response to createTxSpsFlow

**Parameters**

in	<i>spsFlow</i>	- Sps flow
in	<i>eventFlow</i>	- Optional event flow. Will be nullptr if event flow was not specified in the request
in	<i>error</i>	- Indicates whether Tx SPS flow creation succeeded <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> </ul>
in	<i>error</i>	- Indicates whether optional Tx Event flow creation succeeded <ul style="list-style-type: none"> <li>• SUCCESS</li> </ul>

**6.5.1.3 using telux::cv2x::CreateTxEventFlowCallback = typedef std::function<void (std::shared\_ptr<ICv2xTxFlow> txEventFlow, telux::common::ErrorCode error)>**

This function is called with the response to createTxEventFlow

**Parameters**

in	<i>txEventFlow</i>	- Event flow
in	<i>error</i>	- Indicates whether Tx event flow creation succeeded <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> </ul>

**6.5.1.4 using telux::cv2x::CloseTxFlowCallback = typedef std::function<void (std::shared\_ptr<ICv2xTxFlow> txFlow, telux::common::ErrorCode error)>**

This function is called with the response to closeTxFlow.

in	<i>txFlow</i>	- Closed tx flow
in	<i>error</i>	- Indicates whether close operation succeeded <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> </ul>

#### 6.5.1.5 using `telux::cv2x::CloseRxSubscriptionCallback = typedef std::function<void (std::shared_ptr<ICv2xRxSubscription> rxSub, telux::common::ErrorCode error)>`

This function is called with the response to `closeRxSubscription`.

##### Parameters

in	<i>rxSub</i>	- Closed rx subscription
in	<i>error</i>	- Indicates whether Rx subscription close succeeded <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> </ul>

#### 6.5.1.6 using `telux::cv2x::ChangeSpsFlowInfoCallback = typedef std::function<void (std::shared_ptr<ICv2xTxFlow> txFlow, telux::common::ErrorCode error)>`

This function is called with the response to `changeSpsFlowInfo`.

##### Parameters

in	<i>txFlow</i>	- Sps flow that requested reservation change
in	<i>error</i>	- SUCCESS if Tx reservation change succeeded <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> </ul>

#### 6.5.1.7 using `telux::cv2x::RequestSpsFlowInfoCallback = typedef std::function<void (std::shared_ptr<ICv2xTxFlow> txFlow, const SpsFlowInfo & spsInfo, telux::common::ErrorCode error)>`

This function is called with the response to `requestSpsFlowInfo`.

##### Parameters

in	<i>txFlow</i>	- SPS flow that requested info
in	<i>spsInfo</i>	- SPS flow reservation info
in	<i>error</i>	- SUCCESS if Tx reservation change succeeded <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> </ul>

### 6.5.1.8 using telux::cv2x::ChangeEventFlowInfoCallback = typedef std::function<void (std::shared\_ptr<ICv2xTxFlow> txFlow, telux::common::ErrorCode error)>

This function is called with the response to changeEventFlowInfo.

#### Parameters

in	<i>txFlow</i>	- Event flow that requested reservation change
in	<i>error</i>	- SUCCESS if Tx parameter change succeeded <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> </ul>

### 6.5.1.9 using telux::cv2x::RequestCapabilitiesCallback = typedef std::function<void(const Cv2xRadioCapabilities & capabilities, telux::common::ErrorCode error)>

This function is called with the response to requestCapabilities.

#### Parameters

in	<i>capabilities</i>	- Capability info
in	<i>error</i>	- SUCCESS if capabilities request succeeded <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> </ul>

### 6.5.1.10 using telux::cv2x::RequestDataSessionSettingsCallback = typedef std::function<void (const DataSessionSettings & settings, telux::common::← ErrorCode error)>

This function is called with the response to requestDataSessionSettings.

#### Parameters

in	<i>settings</i>	- Data session settings
in	<i>error</i>	- SUCCESS if data session settings request succeeded <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> </ul>

### 6.5.1.11 using telux::cv2x::UpdateTrustedUEListCallback = typedef std← ::function<void(telux::common::ErrorCode error)>

This function is called with the response to updateTrustedUEList.

in	<i>error</i>	- SUCCESS if update succeeded <ul style="list-style-type: none"> <li>• INVALID_ARGUMENTS if trustedUEs or maliciousIds length greater than maximum value</li> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> <li>• INVALID_ARGUMENTS</li> </ul>
----	--------------	---

### 6.5.1.12 using telux::cv2x::UpdateSrcL2InfoCallback = typedef std::function<void (telux::common::ErrorCode error)>

This function is called with the response to updateSrcL2Info.

#### Parameters

in	<i>error</i>	- SUCCESS if Tx reservation change succeeded <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> </ul>
----	--------------	--

### 6.5.1.13 using telux::cv2x::CreateTcpSocketCallback = typedef std::function<void (std::shared\_ptr<ICv2xTxRxSocket> sock, telux::common::ErrorCode error)>

This function is called with the response to createCv2xTcpSocket.

#### Parameters

in	<i>sock</i>	- TCP socket
in	<i>error</i>	- Indicates whether TCP socket creation succeeded <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> </ul>

### 6.5.1.14 using telux::cv2x::CloseTcpSocketCallback = typedef std::function<void (std::shared\_ptr<ICv2xTxRxSocket> sock, telux::common::ErrorCode error)>

This function is called with the response to closeCv2xTcpSocket.

#### Parameters

in	<i>sock</i>	- Closed TCP socket
in	<i>error</i>	- Indicates whether close operation succeeded <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> </ul>

### 6.5.1.15 using telux::cv2x::StartCv2xCallback = typedef std::function<void (telux::common::ErrorCode error)>

This function is called as a response to startCv2x

#### Parameters

in	<i>error</i>	- SUCCESS if Cv2x mode successfully started • SUCCESS • GENERIC_FAILURE
----	--------------	---

### 6.5.1.16 using telux::cv2x::StopCv2xCallback = typedef std::function<void (telux::common::ErrorCode error)>

This function is called as a response to stopCv2x

#### Parameters

in	<i>error</i>	- SUCCESS if Cv2x mode successfully stopped • SUCCESS • GENERIC_FAILURE
----	--------------	---

### 6.5.1.17 using telux::cv2x::RequestCv2xStatusCallback = typedef std::function<void (Cv2xStatus status, telux::common::ErrorCode error)>

This function is called as a response to requestCv2xStatus

#### Parameters

in	<i>status</i>	- Cv2x status
in	<i>error</i>	- SUCCESS if Cv2x status was successfully retrieved • SUCCESS • GENERIC_FAILURE

**Deprecated** use RequestCv2xStatusCallbackEx

### 6.5.1.18 using telux::cv2x::RequestCv2xStatusCallbackEx = typedef std::function<void (Cv2xStatusEx status, telux::common::ErrorCode error)>

This function is called as a response to requestCv2xStatus

**Parameters**

in	<i>status</i>	- Cv2x status
in	<i>error</i>	- SUCCESS if Cv2x status was successfully retrieved • SUCCESS • GENERIC_FAILURE

### 6.5.1.19 using telux::cv2x::UpdateConfigurationCallback = typedef std::function<void (telux::common::ErrorCode error)>

This function is called with the response to updateConfiguration

**Parameters**

in	<i>error</i>	- SUCCESS if configuration was updated successfully • SUCCESS • GENERIC_FAILURE
----	--------------	---

## 6.6 telux::data Namespace Reference

**Namespaces**

- [net](#)

**Data Structures**

- struct [BackhaulInfo](#)
- struct [DataCallEndReason](#)
- union [DataCallEndReason.\\_\\_unnamed\\_\\_](#)
- struct [DataCallStats](#)
- class [DataFactory](#)  
*DataFactory is the central factory to create all data classes. [More...](#)*
- class [DataProfile](#)  
*DataProfile class represents single data profile on the modem. [More...](#)*
- struct [DataRestrictMode](#)
- struct [EspInfo](#)
- struct [FlowDataRate](#)
- struct [IcmpInfo](#)
- class [IDataCall](#)  
*Represents single established data call on the device. [More...](#)*

- class [IDataConnectionListener](#)

- class [IDataConnectionManager](#)

*[IDataConnectionManager](#) is a primary interface for cellular connectivity. This interface provides APIs for start and stop data call connections, get data call information and listener for monitoring data calls. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready. [More...](#)*

- class [IDataCreateProfileCallback](#)

- class [IDataFilterListener](#)

*Listener class for listening to filtering mode notifications, like Data filtering mode change. Client need to implement these methods. The methods in listener can be invoked from multiple threads. So the client needs to make sure that the implementation is thread-safe. [More...](#)*

- class [IDataFilterManager](#)

*[IDataFilterManager](#) class provides interface to enable/disable the data restrict filters and register for data restrict filter. The filtering can be done at any time. One such use case is to do it when we want the AP to suspend so that we are not waking up the AP due to spurious incoming messages. Also to make sure the DataRestrict mode is enabled. [More...](#)*

- class [IDataProfileCallback](#)

- class [IDataProfileListCallback](#)

*Interface for getting list of [DataProfile](#) using callback. Client needs to implement this interface to get single shot responses for commands like get profile list and query profile. [More...](#)*

- class [IDataProfileListener](#)

*Listener class for getting profile change notification. [More...](#)*

- class [IDataProfileManager](#)

- class [IDataSettingsListener](#)

- class [IDataSettingsManager](#)

*The Data Settings Manager class provides an interface to the data subsystem settings. [More...](#)*

- class [IEspFilter](#)

*This class represents a IP Filter for the ESP, get the new instance from [telux::data::DataFactory](#).*

- class [IcmpFilter](#)

*This class represents a IP Filter for the ICMP, get the new instance from [telux::data::DataFactory](#).*

- class [IpfFilter](#)

*A IP filter class to add specific filters like what data will be allowed from the modem to the application processor. Only data packets that match the filter will be sent to the apps processor. Also used to configure Firewall rules.*

- struct [IpAddrInfo](#)

- struct [IpFamilyInfo](#)

- struct [IPv4Info](#)

- struct [IPv6Info](#)



- class [IServingSystemListener](#)

*Listener class for data serving system change notification. [More...](#)*

- class [IServingSystemManager](#)

*Serving System Manager class provides APIs related to the serving system for data functionality. For example, ability to query or be notified about the state of the platform's WWAN PS data serving information. [More...](#)*

- class [ITcpFilter](#)

*This class represents a IP Filter for the TCP, get the new instance from [telux::data::DataFactory](#).*

- class [IUDPFilter](#)

*This class represents a IP Filter for the UDP, get the new instance from [telux::data::DataFactory](#).*

- struct [LatencyConfig](#)

- struct [PortInfo](#)

- struct [ProfileParams](#)

- struct [QosFilterRule](#)

- struct [QosIPFlowInfo](#)

- struct [RoamingStatus](#)

*Roaming Status. [More...](#)*

- struct [ServiceStatus](#)

*Data Service Status Info. [More...](#)*

- struct [TcpInfo](#)

- struct [TftChangeInfo](#)

- struct [TrafficFlowTemplate](#)

- struct [UdpInfo](#)

- struct [VlanConfig](#)

## 6.6.1 Data Structure Documentation

### 6.6.1.1 struct [telux::data::EspInfo](#)

Encapsulating Security Payload

#### Data fields

Type	Field	Description
uint32_t	spi	Security Parameters Index

### 6.6.1.2 struct telux::data::IcmpInfo

Internet Control Message Protocol (ICMP)

#### Data fields

Type	Field	Description
uint8_t	type	ICMP message type - RFC2780
uint8_t	code	ICMP message code - RFC2780

### 6.6.1.3 struct telux::data::IpFamilyInfo

IP Family related Info

#### Data fields

Type	Field	Description
<a href="#">DataCallStatus</a>	status	
<a href="#">IpAddrInfo</a>	addr	

### 6.6.1.4 struct telux::data::IPv4Info

IPv4 header info

#### Data fields

Type	Field	Description
string	srcAddr	address of the device that sends the packet.
string	srcSubnetMask	
string	destAddr	address of receiving end
string	destSubnet↔ Mask	
<a href="#">TypeOfService</a>	value	level of throughput, reliability, and delay
<a href="#">TypeOfService</a>	mask	
<a href="#">IpProtocol</a>	nextProtoId	Protocol ID (i.e TCP, UDP or ICMP )

### 6.6.1.5 struct telux::data::IPv6Info

IPv6 header info

#### Data fields

Type	Field	Description
string	srcAddr	address of the device that sends the packet.
uint8_t	srcPrefixLen	source prefix length used to create subnet
string	destAddr	address of receiving end
uint8_t	dstPrefixLen	destination prefix length used to create subnet
<a href="#">IpProtocol</a>	nextProtoId	Protocol ID (i.e TCP, UDP or ICMP )

Type	Field	Description
<a href="#">TrafficClass</a>	val	indicates the class or priority of the IPv6 packet, enables the ability to track specific traffic flows at the network layer.
<a href="#">TrafficClass</a>	mask	
<a href="#">FlowLabel</a>	flowLabel	Indicates that this packet belongs to a specific sequence of packets between a source and destination, requiring special handling by intermediate IPv6 routers.
uint8_t	natEnabled	

### 6.6.1.6 struct telux::data::QosFilterRule

Encapsulate the Qos Filter rule

#### Data fields

Type	Field	Description
vector< shared_ptr< <a href="#">IIPFilter</a> > >	filter	<a href="#">IIPFilter</a>
uint16_t	filterId	Unique identifier for each filter.
uint16_t	filterPrecedence	Specifies the order in which filters are applied. A lower numerical value has a higher precedence.

### 6.6.1.7 struct telux::data::TcplInfo

TCP header info

#### Data fields

Type	Field	Description
<a href="#">PortInfo</a>	src	Source port and range
<a href="#">PortInfo</a>	dest	Destination port and range

### 6.6.1.8 struct telux::data::TftChangeInfo

QOS TFT flow change info

#### Data fields

Type	Field	Description
shared_ptr< <a href="#">TrafficFlow</a> ↔ <a href="#">Template</a> >	tft	TFT flow info <a href="#">TrafficFlowTemplate</a>
<a href="#">QosFlow</a> ↔ <a href="#">StateChange</a> ↔ <a href="#">Event</a>	stateChange	Flow state change event

### 6.6.1.9 struct telux::data::TrafficFlowTemplate

QOS TFT Flow info

#### Data fields

Type	Field	Description
<a href="#">QosFlowId</a>	qosId	Mandatory defines current flow id
<a href="#">QosFlow↔ StateChange↔ Event</a>	stateChange	Flow state change event
<a href="#">QosFlowMask</a>	mask	bitmask to denote which of the optional fields in <a href="#">TrafficFlowTemplate</a> are valid
<a href="#">QosIPFlowInfo</a>	txGrantedFlow	Optional
<a href="#">QosIPFlowInfo</a>	rxGrantedFlow	
uint32_t	txFiltersLength	
<a href="#">QosFilterRule</a>	txFilters[ <a href="#">MA↔ X_QOS_FIL↔ TERS</a> ]	
uint32_t	rxFiltersLength	
<a href="#">QosFilterRule</a>	rxFilters[ <a href="#">MA↔ X_QOS_FIL↔ TERS</a> ]	

### 6.6.1.10 struct telux::data::UdpInfo

UDP header info

#### Data fields

Type	Field	Description
<a href="#">PortInfo</a>	src	Source port and range
<a href="#">PortInfo</a>	dest	Destination port and range

## 6.6.2 Typedef Documentation

### 6.6.2.1 using telux::data::DataCallResponseCb = typedef std::function<void( const std::shared\_ptr<IDataCall> &dataCall, telux::common::ErrorCode error)>

This function is called with the response to startDataCall / stopDataCall API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

When callback is used with startDataCall, expected behavior is as following:

- If this is first client to start datacall in the system and no error is detected, state of data call will be NET\_CONNECTING and onDataCallInfoChanged will be called once data call is brought up successfully or failed.
- If client tries to start data call that is already up and no error is detected, state of data call will be NET\_CONNECTED and onDataCallInfoChanged will not get called.

- If any client that start data call and error is detected, error argument will contain error code and `onDataCallInfoChanged` will not get called.
- If client tries to start data call on specific profile id and specifies a network interface name to be used the behavior would be as follows:
  - if data call already exists with the same profile ID but a different network interface then error will be set to `telux::ErrorCode::DEVICE_IN_USE` and `onDataCallInfoChanged` will not get called.
  - if data call already exists with the same network interface but different profile ID then error will be set to `telux::ErrorCode::INVALID_OPERATION` and `onDataCallInfoChanged` will not get called.
  - If a data call exists with the same profile id and network interface name, and that data call is currently being torn down, error will be set to `telux::ErrorCode::OP_IN_PROGRESS` and `onDataCallInfoChanged` will not get called. Client can reattempt after the tear down is complete.
- On platforms where a client is not allowed to specify the name of the network interface when starting a data call, the callback will be invoked with error `telux::ErrorCode::REQUEST_NOT_SUPPORTED` and `onDataCallInfoChanged` will not get called.

When callback is used with `stopDataCall`, expected behavior is as following:

- First/Last client that attempts to stop data call and no error is detected, state of data call will be `NET_DISCONNECTING` and `onDataCallInfoChanged` will be called once data call is down.
- If a client starts a data call and then tries to stop it while there are other clients in the system who also started the same data call, and no error is detected, data call status will be `NET_CONNECTED` and `onDataCallInfoChanged` will not get called.
- If any client attempt to stop data call and error detected, error argument will contain error code and `onDataCallInfoChanged` will not get called.

#### Parameters

<i>in</i>	<i>dataCall</i>	Pointer to <a href="#">IDataCall</a>
<i>in</i>	<i>error</i>	Return code for whether the operation succeeded or failed

#### 6.6.2.2 using `telux::data::StatisticsResponseCb = typedef std::function<void(const DataCallStats dataStats, telux::common::ErrorCode error)>`

This function is called with the response to `requestDataCallStatistics` API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

#### Parameters

<i>in</i>	<i>dataStats</i>	Data Call statistics
<i>in</i>	<i>error</i>	Return code for whether the operation succeeded or failed

#### 6.6.2.3 using `telux::data::DataCallListResponseCb = typedef std::function<void(const std::vector<std::shared_ptr<IDataCall>> &dataCallList, telux::common::ErrorCode error)>`

This function is called with the response to `requestDataCallList` API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

### Parameters

in	<i>dataCall</i>	vector of of <a href="#">IDataCall</a> list
in	<i>error</i>	Return code for whether the operation succeeded or failed

#### 6.6.2.4 using telux::data::TrafficFlowTemplateCb = typedef std::function<void(const std::vector<std::shared\_ptr<TrafficFlowTemplate>> &tft, telux::common::← ErrorCode error)>

This function is called in the response to requestTrafficFlowTemplate().

### Parameters

in	<i>tft</i>	Vector of TFT flow info. <a href="#">TrafficFlowTemplate</a>
in	<i>error</i>	Code which indicates whether the operation succeeded or not. ErrorCode.

#### 6.6.2.5 using telux::data::DataRestrictModeCb = typedef std::function<void(Data← RestrictMode mode, telux::common::ErrorCode error)>

This function is called in the response to requestDataRestrictMode().

### Parameters

in	<i>mode</i>	Return current data restrict mode.
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. ErrorCode.

#### 6.6.2.6 using telux::data::TypeOfService = typedef uint8\_t

#### 6.6.2.7 using telux::data::TrafficClass = typedef uint8\_t

#### 6.6.2.8 using telux::data::FlowLabel = typedef uint32\_t

## 6.7 telux::data::net Namespace Reference

### Data Structures

- struct [BridgeInfo](#)
- struct [DmzConfig](#)
- struct [FirewallConfig](#)
- struct [FirewallEntryInfo](#)
- class [IBridgeListener](#)

- class [IBridgeManager](#)

*IBridgeManager* provides APIs to enable/disable and set/get/delete software bridges for various WLAN and Ethernet interfaces. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready. [More...](#)

- class [IFirewallEntry](#)

*Firewall entry class is used for configuring firewall rules. [More...](#)*

- class [IFirewallListener](#)

- class [IFirewallManager](#)

*FirewallManager* is a primary interface that filters and controls the network traffic on a pre-configured set of rules. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready. [More...](#)

- class [IL2tpListener](#)

- class [IL2tpManager](#)

*L2tpManager* is a primary interface for configuring L2TP Service. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready. [More...](#)

- class [INatListener](#)

- class [INatManager](#)

*NatManager* is a primary interface for configuring static network address translation(SNAT) and DMZ (demilitarized zone). It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready. [More...](#)

- class [ISocksListener](#)

- class [ISocksManager](#)

*SocksManager* is a primary interface for configuring legacy Socks proxy server. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready. [More...](#)

- class [IVlanListener](#)

- class [IVlanManager](#)

*VlanManager* is a primary interface for configuring VLAN (Virtual Local Area Network). it provide APIs for create, query, remove VLAN interfaces and associate or disassociate with profile IDs. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready. [More...](#)

- struct [L2tpSessionConfig](#)

- struct [L2tpSysConfig](#)

- struct [L2tpTunnelConfig](#)

- struct [NatConfig](#)

- struct [VlanBindConfig](#)

## 6.7.1 Data Structure Documentation

### 6.7.1.1 struct telux::data::net::BridgeInfo

Structure to configure a software bridge for an interface

#### Data fields

Type	Field	Description
string	ifaceName	Interface name
<a href="#">BridgeFace</a> ↔ Type	ifaceType	Interface type
uint32_t	bandwidth	Bandwidth(in Mbps) required for software bridge

### 6.7.1.2 struct telux::data::net::DmzConfig

DMZ configuration parameters

#### Data fields

Type	Field	Description
<a href="#">BackhaulInfo</a>	bhInfo	Backhaul Information to apply firewal settings on
string	ipAddr	IP address for which DMZ will be enabled

### 6.7.1.3 struct telux::data::net::FirewallConfig

Firewall configuration parameters

#### Data fields

Type	Field	Description
<a href="#">BackhaulInfo</a>	bhInfo	Backhaul Information to apply firewal settings on
bool	enable	True: Firewall enabled. False: Firewall disabled
bool	allowPackets	True: Packets that match rules will be allowed. False: Packets that match rules will be dropped

### 6.7.1.4 struct telux::data::net::FirewallEntryInfo

Firewall rules parameters

#### Data fields

Type	Field	Description
shared_ptr< <a href="#">I</a> ↔ <a href="#">FirewallEntry</a> >	fwEntry	Shared pointer to firewall rules for the backhaul
<a href="#">BackhaulInfo</a>	bhInfo	Backhaul Information to add firewal rules on



### 6.7.1.5 struct telux::data::net::VlanBindConfig

#### Data fields

Type	Field	Description
int	vlanId	
<a href="#">BackhaulInfo</a>	bhInfo	VLAN ID to be bound to the specified backhaul Configuration of Backhaul to bind VLAN to

## 6.7.2 Typedef Documentation

### 6.7.2.1 using telux::data::net::BridgeInfoResponseCb = typedef std::function<void(const std::vector<BridgeInfo> &infos, telux::common::ErrorCode error)>

This function is called as a response to requestBridgeInfo

#### Parameters

in	<i>infos</i>	List of the software bridge configurations in the system
in	<i>error</i>	Return code which indicates whether the operation is succeeded or not <a href="#">telux::common::ErrorCode</a>

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 6.7.2.2 using telux::data::net::FirewallConfigCb = typedef std::function<void(Firewall↔ Config status, telux::common::ErrorCode error)>

This function is called as a response to requestFirewallConfig()

#### Parameters

in	<i>config</i>	Firewall configuration status for specific backhaul <a href="#">telux::data::FirewallConfig</a> .
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. <a href="#">telux::common::ErrorCode</a>

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 6.7.2.3 using telux::data::net::FirewallEntryInfoCb = typedef std::function<void(std::vector<FirewallEntryInfo> entry, telux::common::ErrorCode error)>

This function is called as a response to requestFirewallEntries()

in	<i>entries</i>	Vector of firewall entries for specific backhaul
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. <a href="#">telux::common::ErrorCode</a>

#### 6.7.2.4 using `telux::data::net::DmzEntryInfoCb = typedef std::function<void(std::vector<DmzConfig> dmzEntries, telux::common::ErrorCode error)>`

This function is called as a response to `requestDmzEntries()`

##### Parameters

in	<i>dmzEntries</i>	list of dmz entries configurations
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. <a href="#">telux::common::ErrorCode</a>

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 6.7.2.5 using `telux::data::net::FirewallStatusCb = typedef std::function<void(bool enable, bool allowPackets, telux::common::ErrorCode error)>`

This function is called as a response to `requestFirewallStatus()`

##### Parameters

in	<i>enable</i>	Indicates whether the firewall is enabled
in	<i>allowPackets</i>	Indicates whether to accept or drop packets matching the rules
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. <a href="#">telux::common::ErrorCode</a>

**Deprecated** `telux::data::FirewallConfigCb` callback is used to get Firewall configuration

#### 6.7.2.6 using `telux::data::net::FirewallEntriesCb = typedef std::function<void(std::vector<std::shared_ptr<IFirewallEntry>> entries, telux::common::ErrorCode error)>`

This function is called as a response to `requestFirewallEntries()`

##### Parameters

in	<i>entries</i>	list of firewall entries
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. <a href="#">telux::common::ErrorCode</a>

**Deprecated** `telux::data::FirewallEntryInfoCb` callback is used to get Firewall entries

### 6.7.2.7 using `telux::data::net::DmzEntriesCb = typedef std::function<void(std::vector<std::string> dmzEntries, telux::common::ErrorCode error)>`

This function is called as a response to `requestDmzEntries()`

#### Parameters

in	<i>dmzEntries</i>	list of dmz entries
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. <a href="#">telux::common::ErrorCode</a>

**Deprecated** `telux::data::DmzEntryInfoCb` callback is used to get DMZ entries

### 6.7.2.8 using `telux::data::net::L2tpConfigCb = typedef std::function<void(const L2tpSysConfig &l2tpSysConfig, telux::common::ErrorCode error)>`

This function is called as a response to `requestConfig()`

#### Parameters

in	<i>l2tpSysConfig</i>	Current L2TP configuration
in	<i>error</i>	Return code which indicates whether the operation succeeded or not <a href="#">telux::common::ErrorCode</a>

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 6.7.2.9 using `telux::data::net::StaticNatEntriesCb = typedef std::function<void(const std::vector<NatConfig> &snatEntries, telux::common::ErrorCode error)>`

This function is called as a response to `requestStaticNatEntries()`

#### Parameters

in	<i>snatEntries</i>	list of static Network Address Translation (NAT)
in	<i>error</i>	Return code which indicates whether the operation succeeded or not <a href="#">telux::common::ErrorCode</a>

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 6.7.2.10 using telux::data::net::CreateVlanCb = typedef std::function<void(bool isAccelerated, telux::common::ErrorCode error)>

This function is called as a response to createVlan()

in	<i>isAccelerated</i>	Offload status returned by server
in	<i>error</i>	Return code which indicates whether the operation succeeded or not <a href="#">telux::common::ErrorCode</a>

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 6.7.2.11 using telux::data::net::QueryVlanResponseCb = typedef std::function<void(const std::vector<VlanConfig> &configs, telux::common::ErrorCode error)>

This function is called as a response to queryVlanInfo()

#### Parameters

in	<i>configs</i>	List of VLAN configs
in	<i>error</i>	Return code which indicates whether the operation succeeded or not <a href="#">telux::common::ErrorCode</a>

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 6.7.2.12 using telux::data::net::VlanMappingResponseCb = typedef std::function<void( const std::list<std::pair<int, int>> &mapping, telux::common::ErrorCode error)>

This function is called as a response to queryVlanMappingList()

#### Parameters

in	<i>mapping</i>	List of profile Id and Vlan id map Key is Profile Id and value is VLAN id
in	<i>error</i>	Return code which indicates whether the operation succeeded or not <a href="#">telux::common::ErrorCode</a>

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

## 6.7.3 Enumeration Type Documentation

### 6.7.3.1 enum telux::data::net::BridgeFaceType [strong]

Interface types supported for bridge configuration

#### Enumerator

**UNKNOWN**

**WLAN\_AP** Wireless Local Area Network (WLAN) in AP mode

**WLAN\_STA** Wireless Local Area Network (WLAN) in STA mode

**ETH** Ethernet (ETH)

## 6.8 telux::loc Namespace Reference

### Data Structures

- struct [BodyToSensorMountParams](#)
- struct [DREngineConfiguration](#)
- struct [ExternalLocationInfo](#)
- struct [GlonassTimeInfo](#)
- struct [GnssData](#)
- struct [GnssEnergyConsumedInfo](#)
- struct [GnssKinematicsData](#)
- struct [GnssMeasurementInfo](#)
- struct [GnssMeasurements](#)
- struct [GnssMeasurementsClock](#)
- struct [GnssMeasurementsData](#)
- class [IDgnssManager](#)

*IRtcManager provides interface to inject RTCM data into modem, register event listener reported by cdfw(correction data framework). [More...](#)*
- class [IDgnssStatusListener](#)

*Listener class for getting RTCM injection event notification information. [More...](#)*
- class [IGnssSignalInfo](#)

*IGnssSignalInfo provides interface to retrieve GNSS data information like jammer metrics and automatic gain control for satellite signal type. [More...](#)*
- class [IGnssSVInfo](#)

*IGnssSVInfo provides interface to retrieve the list of SV info available and whether altitude is assumed or calculated. [More...](#)*
- class [ILocationConfigListener](#)

*ILocationConfigListener* interface is used to receive notifications related to configuration events. [More...](#)

- class [ILocationConfigurator](#)

*ILocationConfigurator* allows general engine configurations (example: TUNC, PACE etc), configuration of specific engines like SPE (example: minSVElevation, minGPSWeek etc) or DRE, deletion of warm and cold aiding data, NMEA configuration and support for XTRA feature. *ILocationConfigurator* APIs strictly adheres to the principle of single client per process. [More...](#)

- class [ILocationInfoBase](#)

*ILocationInfoBase* provides interface to get basic position related information like latitude, longitude, altitude, timestamp. [More...](#)

- class [ILocationInfoEx](#)

*ILocationInfoEx* provides interface to get richer position related information like latitude, longitude, altitude and other information like time stamp, session status, dop, reliabilities, uncertainties etc. [More...](#)

- class [ILocationInjectionListener](#)

- class [ILocationListener](#)

*Listener* class for getting location updates and satellite vehicle information. [More...](#)

- class [ILocationManager](#)

*ILocationManager* provides interface to register and remove listeners. It also allows to set and get configuration/ criteria for position reports. The new APIs(*registerListenerEx*, *deRegisterListenerEx*, *startDetailedReports*, *startBasicReports*) and old/deprecated APIs(*registerListener*, *removeListener*, *setPositionReportTimeout*, *setHorizontalAccuracyLevel*, *setMinIntervalForReports*) should not be used interchangeably, either the new APIs should be used or the old APIs should be used. [More...](#)

- class [ILocationSystemInfoListener](#)

- class [ISVInfo](#)

*ISVInfo* provides interface to retrieve information about Satellite Vehicles, their position and health status. [More...](#)

- struct [LeapSecondChangeInfo](#)

- struct [LeapSecondInfo](#)

- struct [LeverArmParams](#)

- struct [LLAInfo](#)

- class [LocationFactory](#)

*LocationFactory* allows creation of location manager. [More...](#)

- struct [LocationSystemInfo](#)

- struct [NmeaConfig](#)

- struct [RobustLocationConfiguration](#)

- struct [RobustLocationVersion](#)

- struct [SvBlackListInfo](#)

- struct [SvUsedInPosition](#)

- struct [SystemTime](#)

- union [SystemTimeInfo](#)
- struct [TimeInfo](#)
- struct [XtraConfig](#)
- struct [XtraStatus](#)

## 6.9 telux::platform Namespace Reference

### Data Structures

- class [IDeviceInfoListener](#)

*Listener class for getting device info related notifications . The client needs to implement these methods as briefly as possible and avoid blocking calls in it. The methods in this class can be invoked from multiple different threads. Client needs to make sure that the implementation is thread-safe. [More...](#)*

- class [IDeviceInfoManager](#)

*IDeviceInfoManager provides interface to to retrieve IMEI and platform version operations. [More...](#)*

- class [PlatformFactory](#)

*PlatformFactory allows creation of Platform services related classes. [More...](#)*

- struct [PlatformVersion](#)

## 6.10 telux::power Namespace Reference

### Data Structures

- class [ITcuActivityListener](#)

*Listener class for getting notifications related to TCU-activity state and also the updates related to TCU-activity service status. The client needs to implement these methods as briefly as possible and avoid blocking calls in it. The methods in this class can be invoked from multiple different threads. Client needs to make sure that the implementation is thread-safe. [More...](#)*

- class [ITcuActivityManager](#)

*ITcuActivityManager provides interface to register and de-register listeners (to get TCU-activity state updates). And also API to initiate TCU-activity state transition. [More...](#)*

- class [PowerFactory](#)

*PowerFactory allows creation of TCU-activity manager instance. [More...](#)*

## 6.11 telux::tel Namespace Reference

## Data Structures

- struct [CardReaderStatus](#)
- class [CdmaCellIdentity](#)
- class [CdmaCellInfo](#)
- class [CdmaSignalStrengthInfo](#)
- struct [CellBroadcastFilter](#)
- class [CellBroadcastMessage](#)  
*Cell Broadcast message.*
- class [CellInfo](#)
- struct [CellularCapabilityInfo](#)
- class [Circle](#)
- class [CmasInfo](#)
- struct [DcStatus](#)
- struct [EcallConfig](#)
- struct [EcallHlapTimerEvents](#)
- struct [EcallHlapTimerStatus](#)
- struct [EcallModeInfo](#)
- struct [EcallMsdControlBits](#)
- struct [EcallMsdData](#)
- struct [EcallMsdOptionals](#)
- struct [EcallOptionalPdu](#)
- struct [EcallVehicleIdentificationNumber](#)
- struct [EcallVehicleLocation](#)
- struct [EcallVehicleLocationDelta](#)
- struct [EcallVehiclePropulsionStorageType](#)
- class [EtwInfo](#)
- class [Geometry](#)
- class [GsmCellIdentity](#)
- class [GsmCellInfo](#)
- class [GsmSignalStrengthInfo](#)
- class [IAtrResponseCallback](#)
- class [ICall](#)

*ICall* represents a call in progress. An *ICall* cannot be directly created by the client, rather it is returned as



a result of instantiating a call or from the PhoneListener when receiving an incoming call. [More...](#)

- class [ICallListener](#)

A listener class for monitoring changes in call, including call state change and ECall state change. Override the methods for the state that you wish to receive updates for. [More...](#)

- class [ICallManager](#)

Call Manager is the primary interface for call related operations Allows to conference calls, swap calls, make normal voice call and emergency call, send and update MSD pdu. [More...](#)

- class [ICard](#)

*ICard* represents currently inserted UICC or eUICC. [More...](#)

- class [ICardApp](#)

Represents a single card application. [More...](#)

- class [ICardChannelCallback](#)

- class [ICardCommandCallback](#)

- class [ICardListener](#)

- class [ICardManager](#)

- class [ICardReaderCallback](#)

- struct [IccResult](#)

- class [ICellBroadcastListener](#)

A listener class which monitors cell broadcast messages.

- class [ICellBroadcastManager](#)

*CellBroadcastManager* class is primary interface to configure and activate emergency broadcast messages and receive broadcast messages.

- class [ICellularCapabilityCallback](#)

- class [IEcallListener](#)

Listener class to notify service status change notifications. The listener method can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe. [More...](#)

- class [IEcallManager](#)

*IEcallManager* allows operations related to automotive emergency call management and its related configurations. [More...](#)

- class [IimsServingSystemListener](#)

A listener class for monitoring changes in IMS Serving System manager, including IMS registration status change. Override the methods for the state that you wish to receive updates for.

- class [IimsServingSystemManager](#)

*IMS Serving System Manager* is the primary interface for IMS related operations Allows to query IMS registration status.

- class [IimsSettingsListener](#)

Listener class for getting IMS service configuration change notifications. The listener method can be

*invoked from multiple different threads. Client needs to make sure that implementation is thread-safe.*  
[More...](#)

- class [ImsSettingsManager](#)

*ImsSettingsManager allows IMS settings. For example enabling or disabling IMS service, VOIMS service.*  
[More...](#)

- class [IMakeCallCallback](#)

*Interface for Make Call callback object. Client needs to implement this interface to get single shot responses for commands like make call.* [More...](#)

- struct [ImsRegistrationInfo](#)

- struct [ImsServiceConfig](#)

- class [IMultiSimListener](#)

*Listener class for getting high capability change notification. The listener method can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe.* [More...](#)

- class [IMultiSimManager](#)

*MultiSimManager allows to perform operation pertaining to devices which have more than one SIM/UICC card. Clients should check if the subsystem is ready before invoking any of the APIs as follows.* [More...](#)

- class [INetworkSelectionListener](#)

*Listener class for getting network selection mode change notification.* [More...](#)

- class [INetworkSelectionManager](#)

*Network Selection Manager class provides the interface to get and set network selection mode, preferred network list and scan available networks.* [More...](#)

- class [IOperatingModeCallback](#)

- class [IPhone](#)

*This class allows getting system information and registering for system events. Each Phone instance is associated with a single SIM. So on a dual SIM device you would have 2 Phone instances.* [More...](#)

- class [IPhoneListener](#)

*A listener class for monitoring changes in specific telephony states on the device, including service state and signal strength. Override the methods for the state that you wish to receive updates for.* [More...](#)

- class [IPhoneManager](#)

*Phone Manager creates one or more phones based on SIM slot count, it allows clients to register for notification of system events. Clients should check if the subsystem is ready before invoking any of the APIs.*  
[More...](#)

- class [IRemoteSimListener](#)

*A listener class for getting remote SIM notifications.* [More...](#)

- class [IRemoteSimManager](#)

*IRemoteSimManager provides APIs for remote SIM related operations. This allows a device to use a SIM card on another device for its WWAN modem functionality. The SIM provider service is the endpoint that interfaces with the SIM card (e.g. over bluetooth) and sends/receives data to the other endpoint, the modem. The modem sends requests to the SIM provider service to interact with the SIM card (e.g. power up, transmit APDU, etc.), and is notified of events (e.g. card errors, resets, etc.). This API is used by the*

*SIM provider endpoint to provide a SIM card to the modem. [More...](#)*

- class [ISapCardCommandCallback](#)
- class [ISapCardListener](#)
- class [ISapCardManager](#)

*[ISapCardManager](#) provide APIs for SAP related operations. [More...](#)*

- class [IServingSystemListener](#)

*Listener class for getting radio access technology mode preference change notification. [More...](#)*

- class [IServingSystemManager](#)
- class [ISignalStrengthCallback](#)

*Interface for Signal strength callback object. Client needs to implement this interface to get single shot responses for commands like get signal strength. [More...](#)*

- class [ISimProfileListener](#)

*The interface listens for profile download indication and keep track of download and install progress of profile. [More...](#)*

- class [ISimProfileManager](#)

*[ISimProfileManager](#) is a primary interface for remote eUICCs (eSIMs or embedded SIMs) provisioning. This interface provides APIs to add, delete, set profile, update nickname, provide user consent, get Eid on the eUICC. [More...](#)*

- class [ISmscAddressCallback](#)
- class [ISmsListener](#)

*A listener class receives notification for the incoming message(s) and delivery report for sent message(s). [More...](#)*

- class [ISmsManager](#)

*SmsManager class is the primary interface to manage SMS operations such as send and receive an SMS text and raw encoded PDU(s). This class handles single part and multi-part messages. [More...](#)*

- class [ISubscription](#)

*Subscription returns information about network operator subscription details pertaining to a SIM card. [More...](#)*

- class [ISubscriptionListener](#)

*A listener class for receiving device subscription information. The methods in listener can be invoked from multiple different threads. The implementation should be thread safe. [More...](#)*

- class [ISubscriptionManager](#)

- class [IVoiceServiceStateCallback](#)

*Interface for voice service state callback object. Client needs to implement this interface to get single shot responses for commands like request voice radio technology. [More...](#)*

- class [LteCellIdentity](#)
- class [LteCellInfo](#)
- class [LteSignalStrengthInfo](#)

- struct [MessageAttributes](#)

*Contains structure of message attributes like encoding type, number of segments, characters left in last segment. [More...](#)*
- struct [MessagePartInfo](#)

*Structure containing information about the part of multi-part SMS such as concatenated message reference number, number of segments and segment number. During concatenation this information along with originating address helps in associating each part of the multi-part message to the corresponding multi-part message. [More...](#)*
- struct [NetworkTimeInfo](#)
- class [Nr5gSignalStrengthInfo](#)
- class [OperatorInfo](#)
- struct [OperatorStatus](#)
- class [PhoneFactory](#)

*[PhoneFactory](#) is the central factory to create all Telephony SDK Classes and services. [More...](#)*
- struct [Point](#)
- class [Polygon](#)
- struct [PreferredNetworkInfo](#)
- class [SignalStrength](#)
- class [SimProfile](#)

*[SimProfile](#) class represents single eUICC profile on the card. [More...](#)*
- struct [SimRatCapability](#)
- struct [SlotStatus](#)
- class [SmsMessage](#)

*Data structure represents an incoming SMS. This is applicable for single part message or part of the multipart message. [More...](#)*
- class [TdsdmaCellIdentity](#)
- class [TdsdmaCellInfo](#)
- class [TdsdmaSignalStrengthInfo](#)
- class [VoiceServiceInfo](#)
- class [WarningAreaInfo](#)
- class [WcdmaCellIdentity](#)
- class [WcdmaCellInfo](#)
- class [WcdmaSignalStrengthInfo](#)

## 6.11.1 Data Structure Documentation

### 6.11.1.1 struct telux::tel::ImsRegistrationInfo

Defines the IMS registration status parameters and the error code value

#### Data fields

Type	Field	Description
<a href="#">Registration↔ Status</a>	imsRegStatus	The status of the IMS registration with the network
int	errorCode	An error code is returned when the IMS registration status is <a href="#">RegistrationStatus::NOT_REGISTERED</a> . Values(Defined in SIP-RFC3261 section 13.2.2.2 and section 13.2.2.3): <ul style="list-style-type: none"> <li>• 3xx – Redirection responses</li> <li>• 4xx – Client failure responses</li> <li>• 5xx – Server failure responses</li> <li>• 6xx – Global failure responses</li> </ul>
string	errorString	Registration failure error string when the IMS is not registered.

### 6.11.1.2 struct telux::tel::NetworkTimeInfo

Defines Network time information

#### Data fields

Type	Field	Description
uint16_t	year	Year.
uint8_t	month	Month. 1 is January and 12 is December.
uint8_t	day	Day. Range: 1 to 31.
uint8_t	hour	Hour. Range: 0 to 23.
uint8_t	minute	Minute. Range: 0 to 59.
uint8_t	second	Second. Range: 0 to 59.
uint8_t	dayOfWeek	Day of the week. 0 is Monday and 6 is Sunday.
int8_t	timeZone	Offset between UTC and local time in units of 15 minutes (signed value). Actual value = field value * 15 minutes.
uint8_t	dstAdj	Daylight saving adjustment in hours to obtain local time. Possible values: 0, 1, and 2.
string	nitzTime	Network Identity and Time Zone(NITZ) information in the form "yyyy/mm/dd,hh:mm:ss(+/-)tzh:tzm,dt

### 6.11.1.3 struct telux::tel::Point

[Point](#) represented by latitude and longitude.

**Data fields**

Type	Field	Description
double	latitude	
double	longitude	

**6.11.2 Typedef Documentation****6.11.2.1 using telux::tel::MakeCallCallback = typedef std::function<void(telux↔::common::ErrorCode error, std::shared\_ptr<ICall> call)>**

This function is called with the response to make normal call and emergency call.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

**Parameters**

out	<i>error</i>	ErrorCode
out	<i>call</i>	Pointer to Call object or nullptr in case of failure

**6.11.2.2 using telux::tel::ECallHlapTimerStatusCallback = typedef std::function<void(telux↔::common::ErrorCode error, int phoneId, ECallHlapTimerStatus timers↔Status)>**

This function is called with response to request for eCall High Level Application Protocol(HLAP) timers status.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

**Parameters**

out	<i>error</i>	ErrorCode
out	<i>phoneId</i>	Represents the phone corresponding to which the response is being reported.
out	<i>timersStatus</i>	<a href="#">ECallHlapTimerStatus</a>

**6.11.2.3 using telux::tel::PinOperationResponseCb = typedef std::function<void(int retryCount, telux::common::ErrorCode error)>**

This function is called with the response to pin operations like change pin password, unlock card and set card lock.

**Parameters**

in	<i>retryCount</i>	No of retry attempts left
in	<i>error</i>	Return code for whether the operation succeeded or failed

#### 6.11.2.4 using telux::tel::QueryFdnLockResponseCb = typedef std::function<void(bool isAvailable, bool isEnabled, telux::common::ErrorCode error)>

This function is called with the response to queryFdnLockState API.

##### Parameters

in	<i>isAvailable</i>	Determine FDN lock state availability
in	<i>isEnabled</i>	Determine FDN lock state i.e enable or disable
in	<i>error</i>	Return code for whether the operation succeeded or failed

#### 6.11.2.5 using telux::tel::QueryPin1LockResponseCb = typedef std::function<void(bool state, telux::common::ErrorCode error)>

This function is called with the response to queryPin1LockState API.

##### Parameters

in	<i>state</i>	Determine state whether enabled or disabled
in	<i>error</i>	Return code for whether the operation succeeded or failed

#### 6.11.2.6 using telux::tel::EidResponseCallback = typedef std::function<void(const std::string &eid, telux::common::ErrorCode error)>

This function is called with the response to requestEid API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

##### Parameters

in	<i>eid</i>	eUICC identifier.
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. ErrorCode

#### 6.11.2.7 using telux::tel::RequestFiltersResponseCallback = typedef std::function<void(std::vector<CellBroadcastFilter> filters, telux::common::ErrorCode error)>

This function is called with the response to requestMessageIdFilters API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

##### Parameters

in	<i>filters</i>	List of broadcast filters <a href="#">CellBroadcastFilter</a>
in	<i>error</i>	Return code which indicates whether the operation succeeded or not ErrorCode

### 6.11.2.8 using `telux::tel::RequestActivationStatusResponseCallback = typedef std::function<void(bool isActivated, telux::common::ErrorCode error)>`

This function is called with the response to requestActivationStatus API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

#### Parameters

in	<i>isActivated</i>	If true then filters are activated else filters are deactivated
in	<i>error</i>	Return code which indicates whether the operation succeeded or not ErrorCode

### 6.11.2.9 using `telux::tel::ImsRegistrationInfoCb = typedef std::function<void(ImsRegistrationInfo status, telux::common::ErrorCode error)>`

This function is called in the response to requestRegistrationInfo API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

#### Parameters

in	<i>status</i>	Indicates the IMS registration status and the error code <a href="#">telux::tel::ImsRegistrationInfo</a> .
in	<i>error</i>	Return code which indicates whether the operation succeeded or not <a href="#">telux::common::ErrorCode</a> .

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 6.11.2.10 using `telux::tel::ImsServiceConfigValidity = typedef std::bitset<32>`

32 bit mask that denotes which of the IMS settings configuration parameters defined in ImsServiceConfigType enum are configured currently. For example, if the client selects the VOIMS configuration, 1st bit position is set.

### 6.11.2.11 using `telux::tel::ImsServiceConfigCb = typedef std::function<void(SlotId slotId, ImsServiceConfig config, telux::common::ErrorCode error)>`

This function is called in the response to requestServiceConfig API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

#### Parameters

in	<i>slotId</i>	Slot for which the IMS service configuration is intended.
in	<i>config</i>	Indicates which configuration is valid and whether the configuration is enabled or disabled. <a href="#">ImsServiceConfig</a> .



in	<i>error</i>	Return code which indicates whether the operation succeeded or not .
----	--------------	--

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 6.11.2.12 using `telux::tel::HighCapabilityCallback = typedef std::function<void(int slotId, telux::common::ErrorCode error)>`

This function is called in the response to requestHighCapability API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

**Parameters**

in	<i>slotId</i>	SIM corresponding to slot identifier has high capability.
in	<i>error</i>	Return code which indicates whether the operation succeeded or not ErrorCode

### 6.11.2.13 using `telux::tel::SlotStatusCallback = typedef std::function<void(std::map<SlotId, SlotStatus> slotStatus, telux::common::ErrorCode error)>`

This function is called in response to requestSlotStatus API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

**Parameters**

in	<i>slotStatus</i>	list of slots status <a href="#">SlotStatus</a>
in	<i>error</i>	Return code which indicates whether the operation succeeded or not ErrorCode

### 6.11.2.14 using `telux::tel::RatMask = typedef std::bitset<16>`

16 bit mask that denotes which of the radio access technologies defined in RatType enum are used for preferred networks.

### 6.11.2.15 using `telux::tel::SelectionModeResponseCallback = typedef std::function<void(NetworkSelectionMode mode, telux::common::ErrorCode error)>`

This function is called with the response to requestNetworkSelectionMode API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

in	<i>mode</i>	<a href="#">NetworkSelectionMode</a>
in	<i>error</i>	Return code which indicates whether the operation succeeded or not <code>ErrorCode</code>

**6.11.2.16** using `telux::tel::PreferredNetworksCallback = typedef std::function<void(std::vector<PreferredNetworkInfo> info, std::vector<PreferredNetworkInfo> staticInfo, telux::common::ErrorCode error)>`

This function is called with the response to requestPreferredNetworks API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

#### Parameters

in	<i>info</i>	3GPP preferred networks list i.e PLMN list.
in	<i>staticInfo</i>	Static 3GPP preferred networks list i.e OPLMN list.
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. <code>ErrorCode</code>

**6.11.2.17** using `telux::tel::NetworkScanCallback = typedef std::function<void(std::vector<OperatorInfo> operatorInfos, telux::common::ErrorCode error)>`

This function is called with the response to performNetworkScan API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

#### Parameters

in	<i>operatorInfos</i>	Operators info with details of network operator name, MCC, MNC and status.
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. <code>ErrorCode</code>

**6.11.2.18** using `telux::tel::VoiceRadioTechResponseCb = typedef std::function<void(telux::tel::RadioTechnology radioTech, telux::common::ErrorCode error)>`

This function is called with the response to requestVoiceRadioTechnology API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

#### Parameters

in	<i>radioTech</i>	Pointer to radio technology
in	<i>error</i>	Return code for whether the operation succeeded or failed <ul style="list-style-type: none"> <li>• <a href="#">telux::common::ErrorCode::SUCCESS</a></li> <li>• <a href="#">telux::common::ErrorCode::RADIO_NOT_AVAILABLE</a></li> <li>• <a href="#">telux::common::ErrorCode::GENERIC_FAILURE</a></li> </ul>

**6.11.2.19** using `telux::tel::CellInfoCallback = typedef std::function<void(std::vector<std::shared_ptr<CellInfo>> cellInfoList, telux::common::ErrorCode error)>`

This function is called with the response to requestCellInfo API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

#### Parameters

out	<i>cellInfoList</i>	vector of shared pointers to cell info object
out	<i>error</i>	Return code for whether the operation succeeded or failed

**6.11.2.20** using `telux::tel::ECallGetOperatingModeCallback = typedef std::function<void(ECallMode eCallMode, telux::common::ErrorCode error)>`

This function is called with the response to requestECallOperatingMode API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

#### Parameters

out	<i>eCallMode</i>	<a href="#">ECallMode</a>
out	<i>error</i>	Return code for whether the operation succeeded or failed

**6.11.2.21** using `telux::tel::RATCapabilitiesMask = typedef std::bitset<16>`

**6.11.2.22** using `telux::tel::VoiceServiceTechnologiesMask = typedef std::bitset<16>`

**6.11.2.23** using `telux::tel::SapStateResponseCallback = typedef std::function<void(SapState sapState, telux::common::ErrorCode error)>`

This function is called with the response to requestSapState API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

#### Parameters

in	<i>sapState</i>	<a href="#">SapState</a> of SIM access profile (SAP) connection
in	<i>error</i>	Return code for whether the operation succeeded or failed

**6.11.2.24** using `telux::tel::RatPreference = typedef std::bitset<16>`

16 bit mask that denotes which of the radio access technology mode preference defined in RatPrefType enum are used to set or get RAT preference.

### 6.11.2.25 using telux::tel::RatPreferenceCallback = typedef std::function<void(RatPreference preference, telux::common::ErrorCode error)>

This function is called with the response to requestRatPreference API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

#### Parameters

in	<i>preference</i>	<a href="#">RatPreference</a>
in	<i>error</i>	Return code which indicates whether the operation succeeded or not ErrorCode

### 6.11.2.26 using telux::tel::ServiceDomainPreferenceCallback = typedef std::function<void(ServiceDomainPreference preference, telux::common::ErrorCode error)>

This function is called with the response to requestServiceDomainPreference API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

#### Parameters

in	<i>preference</i>	<a href="#">ServiceDomainPreference</a>
in	<i>error</i>	Return code which indicates whether the operation succeeded or not ErrorCode

### 6.11.2.27 using telux::tel::ProfileListResponseCb = typedef std::function<void( const std::vector<std::shared\_ptr<SimProfile>> &profiles, telux::common::ErrorCode error)>

This function is called with the response to requestProfileList API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

#### Parameters

in	<i>info</i>	Profiles information <a href="#">SimProfile</a> .
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. ErrorCode.

### 6.11.2.28 using telux::tel::EidResponseCb = typedef std::function<void(std::string eid, telux::common::ErrorCode error)>

This function is called with the response to requestEid API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

in	<i>eid</i>	eUICC identifier.
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. ErrorCode.

## 6.11.3 Enumeration Type Documentation

### 6.11.3.1 enum telux::tel::RegistrationStatus [strong]

Defines the IMS registration status parameters

#### Enumerator

**UNKOWN\_STATE** Unknown status for IMS  
**NOT\_REGISTERED** Not registered status for IMS  
**REGISTERING** Registering status for IMS  
**REGISTERED** Registered status for IMS  
**LIMITED\_REGISTERED** Limited registration status for IMS

## 6.12 telux::therm Namespace Reference

### Data Structures

- struct [BoundCoolingDevice](#)
- class [ICoolingDevice](#)  
*ICoolingDevice* provides interface to get type of the cooling device, the maximum throttle state and the currently requested throttle state of the cooling device. [More...](#)
- class [IThermalManager](#)  
*IThermalManager* provides interface to get thermal zone and cooling device information. [More...](#)
- class [IThermalShutdownListener](#)  
*Listener* class for getting notifications when automatic thermal shutdown mode is enabled/ disabled or will be enabled imminently. The client needs to implement these methods as briefly as possible and avoid blocking calls in it. The methods in this class can be invoked from multiple different threads. Client needs to make sure that the implementation is thread-safe. [More...](#)
- class [IThermalShutdownManager](#)  
*IThermalShutdownManager* class provides interface to enable/disable automatic thermal shutdown. Additionally it facilitates to register for notifications when the automatic shutdown mode changes. [More...](#)
- class [IThermalZone](#)  
*IThermalZone* provides interface to get type of the sensor, the current temperature reading, trip points and the cooling devices binded etc. [More...](#)
- class [ITripPoint](#)  
*ITripPoint* provides interface to get trip point type, trip point temperature and hysteresis value for that trip point. [More...](#)
- class [ThermalFactory](#)

*ThermalFactory* allows creation of thermal manager. [More...](#)

## 6.13 telux::wlan Namespace Reference

### Data Structures

- struct [ApConfig](#)
- struct [ApInfo](#)
- struct [ApNetConfig](#)
- struct [ApNetInfo](#)
- struct [ApStatus](#)
- struct [DeviceInfo](#)
- struct [DeviceStats](#)
- class [IApInterfaceManager](#)  
*Manager class for configuring Wlan Access Points. [More...](#)*
- class [IApListener](#)
- struct [InterfaceStatus](#)
- class [IStaInterfaceManager](#)  
*Manager class for configuring Wlan Station Mode. [More...](#)*
- class [IStaListener](#)
- class [IWlanDeviceManager](#)  
*WlanDeviceManager is a primary interface for configuring Wireless LAN. it provide APIs to enable, configure, activate, and modify modes. [More...](#)*
- class [IWlanListener](#)
- struct [StaConfig](#)
- struct [StaStaticIpConfig](#)
- struct [StaStatus](#)
- class [WlanFactory](#)  
*WlanFactory is the central factory to create all wlan classes. [More...](#)*

# 7 Data Structure Documentation

---

## 7.1 telux::tel::CellBroadcastMessage Class Reference

Cell Broadcast message.

### Public member functions

- [CellBroadcastMessage](#) (std::shared\_ptr< [EtwsInfo](#) > etwsInfo)
- [CellBroadcastMessage](#) (std::shared\_ptr< [CmasInfo](#) > cmasInfo)
- [MessageType](#) getMessageType () const
- std::shared\_ptr< [EtwsInfo](#) > getEtwsInfo () const
- std::shared\_ptr< [CmasInfo](#) > getCmasInfo () const

Cell Broadcast message.

### 7.1.1 Constructors and Destructors

#### 7.1.1.1 telux::tel::CellBroadcastMessage::CellBroadcastMessage ( std::shared\_ptr< [EtwsInfo](#) > *etwsInfo* )

[CellBroadcastMessage](#) constructor.

#### Parameters

in	<i>etwsInfo</i>	<a href="#">EtwsInfo</a>
----	-----------------	--------------------------

#### 7.1.1.2 telux::tel::CellBroadcastMessage::CellBroadcastMessage ( std::shared\_ptr< [CmasInfo](#) > *cmasInfo* )

[CellBroadcastMessage](#) constructor.

#### Parameters

in	<i>cmasInfo</i>	<a href="#">CmasInfo</a>
----	-----------------	--------------------------

## 7.1.2 Member Function Documentation

### 7.1.2.1 `MessageType telux::tel::CellBroadcastMessage::getMessageType ( ) const`

Get the cellbroadcast message type.

#### Returns

[MessageType](#).

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.1.2.2 `std::shared_ptr<EtwsInfo> telux::tel::CellBroadcastMessage::getEtwsInfo ( ) const`

Get ETWS warning notification containing information about the ETWS warning type, the emergency user alert flag and the popup flag. This method should be called only if message type returned by [getMessageType\(\)](#) API is [MessageType::ETWS](#)

#### Returns

pointer to [EtwsInfo](#) or null if this is not an ETWS warning notification

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.1.2.3 `std::shared_ptr<CmasInfo> telux::tel::CellBroadcastMessage::getCmasInfo ( ) const`

Get CMAS warning notification containing information about the CMAS message class, severity, urgency and certainty. This method should be called only if message type returned by [getMessageType\(\)](#) API is [MessageType::CMAS](#)

#### Returns

pointer to [CmasInfo](#) or null if this is not a CMAS warning notification

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

## 7.2 telux::tel::Circle Class Reference



**Public member functions**

- [Circle](#) ([Point](#) center, double radius)
- [Point](#) `getCenter` ()
- double `getRadius` ()

This class represents a geometry represented as simple circle.

**7.2.1 Constructors and Destructors****7.2.1.1 `telux::tel::Circle::Circle ( Point center, double radius )`**

[Circle](#) constructor.

**Parameters**

in	<i>center</i>	Center of circle represented by <a href="#">telux::tel::Point</a>
in	<i>radius</i>	Radius of circle in meters

**7.2.2 Member Function Documentation****7.2.2.1 `Point telux::tel::Circle::getCenter ( )`**

Get center point of circle.

**Returns**

Center of circle.

**7.2.2.2 `double telux::tel::Circle::getRadius ( )`**

Get radius of circle.

**Returns**

Radius of circle.

**7.3 `telux::tel::CmasInfo` Class Reference****Public member functions**

- [CmasInfo](#) ([GeographicalScope](#) geographicalScope, int msgId, int serialNumber, std::string languageCode, std::string messageText, [MessagePriority](#) priority, [CmasMessageClass](#) messageClass, [CmasSeverity](#) severity, [CmasUrgency](#) urgency, [CmasCertainty](#) certainty, std::shared\_ptr<[WarningAreaInfo](#)> warningAreaInfo)
- [GeographicalScope](#) `getGeographicalScope` () const
- int `getMessageId` () const

- int [getSerialNumber](#) () const
- std::string [getLanguageCode](#) () const
- std::string [getMessageBody](#) () const
- [MessagePriority](#) [getPriority](#) () const
- int [getMessageCode](#) () const
- int [getUpdateNumber](#) () const
- [CmasMessageClass](#) [getMessageClass](#) ()
- [CmasSeverity](#) [getSeverity](#) ()
- [CmasUrgency](#) [getUrgency](#) ()
- [CmasCertainty](#) [getCertainty](#) ()
- std::shared\_ptr< [WarningAreaInfo](#) > [getWarningAreaInfo](#) ()

Contains information elements for a GSM/UMTS/E-UTRAN/NG-RAN CMAS warning notification. Supported values for each element are defined in 3GPP TS 23.041.

## 7.3.1 Constructors and Destructors

**7.3.1.1 [telux::tel::CmasInfo::CmasInfo](#) ( [GeographicalScope](#) *geographicalScope*, int *msgId*, int *serialNumber*, std::string *languageCode*, std::string *messageText*, [MessagePriority](#) *priority*, [CmasMessageClass](#) *messageClass*, [CmasSeverity](#) *severity*, [CmasUrgency](#) *urgency*, [CmasCertainty](#) *certainty*, std::shared\_ptr< [WarningAreaInfo](#) > *warningAreaInfo* )**

[CmasInfo](#) constructor.

### Parameters

in	<i>geographicalScope</i>	<a href="#">GeographicalScope</a>
in	<i>msgId</i>	Unique message identifier
in	<i>serialNumber</i>	Serial number for message
in	<i>languageCode</i>	ISO-639-1 language code for message
in	<i>messageText</i>	Message text
in	<i>priority</i>	<a href="#">MessagePriority</a>
in	<i>messageClass</i>	<a href="#">CmasMessageClass</a>
in	<i>severity</i>	<a href="#">CmasSeverity</a>
in	<i>urgency</i>	<a href="#">CmasUrgency</a>
in	<i>certainty</i>	<a href="#">CmasCertainty</a>

## 7.3.2 Member Function Documentation

### 7.3.2.1 GeographicalScope telux::tel::CmasInfo::getGeographicalScope ( ) const

Get the geographicalScope of cellbroadcast message.

#### Returns

GeographicalScope.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.3.2.2 int telux::tel::CmasInfo::getMessageId ( ) const

Get cellbroadcast message identifier. The message identifier identifies the type of the cell broadcast message defined in spec 3GPP TS 23.041 9.4.1.2.2

#### Returns

Message identifier.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.3.2.3 int telux::tel::CmasInfo::getSerialNumber ( ) const

Get the serial number of broadcast (geographical scope + message code + update number for GSM/UMTS).

#### Returns

int containing cellbroadcast serial number.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.3.2.4 std::string telux::tel::CmasInfo::getLanguageCode ( ) const

Get the ISO-639-1 language code for cell broadcast message, or empty string if unspecified. This is not applicable for ETWS primary notification.

#### Returns

Language code

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**7.3.2.5 std::string telux::tel::CmasInfo::getMessageBody ( ) const**

Get the body of cell broadcast message, or empty string if no body available. For ETWS primary notification based on message identifier pre canned message will be sent.

**Returns**

body or empty string

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**7.3.2.6 MessagePriority telux::tel::CmasInfo::getPriority ( ) const**

Get the priority for the cell broadcast message.

**Returns**

[MessagePriority](#).

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**7.3.2.7 int telux::tel::CmasInfo::getMessageCode ( ) const**

Get the cellbroadcast message code.

**Returns**

int containing cellbroadcast message code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.3.2.8 `int telux::tel::CmasInfo::getUpdateNumber ( ) const`

Get the cellbroadcast message update number.

#### Returns

int containing cellbroadcast message's update number.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.3.2.9 `CmasMessageClass telux::tel::CmasInfo::getMessageClass ( )`

Get CMAS message class.

#### Returns

[CmasMessageClass](#).

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.3.2.10 `CmasSeverity telux::tel::CmasInfo::getSeverity ( )`

Get CMAS message severity.

#### Returns

[CmasSeverity](#).

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.3.2.11 `CmasUrgency telux::tel::CmasInfo::getUrgency ( )`

Get CMAS message urgency.

#### Returns

[CmasUrgency](#).

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.3.2.12 CmasCertainty telux::tel::CmasInfo::getCertainty ( )

Get CMAS message certainty.

#### Returns

[CmasCertainty](#).

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.3.2.13 std::shared\_ptr<WarningAreaInfo> telux::tel::CmasInfo::getWarningAreaInfo ( )

Returns warning area information for alert. This is applicable for LTE and NR5G

#### Returns

pointer to [WarningAreaInfo](#) or null if there is no warning area information available.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

## 7.4 telux::tel::EtwsInfo Class Reference

### Public member functions

- [EtwsInfo](#) ([GeographicalScope](#) geographicalScope, int msgId, int serialNumber, std::string languageCode, std::string messageText, [MessagePriority](#) priority, [EtwsWarningType](#) warningType, bool emergencyUserAlert, bool activatePopup, bool primary, std::vector< uint8\_t > warningSecurityInformation)
- [GeographicalScope](#) [getGeographicalScope](#) () const
- int [getMessageId](#) () const
- int [getSerialNumber](#) () const
- std::string [getLanguageCode](#) () const
- std::string [getMessageBody](#) () const
- [MessagePriority](#) [getPriority](#) () const
- int [getMessageCode](#) () const
- int [getUpdateNumber](#) () const
- [EtwsWarningType](#) [getEtwsWarningType](#) ()

- bool [isEmergencyUserAlert](#) ()
- bool [isPopupAlert](#) ()
- bool [isPrimary](#) ()
- std::vector< uint8\_t > [getWarningSecurityInformation](#) ()

Contains information elements for a GSM/UMTS/E-UTRAN/NG-RAN ETWS warning notification. Supported values for each element are defined in 3GPP TS 23.041.

## 7.4.1 Constructors and Destructors

**7.4.1.1** `telux::tel::EtwInfo::EtwInfo ( GeographicalScope geographicalScope, int msgId, int serialNumber, std::string languageCode, std::string messageText, MessagePriority priority, EtwWarningType warningType, bool emergencyUserAlert, bool activatePopup, bool primary, std::vector< uint8_t > warningSecurityInformation )`

[EtwInfo](#) constructor.

### Parameters

in	<i>geographicalScope</i>	<a href="#">GeographicalScope</a>
in	<i>msgId</i>	Unique message identifier
in	<i>serialNumber</i>	Serial number for message
in	<i>languageCode</i>	ISO-639-1 language code for message
in	<i>messageText</i>	Message text
in	<i>priority</i>	<a href="#">MessagePriority</a>
in	<i>warningType</i>	<a href="#">EtwWarningType</a>
in	<i>emergencyUserAlert</i>	If true message is emergency user alert otherwise not
in	<i>activatePopup</i>	If true message message activate popup flag is set, otherwise popup flag is false.
in	<i>primary</i>	If true ETWS message is primary notification otherwise not
in	<i>warningSecurityInformation</i>	Buffer containing security information about ETWS primary notification such as timestamp and digital signature

## 7.4.2 Member Function Documentation

**7.4.2.1** `GeographicalScope telux::tel::EtwInfo::getGeographicalScope ( ) const`

Get the `geographicalScope` of cellbroadcast message.

### Returns

[GeographicalScope](#).

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**7.4.2.2 int telux::tel::EtwsInfo::getMessageId ( ) const**

Get cellbroadcast message identifier. The message identifier identifies the type of the cell broadcast message defined in spec 3GPP TS 23.041 9.4.1.2.2

**Returns**

Message identifier.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**7.4.2.3 int telux::tel::EtwsInfo::getSerialNumber ( ) const**

Get the serial number of broadcast (geographical scope + message code + update number for GSM/UMTS).

**Returns**

int containing cellbroadcast serial number.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**7.4.2.4 std::string telux::tel::EtwsInfo::getLanguageCode ( ) const**

Get the ISO-639-1 language code for cell broadcast message, or empty string if unspecified. This is not applicable for ETWS primary notification.

**Returns**

Language code

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.



#### 7.4.2.5 `std::string telux::tel::EtwsInfo::getMessageBody ( ) const`

Get the body of cell broadcast message, or empty string if no body available. For ETWS primary notification based on message identifier pre canned message will be sent.

##### Returns

body or empty string

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 7.4.2.6 `MessagePriority telux::tel::EtwsInfo::getPriority ( ) const`

Get the priority for the cell broadcast message.

##### Returns

[MessagePriority](#).

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 7.4.2.7 `int telux::tel::EtwsInfo::getMessageCode ( ) const`

Get the cellbroadcast message code.

##### Returns

int containing cellbroadcast message code.

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 7.4.2.8 `int telux::tel::EtwsInfo::getUpdateNumber ( ) const`

Get the cellbroadcast message update number.

##### Returns

int containing cellbroadcast message's update number.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**7.4.2.9 EtwsWarningType telux::tel::EtwsInfo::getEtwsWarningType ( )**

Get ETWS warning type.

**Returns**

[EtwsWarningType](#).

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**7.4.2.10 bool telux::tel::EtwsInfo::isEmergencyUserAlert ( )**

Returns the ETWS emergency user alert flag.

**Returns**

true to notify terminal to activate emergency user alert or false otherwise

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**7.4.2.11 bool telux::tel::EtwsInfo::isPopupAlert ( )**

Returns the ETWS activate popup flag.

**Returns**

true to notify terminal to activate display popup or false otherwise

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**7.4.2.12 bool telux::tel::EtwsInfo::isPrimary ( )**

Returns the ETWS format flag. This flag determine whether ETWS message is primary notification or not.

**Returns**

true if the message is primary message, otherwise secondary message

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**7.4.2.13 std::vector<uint8\_t> telux::tel::EtwsInfo::getWarningSecurityInformation ( )**

Returns security information about ETWS primary notification such as timestamp and digital signature(applicable only for GSM).

**Returns**

byte buffer

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**7.5 telux::tel::Geometry Class Reference****Public member functions**

- [Geometry](#) (std::shared\_ptr< [Polygon](#) > polygon)
- [Geometry](#) (std::shared\_ptr< [Circle](#) > circle)
- [GeometryType](#) [getType](#) () const
- std::shared\_ptr< [Polygon](#) > [getPolygon](#) () const
- std::shared\_ptr< [Circle](#) > [getCircle](#) () const

This class represents warning area geometry to perform geofencing on alert.

**7.5.1 Constructors and Destructors****7.5.1.1 telux::tel::Geometry::Geometry ( std::shared\_ptr< Polygon > polygon )**

[Geometry](#) constructor.

**Parameters**

in	<i>polygon</i>	<a href="#">Polygon</a>
----	----------------	-------------------------

**7.5.1.2 telux::tel::Geometry::Geometry ( std::shared\_ptr< Circle > circle )**

[Geometry](#) constructor.

in	circle	Circle
----	--------	--------

## 7.5.2 Member Function Documentation

### 7.5.2.1 GeometryType telux::tel::Geometry::getType ( ) const

Get the geometry type.

#### Returns

[GeometryType](#).

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.5.2.2 std::shared\_ptr<Polygon> telux::tel::Geometry::getPolygon ( ) const

Get polygon geometry as warning area to perform geofencing. This method should be called only if geometry type returned by [getType\(\)](#) API is [GeometryType::POLYGON](#)

#### Returns

[Polygon](#) geometry object.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.5.2.3 std::shared\_ptr<Circle> telux::tel::Geometry::getCircle ( ) const

Get circle geometry as warning area to perform geofencing. This method should be called only if geometry type returned by [getType\(\)](#) API is [GeometryType::CIRCLE](#)

#### Returns

[Circle](#) geometry object.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

## 7.6 telux::tel::ICellBroadcastListener Class Reference

A listener class which monitors cell broadcast messages.

## Public member functions

- virtual void [onIncomingMessage](#) (SlotId slotId, const std::shared\_ptr< [CellBroadcastMessage](#) > cbMessage)
- virtual void [onMessageFilterChange](#) (SlotId slotId, std::vector< [CellBroadcastFilter](#) > filters)
- virtual [~ICellBroadcastListener](#) ()

A listener class which monitors cell broadcast messages.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

## 7.6.1 Constructors and Destructors

**7.6.1.1 virtual telux::tel::ICellBroadcastListener::~~ICellBroadcastListener ( )**  
[virtual]

## 7.6.2 Member Function Documentation

**7.6.2.1 virtual void telux::tel::ICellBroadcastListener::onIncomingMessage ( SlotId *slotId*, const std::shared\_ptr< [CellBroadcastMessage](#) > *cbMessage* )**  
[virtual]

This function is called when device receives an incoming cell broadcast message.

### Parameters

in	<i>slotId</i>	- Slot Id on which broadcast message is received.
in	<i>cbMessage</i>	- Broadcast message with information related to ETWS/CMAS notification.

### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**7.6.2.2 virtual void telux::tel::ICellBroadcastListener::onMessageFilterChange ( SlotId *slotId*, std::vector< [CellBroadcastFilter](#) > *filters* )** [virtual]

This function is called when there is change in broadcast configuration like updation of message filters by the client using [ICellBroadcastManager::updateMessageFilters](#).

### Parameters

in	<i>slotId</i>	- Slot Id on which change in message filters is received.
in	<i>filters</i>	- Complete list of configured broadcast message filters.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

## 7.7 telux::tel::ICellBroadcastManager Class Reference

CellBroadcastManager class is primary interface to configure and activate emergency broadcast messages and receive broadcast messages.

**Public member functions**

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual SlotId [getSlotId](#) ()=0
- virtual [telux::common::Status updateMessageFilters](#) (std::vector< [CellBroadcastFilter](#) > filters, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status requestMessageFilters](#) ([RequestFiltersResponseCallback](#) callback)=0
- virtual [telux::common::Status setActivationStatus](#) (bool activate, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status requestActivationStatus](#) ([RequestActivationStatusResponseCallback](#) callback)=0
- virtual [telux::common::Status registerListener](#) (std::weak\_ptr< [ICellBroadcastListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak\_ptr< [ICellBroadcastListener](#) > listener)=0
- virtual [~ICellBroadcastManager](#) ()

CellBroadcastManager class is primary interface to configure and activate emergency broadcast messages and receive broadcast messages.

### 7.7.1 Constructors and Destructors

**7.7.1.1 virtual telux::tel::ICellBroadcastManager::~~ICellBroadcastManager ( )**  
[virtual]

### 7.7.2 Member Function Documentation

**7.7.2.1 virtual bool telux::tel::ICellBroadcastManager::isSubsystemReady ( )** [pure virtual]

Checks the status of network subsystem and returns the result.

**Returns**

True if network subsystem is ready for service otherwise false.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.7.2.2 **virtual std::future<bool> telux::tel::ICellBroadcastManager::onSubsystemReady ( ) [pure virtual]**

Wait for network subsystem to be ready.

**Returns**

A future that caller can wait on to be notified when network subsystem is ready.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.7.2.3 **virtual SlotId telux::tel::ICellBroadcastManager::getSlotId ( ) [pure virtual]**

Get associated slot for this CellBroadcastManager.

**Returns**

SlotId

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.7.2.4 **virtual telux::common::Status telux::tel::ICellBroadcastManager::updateMessageFilters ( std::vector< CellBroadcastFilter > filters, telux::common::ResponseCallback callback = nullptr ) [pure virtual]**

Configures the broadcast messages to be received.

**Parameters**

in	<i>filters</i>	List of filtered broadcast message identifiers.
in	<i>callback</i>	Optional callback to get the response of set cell broadcast filters.

**Returns**

Status of updateMessageIdFilters i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.7.2.5 **virtual telux::common::Status telux::tel::ICellBroadcastManager::request← MessageFilters ( RequestFiltersResponseCallback *callback* ) [pure virtual]**

Retrieve configured message filters for which broadcast messages will be received.

**Parameters**

in	<i>callback</i>	Callback to get the response of get cell broadcast filters.
----	-----------------	---

**Returns**

Status of requestMessageIdFilters i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.7.2.6 **virtual telux::common::Status telux::tel::ICellBroadcastManager::set← ActivationStatus ( bool *activate*, telux::common::ResponseCallback *callback* = *nullptr* ) [pure virtual]**

Allows activation and deactivation of configured broadcast messages.

**Parameters**

in	<i>activate</i>	Activate/deactivate broadcast messages.
in	<i>callback</i>	Optional callback pointer to get the response.

**Returns**

Status of setActivationStatus i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.7.2.7 **virtual telux::common::Status telux::tel::ICellBroadcastManager::request← ActivationStatus ( RequestActivationStatusResponseCallback *callback* ) [pure virtual]**

Get activation status for configured broadcast messages.



in	<i>callback</i>	Callback pointer to get the response.
----	-----------------	---------------------------------------

**Returns**

Status of requestActivationStatus i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.7.2.8 virtual telux::common::Status telux::tel::ICellBroadcastManager::register↔ Listener ( std::weak\_ptr< ICellBroadcastListener > *listener* ) [pure virtual]

Register a listener for cell broadcast messages.

**Parameters**

in	<i>listener</i>	Pointer to <a href="#">ICellBroadcastListener</a> object which receives broadcast message.
----	-----------------	--

**Returns**

Status of registerListener i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.7.2.9 virtual telux::common::Status telux::tel::ICellBroadcastManager::deregister↔ Listener ( std::weak\_ptr< ICellBroadcastListener > *listener* ) [pure virtual]

De-register the listener.

**Parameters**

in	<i>listener</i>	Listener to be de-registered
----	-----------------	------------------------------

**Returns**

Status of deregisterListener i.e. success or suitable error code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

## 7.8 telux::cv2x::ICv2xListener Class Reference

Cv2x Radio Manager listeners implement this interface.

### Public member functions

- virtual void [onStatusChanged](#) (Cv2xStatus status)
- virtual void [onStatusChanged](#) (Cv2xStatusEx status)
- virtual [~ICv2xListener](#) ()

Cv2x Radio Manager listeners implement this interface.

### 7.8.1 Constructors and Destructors

#### 7.8.1.1 virtual telux::cv2x::ICv2xListener::~~ICv2xListener ( ) [virtual]

Destructor for [ICv2xListener](#)

### 7.8.2 Member Function Documentation

#### 7.8.2.1 virtual void telux::cv2x::ICv2xListener::onStatusChanged ( Cv2xStatus *status* ) [virtual]

Called when the status of the CV2X radio has changed.

#### Parameters

in	<i>status</i>	- CV2X radio status.
----	---------------	----------------------

**Deprecated** use [onStatusChanged\(Cv2xStatusEx status\)](#)

#### 7.8.2.2 virtual void telux::cv2x::ICv2xListener::onStatusChanged ( Cv2xStatusEx *status* ) [virtual]

Called when the status of the CV2X radio has changed.

#### Parameters

in	<i>status</i>	- CV2X radio status.
----	---------------	----------------------

## 7.9 telux::data::IEspFilter Class Reference

This class represents a IP Filter for the ESP, get the new instance from [telux::data::DataFactory](#).

**Public member functions**

- virtual [EspInfo](#) `getEspInfo ()=0`
- virtual `telux::common::Status` `setEspInfo (const EspInfo &espInfo)=0`
- virtual `~IEspFilter ()`

This class represents a IP Filter for the ESP, get the new instance from [telux::data::DataFactory](#).

**7.9.1 Constructors and Destructors****7.9.1.1 virtual `telux::data::IEspFilter::~IEspFilter ( ) [virtual]`**

Destructor for [IEspFilter](#)

**7.9.2 Member Function Documentation****7.9.2.1 virtual `EspInfo` `telux::data::IEspFilter::getEspInfo ( ) [pure virtual]`**

Get the ESP header info

**Returns**

[telux::data::EspInfo](#)

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**7.9.2.2 virtual `telux::common::Status` `telux::data::IEspFilter::setEspInfo ( const EspInfo & espInfo ) [pure virtual]`**

sets the ICMP header info

**Parameters**

in	<i>espInfo</i>	<a href="#">EspInfo</a> structure <a href="#">telux::data::EspInfo</a>
----	----------------	--

**Returns**

Immediate status of [setEspInfo\(\)](#) request sent i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

## 7.10 telux::data::IcmpFilter Class Reference

This class represents a IP Filter for the ICMP, get the new instance from [telux::data::DataFactory](#).

**Public member functions**

- virtual [IcmpInfo](#) [getIcmpInfo](#) ()=0
- virtual [telux::common::Status](#) [setIcmpInfo](#) (const [IcmpInfo](#) &icmpInfo)=0
- virtual [~IcmpFilter](#) ()

This class represents a IP Filter for the ICMP, get the new instance from [telux::data::DataFactory](#).

### 7.10.1 Constructors and Destructors

#### 7.10.1.1 virtual telux::data::IcmpFilter::~~IcmpFilter ( ) [virtual]

Destructor for [IcmpFilter](#)

### 7.10.2 Member Function Documentation

#### 7.10.2.1 virtual IcmpInfo telux::data::IcmpFilter::getIcmpInfo ( ) [pure virtual]

Get the ICMP header info

**Returns**

[telux::data::IcmpInfo](#)

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 7.10.2.2 virtual telux::common::Status telux::data::IcmpFilter::setIcmpInfo ( const IcmpInfo & icmpInfo ) [pure virtual]

sets the ICMP header info

**Parameters**

in	<i>icmpInfo</i>	<a href="#">TcpInfo</a> structure <a href="#">telux::data::IcmpInfo</a>
----	-----------------	---

**Returns**

Immediate status of [setIcmpInfo\(\)](#) request sent i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

## 7.11 telux::tel::IImServingSystemListener Class Reference

A listener class for monitoring changes in IMS Serving System manager, including IMS registration status change. Override the methods for the state that you wish to receive updates for.

**Public member functions**

- virtual void [onServiceStatusChange](#) ([telux::common::ServiceStatus](#) status)
- virtual void [onImsRegStatusChange](#) ([ImsRegistrationInfo](#) status)
- virtual [~IImServingSystemListener](#) ()

A listener class for monitoring changes in IMS Serving System manager, including IMS registration status change. Override the methods for the state that you wish to receive updates for.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

### 7.11.1 Constructors and Destructors

**7.11.1.1** virtual [telux::tel::IImServingSystemListener::~~IImServingSystemListener](#) (  
) [**virtual**]

### 7.11.2 Member Function Documentation

**7.11.2.1** virtual void [telux::tel::IImServingSystemListener::onServiceStatusChange](#) (  
[telux::common::ServiceStatus](#) *status* ) [**virtual**]

This function is called when service status changes.

**Parameters**

in	<i>status</i>	- <a href="#">telux::common::ServiceStatus</a>
----	---------------	--

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

Reimplemented from [telux::common::IServiceStatusListener](#).

### 7.11.2.2 virtual void telux::tel::ImsServingSystemListener::onImsRegStatusChange ( ImsRegistrationInfo status ) [virtual]

This function is called whenever any IMS service configuration is changed.

in	<i>status</i>	Indicates which registration status is the IMS service changed to. <a href="#">telux::tel::ImsRegistrationInfo</a> .
----	---------------	--

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

## 7.12 telux::tel::ImsServingSystemManager Class Reference

IMS Serving System Manager is the primary interface for IMS related operations Allows to query IMS registration status.

#### Public member functions

- virtual [telux::common::ServiceStatus](#) getServiceStatus ()=0
- virtual [telux::common::Status](#) requestRegistrationInfo (ImsRegistrationInfoCb callback)=0
- virtual [telux::common::Status](#) registerListener (std::weak\_ptr< [telux::tel::ImsServingSystemListener](#) > listener)=0
- virtual [telux::common::Status](#) deregisterListener (std::weak\_ptr< [telux::tel::ImsServingSystemListener](#) > listener)=0
- virtual [~ImsServingSystemManager](#) ()

IMS Serving System Manager is the primary interface for IMS related operations Allows to query IMS registration status.

### 7.12.1 Constructors and Destructors

#### 7.12.1.1 virtual telux::tel::ImsServingSystemManager::~~ImsServingSystemManager ( ) [virtual]

### 7.12.2 Member Function Documentation

#### 7.12.2.1 virtual telux::common::ServiceStatus telux::tel::ImsServingSystemManager::getServiceStatus ( ) [pure virtual]

This status indicates whether the [ImsServingSystemManager](#) object is in a usable state.

#### Returns

SERVICE\_AVAILABLE - If IMS Serving System manager is ready for service.

SERVICE\_UNAVAILABLE - If IMS Serving System manager is temporarily unavailable.  
 SERVICE\_FAILED - If IMS Serving System manager encountered an irrecoverable failure.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 7.12.2.2 **virtual telux::common::Status telux::tel::ImsServingSystemManager↔ ::requestRegistrationInfo ( ImsRegistrationInfoCb *callback* ) [pure virtual]**

Request IMS registration information.

#### Parameters

in	<i>callback</i>	Callback pointer to get the response of requestRegistrationInfo.
----	-----------------	--

#### Returns

Status of requestRegistrationInfo i.e. success or suitable status code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 7.12.2.3 **virtual telux::common::Status telux::tel::ImsServingSystemManager↔ ::registerListener ( std::weak\_ptr< telux::tel::ImsServingSystemListener > *listener* ) [pure virtual]**

Add a listener to listen for specific events in the IMS Serving System subsystem.

#### Parameters

in	<i>listener</i>	Pointer to <a href="#">ImsServingSystemListener</a> object that processes the notification
----	-----------------	--

#### Returns

Status of registerListener i.e. success or suitable error code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

#### 7.12.2.4 virtual telux::common::Status telux::tel::llmservingSystemManager↔ ::deregisterListener ( std::weak\_ptr< telux::tel::llmservingSystemListener > *listener* ) [pure virtual]

Remove a previously added listener.

##### Parameters

in	<i>listener</i>	Listener to be removed.
----	-----------------	-------------------------

##### Returns

Status of deregisterListener i.e. success or suitable error code.

##### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

## 7.13 telux::data::llpfilter Class Reference

A IP filter class to add specific filters like what data will be allowed from the modem to the application processor. Only data packets that match the filter will be sent to the apps processor. Also used to configure Firewall rules.

##### Public member functions

- virtual [IPv4Info](#) getIPv4Info ()=0
- virtual [telux::common::Status](#) setIPv4Info (const [IPv4Info](#) &ipv4Info)=0
- virtual [IPv6Info](#) getIPv6Info ()=0
- virtual [telux::common::Status](#) setIPv6Info (const [IPv6Info](#) &ipv6Info)=0
- virtual [IpProtocol](#) getIpProtocol ()=0
- virtual [IpFamilyType](#) getIpFamily ()=0
- virtual [~IIPFilter](#) ()

A IP filter class to add specific filters like what data will be allowed from the modem to the application processor. Only data packets that match the filter will be sent to the apps processor. Also used to configure Firewall rules.

### 7.13.1 Constructors and Destructors

#### 7.13.1.1 virtual telux::data::llpfilter::~llpfilter ( ) [virtual]

Destructor for [IIPFilter](#)



## 7.13.2 Member Function Documentation

### 7.13.2.1 virtual IPv4Info telux::data::llpFilter::getIPv4Info ( ) [pure virtual]

Get the IPv4 header info

#### Returns

[telux::data::IPv4Info](#)

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.13.2.2 virtual telux::common::Status telux::data::llpFilter::setIPv4Info ( const IPv4Info & *ipv4Info* ) [pure virtual]

sets the IPv4 header info

#### Parameters

in	<i>ipv4Info</i>	IPv4 structure <a href="#">telux::data::IPv4Info</a>
----	-----------------	--

#### Returns

Immediate status of [setIPv4Info\(\)](#) request sent i.e. success or suitable status code.

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.13.2.3 virtual IPv6Info telux::data::llpFilter::getIPv6Info ( ) [pure virtual]

Get the IPv6 header info

#### Returns

[telux::data::IPv6Info](#)

#### Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.13.2.4 virtual telux::common::Status telux::data::llpFilter::setIPv6Info ( const IPv6Info & *ipv6Info* ) [pure virtual]

sets the IPv6 header info

in	<i>ipv6Info</i>	IPv6 structure <a href="#">telux::data::IPv6Info</a>
----	-----------------	--

**Returns**

Immediate status of [setIPv6Info\(\)](#) request sent i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**7.13.2.5 virtual IpProtocol telux::data::IIPFilter::getIpProtocol ( ) [pure virtual]**

Get the IpProtocol Number

**Returns**

[telux::data::IpProtocol](#)

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**7.13.2.6 virtual IpFamilyType telux::data::IIPFilter::getIpFamily ( ) [pure virtual]**

Get the IP family type

**Returns**

[telux::data::IpFamilyType](#)

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

**7.14 telux::common::IServiceStatusListener Class Reference****Public member functions**

- virtual void [onServiceStatusChange](#) ([ServiceStatus](#) status)
- virtual [~IServiceStatusListener](#) ()

## 7.14.1 Constructors and Destructors

7.14.1.1 `virtual telux::common::IServiceStatusListener::~~IServiceStatusListener ( ) [virtual]`

## 7.14.2 Member Function Documentation

7.14.2.1 `virtual void telux::common::IServiceStatusListener::onServiceStatusChange ( ServiceStatus status ) [virtual]`

This function is called when service status changes.

### Parameters

in	<i>status</i>	- <a href="#">ServiceStatus</a>
----	---------------	---------------------------------

Reimplemented in [telux::tel::ImsServingSystemListener](#).

## 7.15 telux::data::ITcpFilter Class Reference

This class represents a IP Filter for the TCP, get the new instance from [telux::data::DataFactory](#).

### Public member functions

- virtual [TcpInfo](#) `getTcpInfo ()=0`
- virtual [telux::common::Status](#) `setTcpInfo (const TcpInfo &tcpInfo)=0`
- virtual `~ITcpFilter ()`

This class represents a IP Filter for the TCP, get the new instance from [telux::data::DataFactory](#).

## 7.15.1 Constructors and Destructors

7.15.1.1 `virtual telux::data::ITcpFilter::~~ITcpFilter ( ) [virtual]`

Destructor for [ITcpFilter](#)

## 7.15.2 Member Function Documentation

7.15.2.1 `virtual TcpInfo telux::data::ITcpFilter::getTcpInfo ( ) [pure virtual]`

Get the TCP header info

### Returns

[telux::data::TcpInfo](#)

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.15.2.2 **virtual telux::common::Status telux::data::ITcpFilter::setTcpInfo ( const TcpInfo & tcpInfo ) [pure virtual]**

sets the TCP header info

**Parameters**

in	<i>tcpInfo</i>	TcpInfo structure <a href="#">telux::data::TcpInfo</a>
----	----------------	--

**Returns**

Immediate status of [setTcpInfo\(\)](#) request sent i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

## 7.16 telux::data::IUdpFilter Class Reference

This class represents a IP Filter for the UDP, get the new instance from [telux::data::DataFactory](#).

**Public member functions**

- virtual [UdpInfo](#) [getUdpInfo](#) ()=0
- virtual [telux::common::Status](#) [setUdpInfo](#) (const [UdpInfo](#) &udpInfo)=0
- virtual [~IUdpFilter](#) ()

This class represents a IP Filter for the UDP, get the new instance from [telux::data::DataFactory](#).

### 7.16.1 Constructors and Destructors

#### 7.16.1.1 **virtual telux::data::IUdpFilter::~~IUdpFilter ( ) [virtual]**

Destructor for [IUdpFilter](#)

### 7.16.2 Member Function Documentation

#### 7.16.2.1 **virtual UdpInfo telux::data::IUdpFilter::getUdpInfo ( ) [pure virtual]**

Get the UDP header info

**Returns**

[telux::data::UdpInfo](#)

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

### 7.16.2.2 `virtual telux::common::Status telux::data::UdpFilter::setUdpInfo ( const UdpInfo & udpInfo ) [pure virtual]`

sets the UDP header info

**Parameters**

in	<i>udpInfo</i>	UdpInfo structure <a href="#">telux::data::UdpInfo</a>
----	----------------	--

**Returns**

Immediate status of [setUdpInfo\(\)](#) request sent i.e. success or suitable status code.

**Note**

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

## 7.17 `telux::tel::Polygon` Class Reference

**Public member functions**

- [Polygon](#) (std::vector< [Point](#) > vertices)
- std::vector< [Point](#) > [getVertices](#) ()

This class represents a simple polygon with different points.

### 7.17.1 Constructors and Destructors

#### 7.17.1.1 `telux::tel::Polygon::Polygon ( std::vector< Point > vertices )`

[Polygon](#) constructor.

**Parameters**

in	<i>vertices</i>	List of <a href="#">telux::tel::Point</a>
----	-----------------	---

### 7.17.2 Member Function Documentation

### 7.17.2.1 `std::vector<Point> telux::tel::Polygon::getVertices ( )`

Get vertices of polygon.

#### Returns

List of [telux::tel::Point](#).

## 7.18 `telux::tel::WarningAreaInfo` Class Reference

### Public member functions

- [WarningAreaInfo](#) (int *maxWaitTime*, `std::vector< Geometry >` *geometries*)
- int [getGeoFenceMaxWaitTime](#) ()
- `std::vector< Geometry >` [getGeometries](#) ()

This class represents warning area information for alert.

### 7.18.1 Constructors and Destructors

#### 7.18.1.1 `telux::tel::WarningAreaInfo::WarningAreaInfo ( int maxWaitTime, std::vector< Geometry > geometries )`

Warning Area Information constructor.

#### Parameters

in	<i>maxWaitTime</i>	Maximum wait time allowed to determine position for alert Range is 0 to 255 where 0 means Zero wait time, 1 - 254 is Geo-Fencing Maximum Wait Time in seconds and 255 means use device default wait time.
in	<i>geometries</i>	Geometries to perform geofencing on alert

### 7.18.2 Member Function Documentation

#### 7.18.2.1 `int telux::tel::WarningAreaInfo::getGeoFenceMaxWaitTime ( )`

Get maximum wait time allowed to determine position for alert.

#### Returns

Maximum wait time for alert in seconds.

### 7.18.2.2 `std::vector<Geometry> telux::tel::WarningAreaInfo::getGeometries ( )`

Get geometries to perform geofencing on alert.

#### Returns

List of [telux::tel::Geometry](#).