

# Telematics SDK

## Interface Specification

v1.26.15

80-PF458-2 Rev. H

June 30, 2022

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

Revision	Date	Description
A	Sep 2017	Initial release
B	Dec 2017	Added subscription APIs and section on Versioning and API Status
C	Jun 2018	Added Cellular Connection Management APIs
D	Oct 2018	Addition of Network Selection and Serving System Management APIs and call flows
E	Nov 2018	Addition of C-V2X APIs
F	Jan 2019	Addition of Audio APIs and call flows
G	Mar 2019	Addition of Thermal manager APIs and call flows
H	May 2019	Addition of TCU Activity Management APIs and call flows
I	Jun 2019	Addition of Audio APIs for play, capture, DTMF and related call flows
J	Jul 2019	Updated the Location APIs and call flows
K	Jul 2019	Addition of Thermal shutdown manager APIs and call flows
L	Sep 2019	Addition of Remote SIM APIs and call flows
M	Sep 2019	Addition of Audio APIs for Loopback, Tone Generator and related call flows
N	Sep 2019	Addition of Audio APIs for compressed audio format playback and related call flows
O	Sep 2019	Addition of modem config APIs and related call flows
P	Oct 2019	Addition of Data Filter APIs and call flows
Q	Oct 2019	Addition of Location concurrent report APIs and call flows
R	Oct 2019	Addition of Location constraint time uncertainty APIs and call flows
S	Nov 2019	Addition of Audio Format Transcoding APIs and call flows
T	Nov 2019	Addition of Compressed audio format playback on voice paths APIs and call flows

# 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	8
2.2.1	Call Management	8
2.2.2	SMS	9
2.2.3	SIM Card Services	9
2.2.4	Phone Information	9
2.2.5	Location Services	10
2.2.6	Data Services	10
2.2.7	Network Selection and Serving System Management	10
2.2.8	C-V2X	11
2.2.9	Audio	11
2.2.10	Thermal Management	14
2.2.11	Thermal Shutdown Management	14
2.2.12	TCU Activity Management	14
2.2.13	Remote SIM	14
2.2.14	Modem Config Management	15
<b>3</b>	<b>Call Flow Diagrams</b>	<b>16</b>
3.1	Application initialization call flow	16
3.1.1	Phone manager initialization	16
3.2	Dial call flow	17
3.3	ECall call flow	18
3.4	Signal strength call flow	20
3.5	Answer, Reject, RejectWithSMS call flow	21
3.6	Hold call flow	22
3.7	Hold, Conference, Swap call flow	23
3.8	SMS call flow	25
3.9	Get applications call flow	26
3.10	Transmit APDU call flow	27
3.10.1	On logical channel	27
3.10.2	On basic channel	28

3.11	SAP card manager call flow	29
3.11.1	Request card reader status, Request ATR, Transmit APDU call flow	29
3.11.2	SIM Turn off, Turn on and Reset call flow	31
3.12	Radio and Service state call flow	33
3.13	Subscription Call flow	33
3.13.1	Subscription initialization	34
3.13.2	Subscription call flow	35
3.14	Call flow for location services	36
3.14.1	Call flow to register/remove listener for generating basic reports	36
3.14.2	Call flow to register/remove listener for generating detailed reports	38
3.14.3	Call flow to register/remove listener for generating detailed engine reports	39
3.14.4	Call flow to enable/disable constraint time uncertainty	40
3.15	Data Connection Manager Call Flow	41
3.15.1	Start/Stop for data connection manager call flow	41
3.16	Data Profile Manager Call Flow	42
3.16.1	Request/Create/Delete/Modify for data profile manager call flow	43
3.17	Data Filter Manager Call Flow	44
3.17.1	Call flow to Set/Get data filter mode	45
3.17.2	Call flow to Add data restrict filter	46
3.17.3	Call flow to Remove data restrict filter	47
3.18	Network Selection Call Flow	48
3.18.1	Network Selection Call Flow	48
3.19	Serving System Call Flow	50
3.19.1	Serving System Call Flow	51
3.20	C-V2X	52
3.20.1	Start/Stop C-V2X Mode	53
3.20.2	C-V2X Radio Manager API	54
3.20.3	C-V2X Radio Initialization	55
3.20.4	C-V2X Radio RX subscription	56
3.20.5	C-V2X Radio TX event-driven flow	57
3.20.6	C-V2X Radio TX SPS flow	58
3.21	Audio	59
3.21.1	Audio Manager API call flow	60
3.21.2	Audio Voice Call Start/Stop call flow	62
3.21.3	Audio Voice Call Device Switch call flow	64
3.21.4	Audio Voice Call Volume/Mute control call flow	66
3.21.5	Call flow to play DTMF tone	68
3.21.6	Call flow to detect DTMF tones	70
3.21.7	Audio Playback call flow	72
3.21.8	Audio Capture call flow	74
3.21.9	Audio Tone Generator call flow	76
3.21.10	Audio Loopback call flow	77
3.21.11	Compressed audio format playback call flow	79
3.21.12	Audio Transcoding Operation Callflow	81
3.21.13	Compressed audio format playback on Voice Paths Callflow	83
3.22	Thermal manager call flow	85
3.23	Thermal shutdown management	85
3.23.1	Call flow to register/remove listener for Thermal auto-shutdown mode updates.	86
3.23.2	Call flow to set/get the Thermal auto-shutdown mode	87
3.23.3	Call flow to manage thermal auto-shutdown from an eCall application.	88

3.24	TCU Activity Management	89
3.24.1	Call flow to register/remove listener for TCU-activity manager	90
3.24.2	Call flow to set the TCU-activity state	91
3.25	Remote SIM call flow	92
3.26	Modem Config Call Flow	93
3.26.1	Call flow to load and activate a modem config file.	94
3.26.2	Call flow to deactivate and delete a modem config file.	95
3.26.3	Call flow to set and get config auto selection mode	96
<b>4</b>	<b>Deprecated List</b>	<b>98</b>
<b>5</b>	<b>Interfaces</b>	<b>101</b>
5.1	Telematics SDK APIs	101
5.2	Phone Factory	102
5.2.1	Data Structure Documentation	102
5.2.1.1	class telux::tel::PhoneFactory	102
5.3	Phone	105
5.3.1	Data Structure Documentation	105
5.3.1.1	class telux::tel::GsmCellIdentity	105
5.3.1.2	class telux::tel::CdmaCellIdentity	106
5.3.1.3	class telux::tel::LteCellIdentity	108
5.3.1.4	class telux::tel::WcdmaCellIdentity	109
5.3.1.5	class telux::tel::TdscdmaCellIdentity	111
5.3.1.6	class telux::tel::CellInfo	112
5.3.1.7	class telux::tel::GsmCellInfo	113
5.3.1.8	class telux::tel::CdmaCellInfo	114
5.3.1.9	class telux::tel::LteCellInfo	115
5.3.1.10	class telux::tel::WcdmaCellInfo	116
5.3.1.11	class telux::tel::TdscdmaCellInfo	117
5.3.1.12	struct telux::tel::ECallMsdOptionals	118
5.3.1.13	struct telux::tel::ECallMsdControlBits	118
5.3.1.14	struct telux::tel::ECallVehicleIdentificationNumber	118
5.3.1.15	struct telux::tel::ECallVehiclePropulsionStorageType	119
5.3.1.16	struct telux::tel::ECallVehicleLocation	119
5.3.1.17	struct telux::tel::ECallVehicleLocationDelta	119
5.3.1.18	struct telux::tel::ECallOptionalPdu	120
5.3.1.19	struct telux::tel::ECallMsdData	120
5.3.1.20	struct telux::tel::ECallModelInfo	120
5.3.1.21	class telux::tel::IPhone	121
5.3.1.22	class telux::tel::ISignalStrengthCallback	125
5.3.1.23	class telux::tel::IVoiceServiceStateCallback	126
5.3.1.24	struct telux::tel::SimRatCapability	126
5.3.1.25	struct telux::tel::CellularCapabilityInfo	127
5.3.1.26	class telux::tel::IPhoneListener	127
5.3.1.27	class telux::tel::IPhoneManager	130
5.3.1.28	class telux::tel::ICellularCapabilityCallback	133
5.3.1.29	class telux::tel::IOperatingModeCallback	134
5.3.1.30	class telux::tel::SignalStrength	135
5.3.1.31	class telux::tel::LteSignalStrengthInfo	137
5.3.1.32	class telux::tel::GsmSignalStrengthInfo	139

	5.3.1.33	class telux::tel::CdmaSignalStrengthInfo	140
	5.3.1.34	class telux::tel::WcdmaSignalStrengthInfo	141
	5.3.1.35	class telux::tel::TdsdmaSignalStrengthInfo	143
	5.3.1.36	class telux::tel::VoiceServiceInfo	143
5.3.2		Enumeration Type Documentation	144
	5.3.2.1	CellType	144
	5.3.2.2	ECallVariant	145
	5.3.2.3	EmergencyCallType	145
	5.3.2.4	ECallMsdTransmissionStatus	145
	5.3.2.5	ECallCategory	145
	5.3.2.6	ECallVehicleType	145
	5.3.2.7	ECallOptionalDataType	146
	5.3.2.8	ECallMode	146
	5.3.2.9	ECallModeReason	146
	5.3.2.10	RadioState	146
	5.3.2.11	ServiceState	146
	5.3.2.12	RadioTechnology	147
	5.3.2.13	RATCapability	147
	5.3.2.14	VoiceServiceTechnology	148
	5.3.2.15	OperatingMode	148
	5.3.2.16	SignalStrengthLevel	148
	5.3.2.17	VoiceServiceState	148
	5.3.2.18	VoiceServiceDenialCause	149
5.4	Call		151
	5.4.1	Data Structure Documentation	151
		5.4.1.1 class telux::tel::ICall	151
		5.4.1.2 class telux::tel::ICallListener	157
		5.4.1.3 class telux::tel::ICallManager	159
		5.4.1.4 class telux::tel::IMakeCallCallback	166
	5.4.2	Enumeration Type Documentation	167
		5.4.2.1 CallDirection	167
		5.4.2.2 CallState	167
		5.4.2.3 CallEndCause	167
5.5	SMS		170
	5.5.1	Data Structure Documentation	170
		5.5.1.1 struct telux::tel::MessageAttributes	170
		5.5.1.2 class telux::tel::SmsMessage	170
		5.5.1.3 class telux::tel::ISmsManager	172
		5.5.1.4 class telux::tel::ISmsListener	174
		5.5.1.5 class telux::tel::ISmscAddressCallback	175
	5.5.2	Enumeration Type Documentation	175
		5.5.2.1 SmsEncoding	175
5.6	SIM Card Services		177
	5.6.1	Data Structure Documentation	177
		5.6.1.1 class telux::tel::ICardApp	177
		5.6.1.2 struct telux::tel::IccResult	180
		5.6.1.3 class telux::tel::ICardManager	181
		5.6.1.4 class telux::tel::ICard	183
		5.6.1.5 class telux::tel::ICardChannelCallback	187
		5.6.1.6 class telux::tel::ICardCommandCallback	188

	5.6.1.7	class telux::tel::ICardListener	188
	5.6.1.8	struct telux::tel::CardReaderStatus	189
	5.6.1.9	class telux::tel::ISapCardManager	189
	5.6.1.10	class telux::tel::IAtrResponseCallback	194
	5.6.1.11	class telux::tel::ISapCardCommandCallback	194
	5.6.1.12	class telux::tel::ICardReaderCallback	195
5.6.2		Enumeration Type Documentation	195
	5.6.2.1	CardState	195
	5.6.2.2	CardLockType	195
	5.6.2.3	AppType	195
	5.6.2.4	AppState	196
	5.6.2.5	SapState	196
	5.6.2.6	SapCondition	196
5.7		Location Services	197
5.7.1		Data Structure Documentation	197
	5.7.1.1	class telux::loc::ILocationConfigurator	197
	5.7.1.2	struct telux::loc::GnssKinematicsData	198
	5.7.1.3	struct telux::loc::TimeInfo	199
	5.7.1.4	struct telux::loc::GlonassTimeInfo	199
	5.7.1.5	union telux::loc::SystemTimeInfo	200
	5.7.1.6	struct telux::loc::SystemTime	200
	5.7.1.7	struct telux::loc::GnssMeasurementInfo	200
	5.7.1.8	struct telux::loc::GnssData	200
	5.7.1.9	class telux::loc::IGpsTime	201
	5.7.1.10	class telux::loc::ISensorDataUsage	202
	5.7.1.11	class telux::loc::ILocationInfo	202
	5.7.1.12	class telux::loc::ILocationInfoBase	212
	5.7.1.13	class telux::loc::ILocationInfoEx	215
	5.7.1.14	class telux::loc::ISVInfo	222
	5.7.1.15	class telux::loc::IGnssSVInfo	225
	5.7.1.16	class telux::loc::IGnssSignalInfo	225
	5.7.1.17	class telux::loc::LocationFactory	226
	5.7.1.18	class telux::loc::ILocationListener	226
	5.7.1.19	class telux::loc::ILocationManager	228
5.7.2		Enumeration Type Documentation	236
	5.7.2.1	FixRecurrence	236
	5.7.2.2	HorizontalAccuracyLevel	236
	5.7.2.3	PositionTechType	236
	5.7.2.4	LocationReliability	236
	5.7.2.5	SbasCorrectionType	237
	5.7.2.6	SessionStatus	237
	5.7.2.7	AltitudeType	237
	5.7.2.8	GnssConstellationType	237
	5.7.2.9	SVHealthStatus	238
	5.7.2.10	SVStatus	238
	5.7.2.11	SVInfoAvailability	238
	5.7.2.12	SensorType	238
	5.7.2.13	MeasurementType	239
	5.7.2.14	GnssPositionTechType	239
	5.7.2.15	KinematicDataValidityType	239

5.7.2.16	GnssSystem	240
5.7.2.17	GnssTimeValidityType	240
5.7.2.18	GlonassTimeValidity	240
5.7.2.19	GnssSignalType	240
5.7.2.20	LocationTechnologyType	241
5.7.2.21	LocationInfoExValidityType	241
5.7.2.22	GnssDataSignalTypes	242
5.7.2.23	GnssDataValidityType	242
5.7.2.24	DrCalibrationStatusType	243
5.7.2.25	LocReqEngineType	243
5.7.2.26	LocationAggregationType	243
5.7.2.27	PositioningEngineType	243
5.7.3	Variable Documentation	244
5.7.3.1	UNKNOWN_CARRIER_FREQ	244
5.7.3.2	UNKNOWN_SIGNAL_MASK	244
5.7.3.3	DEFAULT_TUNC_THRESHOLD	244
5.7.3.4	DEFAULT_TUNC_ENERGY_THRESHOLD	244
5.8	Data Services	245
5.8.1	Define Documentation	245
5.8.1.1	PROFILE_ID_MAX	245
5.8.1.2	IP_PROT_UNKNOWN	245
5.8.2	Data Structure Documentation	245
5.8.2.1	class telux::data::IDataConnectionManager	245
5.8.2.2	class telux::data::IDataCall	248
5.8.2.3	class telux::data::IDataConnectionListener	251
5.8.2.4	struct telux::data::DataRestrictMode	252
5.8.2.5	struct telux::data::PortInfo	252
5.8.2.6	struct telux::data::ProfileParams	253
5.8.2.7	struct telux::data::DataCallStats	253
5.8.2.8	struct telux::data::IpAddrInfo	253
5.8.2.9	struct telux::data::DataCallEndReason	254
5.8.2.10	struct telux::data::VlanConfig	254
5.8.2.11	class telux::data::DataFactory	254
5.8.2.12	class telux::data::IDataFilterListener	258
5.8.2.13	class telux::data::IDataFilterManager	258
5.8.2.14	class telux::data::DataProfile	263
5.8.2.15	class telux::data::IDataProfileListener	266
5.8.2.16	class telux::data::IDataProfileManager	266
5.8.2.17	class telux::data::IDataCreateProfileCallback	271
5.8.2.18	class telux::data::IDataProfileListCallback	271
5.8.2.19	class telux::data::IDataProfileCallback	272
5.8.2.20	union telux::data::DataCallEndReason.__unnamed__	272
5.8.3	Enumeration Type Documentation	273
5.8.3.1	IpFamilyType	273
5.8.3.2	TechPreference	273
5.8.3.3	AuthProtocolType	273
5.8.3.4	DataRestrictModeType	273
5.8.3.5	DataCallStatus	274
5.8.3.6	DataBearerTechnology	274
5.8.3.7	EndReasonType	275



5.8.3.8	MobileIpReasonCode	275
5.8.3.9	InternalReasonCode	276
5.8.3.10	CallManagerReasonCode	277
5.8.3.11	SpecReasonCode	280
5.8.3.12	PPPReasonCode	282
5.8.3.13	EHRPDRReasonCode	282
5.8.3.14	Ipv6ReasonCode	282
5.8.3.15	HandoffReasonCode	283
5.8.3.16	ProfileChangeEvent	283
5.8.3.17	OperationType	283
5.8.3.18	Direction	283
5.8.3.19	InterfaceType	283
5.9	Subscription Management	285
5.9.1	Data Structure Documentation	285
5.9.1.1	class telux::tel::ISubscription	285
5.9.1.2	class telux::tel::ISubscriptionListener	286
5.9.1.3	class telux::tel::ISubscriptionManager	287
5.10	Network Selection	290
5.10.1	Data Structure Documentation	290
5.10.1.1	struct telux::tel::PreferredNetworkInfo	290
5.10.1.2	struct telux::tel::OperatorStatus	290
5.10.1.3	class telux::tel::INetworkSelectionManager	290
5.10.1.4	class telux::tel::OperatorInfo	294
5.10.1.5	class telux::tel::INetworkSelectionListener	295
5.10.2	Enumeration Type Documentation	296
5.10.2.1	RatType	296
5.10.2.2	NetworkSelectionMode	296
5.10.2.3	InUseStatus	296
5.10.2.4	RoamingStatus	296
5.10.2.5	ForbiddenStatus	296
5.10.2.6	PreferredStatus	297
5.11	Serving System	298
5.11.1	Data Structure Documentation	298
5.11.1.1	class telux::tel::IServingSystemManager	298
5.11.1.2	class telux::tel::IServingSystemListener	301
5.11.2	Enumeration Type Documentation	302
5.11.2.1	ServiceDomainPreference	302
5.11.2.2	RatPrefType	302
5.12	Common	303
5.12.1	Data Structure Documentation	303
5.12.1.1	class telux::common::ICommandCallback	303
5.12.1.2	class telux::common::ICommandResponseCallback	303
5.12.1.3	struct telux::common::SdkVersion	304
5.12.1.4	struct telux::common::PlatformVersion	304
5.12.1.5	class telux::common::Version	304
5.12.2	Enumeration Type Documentation	305
5.12.2.1	Status	305
5.12.2.2	ErrorCode	306
5.13	C-V2X	311
5.13.1	Data Structure Documentation	311

5.13.1.1	class telux::cv2x::Cv2xFactory . . . . .	311
5.13.1.2	class telux::cv2x::ICv2xRadio . . . . .	312
5.13.1.3	class telux::cv2x::ICv2xRadioListener . . . . .	322
5.13.1.4	class telux::cv2x::ICv2xRadioManager . . . . .	324
5.13.1.5	struct telux::cv2x::SocketInfo . . . . .	327
5.13.1.6	struct telux::cv2x::Cv2xStatus . . . . .	328
5.13.1.7	struct telux::cv2x::Cv2xPoolStatus . . . . .	328
5.13.1.8	struct telux::cv2x::Cv2xStatusEx . . . . .	328
5.13.1.9	struct telux::cv2x::TxPoolIdInfo . . . . .	329
5.13.1.10	struct telux::cv2x::EventFlowInfo . . . . .	329
5.13.1.11	struct telux::cv2x::SpsFlowInfo . . . . .	330
5.13.1.12	struct telux::cv2x::Cv2xRadioCapabilities . . . . .	330
5.13.1.13	struct telux::cv2x::MacDetails . . . . .	331
5.13.1.14	struct telux::cv2x::SpsSchedulingInfo . . . . .	332
5.13.1.15	struct telux::cv2x::TrustedUEInfo . . . . .	332
5.13.1.16	struct telux::cv2x::TrustedUEInfoList . . . . .	333
5.13.1.17	struct telux::cv2x::IPv6Address . . . . .	333
5.13.1.18	struct telux::cv2x::DataSessionSettings . . . . .	333
5.13.1.19	struct telux::cv2x::IPv6AddrType . . . . .	333
5.13.1.20	struct telux::cv2x::GlobalIPUnicastRoutingInfo . . . . .	334
5.13.1.21	class telux::cv2x::ICv2xRxSubscription . . . . .	334
5.13.1.22	class telux::cv2x::ICv2xTxFlow . . . . .	336
5.13.1.23	class telux::cv2x::ICv2xTxRxSocket . . . . .	338
5.13.2	Enumeration Type Documentation . . . . .	339
5.13.2.1	TrafficCategory . . . . .	339
5.13.2.2	Cv2xStatusType . . . . .	339
5.13.2.3	Cv2xCauseType . . . . .	339
5.13.2.4	TrafficIpType . . . . .	340
5.13.2.5	RadioConcurrencyMode . . . . .	340
5.13.2.6	Cv2xEvent . . . . .	340
5.13.2.7	Priority . . . . .	340
5.13.2.8	Periodicity . . . . .	341
5.14	Audio . . . . .	342
5.14.1	Data Structure Documentation . . . . .	342
5.14.1.1	struct telux::audio::FormatParams . . . . .	342
5.14.1.2	struct telux::audio::AmrwbpParams . . . . .	342
5.14.1.3	struct telux::audio::StreamConfig . . . . .	342
5.14.1.4	struct telux::audio::FormatInfo . . . . .	343
5.14.1.5	struct telux::audio::ChannelVolume . . . . .	343
5.14.1.6	struct telux::audio::StreamVolume . . . . .	343
5.14.1.7	struct telux::audio::StreamMute . . . . .	343
5.14.1.8	struct telux::audio::StreamBuffer . . . . .	344
5.14.1.9	struct telux::audio::DtmfTone . . . . .	344
5.14.1.10	class telux::audio::AudioFactory . . . . .	344
5.14.1.11	class telux::audio::IVoiceListener . . . . .	345
5.14.1.12	class telux::audio::IPlayListener . . . . .	346
5.14.1.13	class telux::audio::ITranscodeListener . . . . .	346
5.14.1.14	class telux::audio::IAudioListener . . . . .	347
5.14.1.15	class telux::audio::IAudioBuffer . . . . .	347
5.14.1.16	class telux::audio::IStreamBuffer . . . . .	349

5.14.1.17	class telux::audio::IAudioManager	350
5.14.1.18	class telux::audio::IAudioDevice	354
5.14.1.19	class telux::audio::IAudioStream	354
5.14.1.20	class telux::audio::IAudioVoiceStream	358
5.14.1.21	class telux::audio::IAudioPlayStream	361
5.14.1.22	class telux::audio::IAudioCaptureStream	363
5.14.1.23	class telux::audio::IAudioLoopbackStream	364
5.14.1.24	class telux::audio::IAudioToneGeneratorStream	365
5.14.1.25	class telux::audio::ITranscoder	366
5.14.2	Enumeration Type Documentation	369
5.14.2.1	DeviceType	369
5.14.2.2	DeviceDirection	369
5.14.2.3	Direction	370
5.14.2.4	StreamType	370
5.14.2.5	StreamDirection	370
5.14.2.6	ChannelType	370
5.14.2.7	AudioFormat	370
5.14.2.8	DtmfLowFreq	371
5.14.2.9	DtmfHighFreq	371
5.14.2.10	AmrwbpFrameFormat	371
5.14.2.11	StopType	371
5.15	Thermal Management	372
5.15.1	Data Structure Documentation	372
5.15.1.1	class telux::therm::ThermalFactory	372
5.15.1.2	struct telux::therm::BoundCoolingDevice	373
5.15.1.3	class telux::therm::IThermalManager	373
5.15.1.4	class telux::therm::ITripPoint	375
5.15.1.5	class telux::therm::IThermalZone	377
5.15.1.6	class telux::therm::ICoolingDevice	379
5.15.1.7	class telux::therm::IThermalShutdownListener	381
5.15.1.8	class telux::therm::IThermalShutdownManager	382
5.15.2	Enumeration Type Documentation	385
5.15.2.1	AutoShutdownMode	385
5.15.2.2	TripType	385
5.15.3	Variable Documentation	386
5.15.3.1	DEFAULT_TIMEOUT	386
5.16	TCU Activity Manager	387
5.16.1	Data Structure Documentation	387
5.16.1.1	class telux::power::PowerFactory	387
5.16.1.2	class telux::power::ITcuActivityListener	388
5.16.1.3	class telux::power::ITcuActivityManager	388
5.16.2	Enumeration Type Documentation	393
5.16.2.1	TcuActivityState	393
5.16.2.2	TcuActivityStateAck	393
5.17	Remote SIM Provisioning	394
5.17.1	Data Structure Documentation	394
5.17.1.1	struct telux::rsp::CustomHeader	394
5.17.1.2	class telux::rsp::IHttpTransactionListener	394
5.17.1.3	class telux::rsp::IHttpTransactionManager	395
5.17.1.4	class telux::rsp::SimProfile	397

5.17.1.5	class telux::rsp::SimProfileFactory	402
5.17.1.6	class telux::rsp::ISimProfileListener	403
5.17.1.7	class telux::rsp::ISimProfileManager	404
5.17.2	Enumeration Type Documentation	408
5.17.2.1	HttpResult	408
5.17.2.2	ProfileType	408
5.17.2.3	IconType	409
5.17.2.4	ProfileClass	409
5.17.2.5	DownloadStatus	409
5.17.2.6	DownloadErrorCause	409
5.17.2.7	PolicyRuleType	409
5.18	Remote SIM	411
5.18.1	Data Structure Documentation	411
5.18.1.1	class telux::tel::IRemoteSimListener	411
5.18.1.2	class telux::tel::IRemoteSimManager	413
5.18.2	Enumeration Type Documentation	419
5.18.2.1	CardErrorCause	419
5.19	Modem Config	420
5.19.1	Data Structure Documentation	420
5.19.1.1	class telux::config::ConfigFactory	420
5.19.1.2	struct telux::config::ConfigInfo	421
5.19.1.3	class telux::config::IModemConfigListener	421
5.19.1.4	class telux::config::IModemConfigManager	422
5.19.2	Enumeration Type Documentation	427
5.19.2.1	ConfigType	427
5.19.2.2	AutoSelectionMode	427
5.19.2.3	ConfigUpdateStatus	428
5.20	Telematics_cv2x_cpp	429
5.20.1	Data Structure Documentation	429
5.20.1.1	class telux::cv2x::ICv2xThrottleManagerListener	429
5.20.1.2	class telux::cv2x::ICv2xThrottleManager	430
5.21	V2x_api_common	432
5.21.1	Data Structure Documentation	432
5.21.1.1	struct v2x_api_ver_t	432
5.21.2	Enumeration Type Documentation	432
5.21.2.1	v2x_status_enum_type	432
5.22	V2x_api_kinematics	433
5.22.1	Define Documentation	433
5.22.1.1	V2X_KINEMATICS_HANDLE_BAD	433
5.22.2	Data Structure Documentation	433
5.22.2.1	struct v2x_GNSSstatus_t	433
5.22.2.2	struct v2x_gnss_fix_rates_supported_list_t	434
5.22.2.3	struct v2x_init_t	434
5.22.2.4	struct v2x_kinematics_capabilities_t_feature_flags_t	435
5.22.2.5	struct v2x_rates_t	436
5.22.2.6	struct v2x_kinematics_capabilities_t	436
5.22.2.7	struct v2x_location_fix_t	437
5.22.3	Enumeration Type Documentation	440
5.22.3.1	v2x_fix_mode_t	440
5.22.4	Function Documentation	440

5.22.4.1	v2x_kinematics_api_version(void) . . . . .	440
5.22.4.2	v2x_kinematics_init(v2x_init_t *param, v2x_kinematics_init_↔ callback_t cb, void *context) . . . . .	441
5.22.4.3	v2x_kinematics_start_rate_notification(v2x_kinematics_handle_↔ _t handle, v2x_kinematics_rate_notification_listener_t cb, void *context) . . . . .	442
5.22.4.4	v2x_kinematics_set_rate(v2x_kinematics_handle_t handle, v2x_↔ _rates_t *rate, v2x_kinematics_set_rate_callback_t cb, void *context)	443
5.22.4.5	v2x_kinematics_register_listener(v2x_kinematics_handle_t handle, v2x_kinematics_newfix_listener_t listener, void *context) . . . . .	444
5.22.4.6	v2x_kinematics_deregister_listener(v2x_kinematics_handle_↔ t handle, v2x_kinematics_deregister_callback_t cb, void *context)	445
5.22.4.7	v2x_kinematics_final(v2x_kinematics_handle_t handle, v2x_↔ kinematics_final_callback_t cb, void *context) . . . . .	446
5.22.4.8	v2x_kinematics_enable_fixes(v2x_kinematics_handle_t handle)	447
5.22.4.9	v2x_kinematics_disable_fixes(v2x_kinematics_handle_t handle)	447
5.23	V2x_api_radio . . . . .	448
5.23.1	Define Documentation . . . . .	448
5.23.1.1	V2X_RADIO_HANDLE_BAD . . . . .	448
5.23.1.2	V2X_MAX_RADIO_SESSIONS . . . . .	448
5.23.1.3	V2X_RX_WILDCARD_PORTNUM . . . . .	448
5.23.1.4	MAX_POOL_IDS_LIST_LEN . . . . .	448
5.23.1.5	MAX_MALICIOUS_IDS_LIST_LEN . . . . .	448
5.23.1.6	MAX_TRUSTED_IDS_LIST_LEN . . . . .	448
5.23.1.7	MAX_SUBSCRIBE_SIDS_LIST_LEN . . . . .	448
5.23.2	Data Structure Documentation . . . . .	448
5.23.2.1	struct trusted_ue_info_t . . . . .	448
5.23.2.2	struct tx_pool_id_info_t . . . . .	449
5.23.2.3	struct v2x_iface_capabilities_t . . . . .	450
5.23.2.4	struct v2x_tx_bandwidth_reservation_t . . . . .	452
5.23.2.5	struct v2x_chan_meas_params_t . . . . .	453
5.23.2.6	struct v2x_chan_measurements_t . . . . .	453
5.23.2.7	struct v2x_radio_calls_t . . . . .	454
5.23.2.8	struct v2x_sps_mac_details_t . . . . .	457
5.23.2.9	struct v2x_per_sps_reservation_calls_t . . . . .	458
5.23.2.10	struct v2x_tx_flow_info_t . . . . .	460
5.23.2.11	struct v2x_sock_info_t . . . . .	460
5.23.2.12	struct v2x_sid_list_t . . . . .	461
5.23.2.13	struct v2x_tx_sps_flow_info_t . . . . .	461
5.23.2.14	struct socket_info_t . . . . .	461
5.23.3	Enumeration Type Documentation . . . . .	461
5.23.3.1	v2x_concurrency_sel_t . . . . .	461
5.23.3.2	v2x_event_t . . . . .	462
5.23.3.3	v2x_priority_et . . . . .	463
5.23.3.4	v2x_service_status_t . . . . .	463
5.23.3.5	v2x_auto_retransmit_policy_t . . . . .	463
5.23.3.6	traffic_ip_type_t . . . . .	463
5.23.4	Function Documentation . . . . .	464
5.23.4.1	v2x_convert_priority_to_traffic_class(v2x_priority_et priority) . . . . .	464
5.23.4.2	v2x_convert_traffic_class_to_priority(uint16_t traffic_class) . . . . .	465

5.23.4.3	v2x_radio_api_version()	465
5.23.4.4	v2x_radio_query_parameters(const char *iface_name, v2x_↔ iface_capabilities_t *caps)	466
5.23.4.5	v2x_radio_deinit(v2x_radio_handle_t handle)	467
5.23.4.6	v2x_radio_rx_sock_create_and_bind(v2x_radio_handle_t han- dle, int *sock, struct sockaddr_in6 *rx_sockaddr)	468
5.23.4.7	v2x_radio_rx_sock_create_and_bind_v2(v2x_radio_handle_↔ t handle, int id_list_len, uint32_t *id_list, int *sock, struct sockaddr_↔ _in6 *rx_sockaddr)	469
5.23.4.8	v2x_radio_rx_sock_create_and_bind_v3(v2x_radio_handle_↔ t handle, uint16_t port_num, int id_list_len, uint32_t *id_list, int *sock, struct sockaddr_in6 *rx_sockaddr)	471
5.23.4.9	v2x_radio_sock_create_and_bind(v2x_radio_handle_t handle, v2x_tx_sps_flow_info_t *tx_flow_info, v2x_per_sps_reservation_↔ _calls_t *calls, int tx_sps_portnum, int tx_event_portnum, int rx_↔ _portnum, v2x_sid_list_t *rx_id_list, v2x_sock_info_t *tx_sps_↔ _sock, v2x_sock_info_t *tx_event_sock, v2x_sock_info_t *rx_↔ sock)	473
5.23.4.10	v2x_radio_tx_sps_sock_create_and_bind(v2x_radio_handle_t handle, v2x_tx_bandwidth_reservation_t *res, v2x_per_sps_↔ reservation_calls_t *calls, int sps_portnum, int event_portnum, int *sps_sock, struct sockaddr_in6 *sps_sockaddr, int *event_↔ _sock, struct sockaddr_in6 *event_sockaddr)	475
5.23.4.11	v2x_radio_tx_sps_only_create(v2x_radio_handle_t handle, v2x_↔ _tx_bandwidth_reservation_t *res, v2x_per_sps_reservation_↔ calls_t *calls, int sps_portnum, int *sps_sock, struct sockaddr_↔ _in6 *sps_sockaddr)	477
5.23.4.12	v2x_radio_tx_reservation_change(int *sps_sock, v2x_tx_bandwidth_↔ _reservation_t *updated_reservation)	479
5.23.4.13	v2x_radio_tx_event_sock_create_and_bind(const char *interface, int v2x_id, int event_portnum, struct sockaddr_in6 *event_sock_↔ _addr, int *sock)	480
5.23.4.14	v2x_radio_start_measurements(v2x_radio_handle_t handle, v2x_↔ _chan_meas_params_t *measure_this_way)	481
5.23.4.15	v2x_radio_stop_measurements(v2x_radio_handle_t handle)	482
5.23.4.16	v2x_radio_sock_close(int *sock_fd)	483
5.23.4.17	v2x_radio_set_log_level(int new_level, int use_syslog)	484
5.23.4.18	cv2x_status_poll(uint64_t *status_age_useconds)	484
5.23.4.19	v2x_radio_trigger_l2_update(v2x_radio_handle_t handle)	485
5.23.4.20	v2x_radio_update_trusted_ue_list(unsigned int malicious_list_↔ _len, unsigned int malicious_list[MAX_MALICIOUS_IDS_LIST_↔ _LEN], unsigned int trusted_list_len, trusted_ue_info_t trusted_↔ _list[MAX_TRUSTED_IDS_LIST_LEN])	486
5.23.4.21	v2x_radio_tx_sps_sock_create_and_bind_v2(v2x_radio_handle_↔ _t handle, v2x_tx_sps_flow_info_t *sps_flow_info, v2x_per_↔ sps_reservation_calls_t *calls, int sps_portnum, int event_portnum, int *sps_sock, struct sockaddr_in6 *sps_sockaddr, int *event_↔ _sock, struct sockaddr_in6 *event_sockaddr)	487

5.23.4.22	v2x_radio_tx_sps_only_create_v2(v2x_radio_handle_t handle, v2x_tx_sps_flow_info_t *sps_flow_info, v2x_per_sps_reservation_calls_t *calls, int sps_portnum, int *sps_sock, struct sockaddr_in6 *sps_sockaddr) . . . . .	489
5.23.4.23	v2x_radio_tx_reservation_change_v2(int *sps_sock, v2x_tx_sps_flow_info_t *updated_flow_info) . . . . .	491
5.23.4.24	v2x_radio_tx_event_flow_info_change(int *sock, v2x_tx_flow_info_t *updated_flow_info) . . . . .	492
5.23.4.25	v2x_update_configuration(const char *config_file_path) . . . . .	493
5.23.4.26	start_v2x_mode() . . . . .	494
5.23.4.27	stop_v2x_mode() . . . . .	494
5.23.4.28	v2x_radio_init_v2(traffic_ip_type_t ip_type, v2x_concurrency_sel_t mode, v2x_radio_calls_t *callbacks_p, void *ctx_p) . . . . .	495
5.23.4.29	v2x_radio_tx_event_sock_create_and_bind_v3(traffic_ip_type_t ip_type, int v2x_id, int event_portnum, v2x_tx_flow_info_t *event_flow_info, struct sockaddr_in6 *event_sockaddr, int *sock) . . . . .	496
5.23.4.30	get_iface_name(traffic_ip_type_t ip_type, char *iface_name, size_t buffer_len) . . . . .	497
5.23.4.31	v2x_radio_tcp_sock_create_and_bind(v2x_radio_handle_t handle, const v2x_tx_flow_info_t *event_info, const socket_info_t *sock_info, int *sock_fd, struct sockaddr_in6 *sockaddr) . . . . .	498
5.23.4.32	v2x_set_global_IPAddr(uint8_t prefix_len, uint8_t *ipv6_addr) . . . . .	500
5.23.4.33	v2x_set_ip_routing_info(uint8_t *dest_mac_addr) . . . . .	500
5.24	V2x_api_vehicle . . . . .	501
5.24.1	Define Documentation . . . . .	501
5.24.1.1	V2X_VDATA_HANDLE_BAD . . . . .	501
5.24.1.2	V2X_J2735_TRACTION_CONTROL_MAX . . . . .	501
5.24.1.3	V2X_TRACTION_CTRL_MAX . . . . .	501
5.24.1.4	J2735_ABS_MAX . . . . .	501
5.24.1.5	V2X_STABILITY_CONTROL_MAX . . . . .	501
5.24.1.6	V2X_AUX_BRAKE_MAX . . . . .	501
5.24.2	Data Structure Documentation . . . . .	502
5.24.2.1	union v2x_control_status_ut . . . . .	502
5.24.2.2	struct v2x_control_status_ut.bits . . . . .	502
5.24.2.3	union vehicleEventFlags_ut . . . . .	503
5.24.2.4	struct vehicleEventFlags_ut.bits . . . . .	504
5.24.2.5	union ExteriorLights_ut . . . . .	505
5.24.2.6	struct ExteriorLights_ut.bits . . . . .	505
5.24.2.7	struct high_resolution_motion_t . . . . .	506
5.24.2.8	struct current_dynamic_vehicle_state_t . . . . .	507
5.24.2.9	struct static_vehicle_parameters_t . . . . .	507
5.24.3	Enumeration Type Documentation . . . . .	509
5.24.3.1	v2x_transmission_state_enum_type . . . . .	509
5.24.3.2	v2x_BrakeBoostApplied_enum_type . . . . .	509
5.24.3.3	v2x_TractionControlStatus_enum_type . . . . .	509
5.24.3.4	v2x_AntiLockBrakeStatus_enum_type . . . . .	510
5.24.3.5	v2x_StabilityControlStatus_enum_type . . . . .	510
5.24.3.6	v2x_AuxBrakeStatus_enum_type . . . . .	510
5.24.4	Function Documentation . . . . .	511
5.24.4.1	v2x_vehicle_api_version(void) . . . . .	511

5.24.4.2	v2x_vehicle_get_static_params(static_vehicle_parameters_t *parameters)	511
5.24.4.3	v2x_high_res_motion_register_listener(v2x_high_res_motion_listener_t cb)	512
5.24.4.4	v2x_high_res_motion_deregister_listener(v2x_motion_data_handle_t handle)	512
5.24.4.5	v2x_vehicle_register_listener(v2x_vehicle_event_listener_t cb, void *context)	513
5.24.4.6	v2x_vehicle_deregister_for_callback(v2x_vehicle_handle_t handle)	513
5.25	Telematics_net	514
5.25.1	Data Structure Documentation	514
5.25.1.1	class telux::data::net::IFirewallManager	514
5.25.1.2	class telux::data::net::IFirewallEntry	519
5.25.1.3	struct telux::data::net::NatConfig	520
5.25.1.4	class telux::data::net::INatManager	520
5.25.1.5	class telux::data::net::IVlanManager	523
<b>6</b>	<b>Namespace Documentation</b>	<b>528</b>
6.1	telux Namespace Reference	528
6.2	telux::audio Namespace Reference	528
6.2.1	Variable Documentation	530
6.2.1.1	INFINITE_DTMF_DURATION	530
6.2.1.2	INFINITE_TONE_DURATION	530
6.3	telux::common Namespace Reference	530
6.3.1	Typedef Documentation	530
6.3.1.1	ResponseCallback	530
6.3.2	Enumeration Type Documentation	530
6.3.2.1	ServiceStatus	530
6.4	telux::config Namespace Reference	531
6.5	telux::cv2x Namespace Reference	531
6.5.1	Typedef Documentation	532
6.5.1.1	CreateRxSubscriptionCallback	532
6.5.1.2	CreateTxSpsFlowCallback	532
6.5.1.3	CreateTxEventFlowCallback	533
6.5.1.4	CloseTxFlowCallback	533
6.5.1.5	CloseRxSubscriptionCallback	533
6.5.1.6	ChangeSpsFlowInfoCallback	534
6.5.1.7	RequestSpsFlowInfoCallback	534
6.5.1.8	ChangeEventFlowInfoCallback	534
6.5.1.9	RequestCapabilitiesCallback	535
6.5.1.10	RequestDataSessionSettingsCallback	535
6.5.1.11	UpdateTrustedUEListCallback	535
6.5.1.12	UpdateSrcL2InfoCallback	536
6.5.1.13	CreateTcpSocketCallback	536
6.5.1.14	CloseTcpSocketCallback	536
6.5.1.15	StartCv2xCallback	537
6.5.1.16	StopCv2xCallback	537
6.5.1.17	RequestCv2xStatusCallback	537
6.5.1.18	RequestCv2xStatusCallbackEx	538
6.5.1.19	UpdateConfigurationCallback	538



6.6	telux::data Namespace Reference	538
6.6.1	Data Structure Documentation	540
6.6.1.1	struct telux::data::EsplInfo	540
6.6.1.2	struct telux::data::IcmlInfo	540
6.6.1.3	struct telux::data::IPv4Info	540
6.6.1.4	struct telux::data::IPv6Info	541
6.6.1.5	struct telux::data::TcplInfo	541
6.6.1.6	struct telux::data::UdpInfo	541
6.6.2	Typedef Documentation	541
6.6.2.1	DataCallResponseCb	541
6.6.2.2	StatisticsResponseCb	542
6.6.2.3	DataCallListResponseCb	542
6.6.2.4	DataRestrictModeCb	542
6.6.2.5	TypeOfService	542
6.6.2.6	TrafficClass	542
6.6.2.7	FlowLabel	543
6.7	telux::data::net Namespace Reference	543
6.7.1	Typedef Documentation	543
6.7.1.1	FirewallStatusCb	543
6.7.1.2	FirewallEntriesCb	543
6.7.1.3	DmzEntriesCb	544
6.7.1.4	StaticNatEntriesCb	544
6.7.1.5	CreateVlanCb	544
6.7.1.6	QueryVlanResponseCb	545
6.7.1.7	VlanMappingResponseCb	545
6.8	telux::loc Namespace Reference	545
6.9	telux::power Namespace Reference	547
6.10	telux::rsp Namespace Reference	547
6.10.1	Typedef Documentation	548
6.10.1.1	ProfileListResponseCb	548
6.11	telux::tel Namespace Reference	548
6.11.1	Typedef Documentation	551
6.11.1.1	MakeCallCallback	551
6.11.1.2	PinOperationResponseCb	552
6.11.1.3	QueryFdnLockResponseCb	552
6.11.1.4	QueryPin1LockResponseCb	552
6.11.1.5	EidResponseCallback	553
6.11.1.6	RatMask	553
6.11.1.7	SelectionModeResponseCallback	553
6.11.1.8	PreferredNetworksCallback	553
6.11.1.9	NetworkScanCallback	554
6.11.1.10	VoiceRadioTechResponseCb	554
6.11.1.11	CellInfoCallback	554
6.11.1.12	ECallGetOperatingModeCallback	555
6.11.1.13	RATCapabilitiesMask	555
6.11.1.14	VoiceServiceTechnologiesMask	555
6.11.1.15	SapStateResponseCallback	555
6.11.1.16	RatPreference	555
6.11.1.17	RatPreferenceCallback	555
6.11.1.18	ServiceDomainPreferenceCallback	556

6.12	telux::therm Namespace Reference	556
<b>7</b>	<b>Data Structure Documentation</b>	<b>558</b>
7.1	telux::cv2x::ICv2xListener Class Reference	558
7.1.1	Constructors and Destructors	558
7.1.1.1	~ICv2xListener()	558
7.1.2	Member Function Documentation	558
7.1.2.1	onStatusChanged(Cv2xStatus status)	558
7.1.2.2	onStatusChanged(Cv2xStatusEx status)	559
7.2	telux::data::IEspFilter Class Reference	559
7.2.1	Constructors and Destructors	559
7.2.1.1	~IEspFilter()	559
7.2.2	Member Function Documentation	559
7.2.2.1	getEspInfo()=0	559
7.2.2.2	setEspInfo(const EspInfo &espInfo)=0	560
7.3	telux::data::IlcmpFilter Class Reference	560
7.3.1	Constructors and Destructors	560
7.3.1.1	~IlcmpFilter()	560
7.3.2	Member Function Documentation	560
7.3.2.1	getIcmpInfo()=0	560
7.3.2.2	setIcmpInfo(const IcmpInfo &icmpInfo)=0	561
7.4	telux::data::IIPFilter Class Reference	561
7.4.1	Constructors and Destructors	561
7.4.1.1	~IIPFilter()	561
7.4.2	Member Function Documentation	562
7.4.2.1	getIPv4Info()=0	562
7.4.2.2	setIPv4Info(const IPv4Info &ipv4Info)=0	562
7.4.2.3	getIPv6Info()=0	562
7.4.2.4	setIPv6Info(const IPv6Info &ipv6Info)=0	562
7.4.2.5	getIpProtocol()=0	563
7.5	telux::common::IServiceStatusListener Class Reference	563
7.5.1	Constructors and Destructors	563
7.5.1.1	~IServiceStatusListener()	563
7.5.2	Member Function Documentation	563
7.5.2.1	onServiceStatusChange(ServiceStatus status)	563
7.6	telux::data::ITcpFilter Class Reference	564
7.6.1	Constructors and Destructors	564
7.6.1.1	~ITcpFilter()	564
7.6.2	Member Function Documentation	564
7.6.2.1	getTcpInfo()=0	564
7.6.2.2	setTcpInfo(const TcpInfo &tcpInfo)=0	564
7.7	telux::data::IUDPFilter Class Reference	565
7.7.1	Constructors and Destructors	565
7.7.1.1	~IUDPFilter()	565
7.7.2	Member Function Documentation	565
7.7.2.1	getUdpInfo()=0	565
7.7.2.2	setUdpInfo(const UdpInfo &udpInfo)=0	565

## List of Figures

2-1	SDK-Overview	7
3-1	Phone manager initialization	16
3-2	Dial call flow	17
3-3	ECall call flow	18
3-4	Signal strength call flow	20
3-5	Answer, Reject, RejectWithSMS call flow	21
3-6	Hold call flow	22
3-7	Hold, Conference, Swap call flow	23
3-8	SMS call flow	25
3-9	Get applications call flow	26
3-10	On logical channel	27
3-11	On basic channel	28
3-12	Request card reader status, Request ATR, Transmit APDU call flow	29
3-13	SIM Turn off, Turn on and Reset call flow	31
3-14	Radio and Service state call flow	33
3-15	Subscription initialization call flow	34
3-16	Subscription call flow	35
3-17	Call flow to register/remove listener for generating basic reports	36
3-18	Call flow to register/remove listener for generating detailed reports	38
3-19	Call flow to register/remove listener for generating detailed engine reports	39
3-20	Call flow to enable/disable constraint time uncertainty	40
3-21	Start/Stop for data connection manager call flow	41
3-22	Request/Create/Delete/Modify for data profile manager call flow	43
3-23	Get/Set data filter mode call flow	45
3-24	Add data restrict filter call flow	46
3-25	Remove data restrict filter call flow	47
3-26	Network selection manager call flow	49
3-27	Serving System Manager Call Flow	51
3-28	Start/Stop C-V2X Mode call flow	53
3-29	C-V2X Radio Manager call flow	54
3-30	C-V2X Radio Initialization call flow	55
3-31	C-V2X Radio RX subscription call flow	56
3-32	C-V2X Radio TX event-driven flow call flow	57
3-33	C-V2X Radio SPS flow call flow	58
3-34	Audio Manager API call flow	60
3-35	Audio Voice Call Start/Stop call flow	62
3-36	Audio Voice Call Device Switch call flow	64
3-37	Audio Voice Call Volume/Mute control call flow	66
3-38	Call flow to play DTMF tone	68
3-39	Call flow to detect DTMF tone	70
3-40	Audio Playback call flow	72
3-41	Audio Capture call flow	74
3-42	Call flow to play/stop tone on a sink device	76
3-43	Call flow to start/stop loopback between source and sink devices	77
3-44	Call flow to play Compressed audio format	79
3-45	Audio Transcoding Operation Callflow	81

3-46 Compressed audio format playback on Voice Paths Callflow . . . . .	83
3-47 Thermal manager call flow . . . . .	85
3-48 Call flow to register/remove listener for Thermal shutdown manager . . . . .	86
3-49 Call flow to set/get the Thermal auto-shutdown mode . . . . .	87
3-50 Call flow to manage thermal auto-shutdown from an eCall application . . . . .	88
3-51 Call flow to register/remove listener for TCU-activity manager . . . . .	90
3-52 Call flow to set the TCU-activity state . . . . .	91
3-53 Remote SIM call flow . . . . .	92
3-54 Modem Config load and activate call flow . . . . .	94
3-55 Modem Config deactivate and delete Call Flow . . . . .	95
3-56 Modem Config get and set Auto Selection Mode Call Flow . . . . .	96

# 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 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

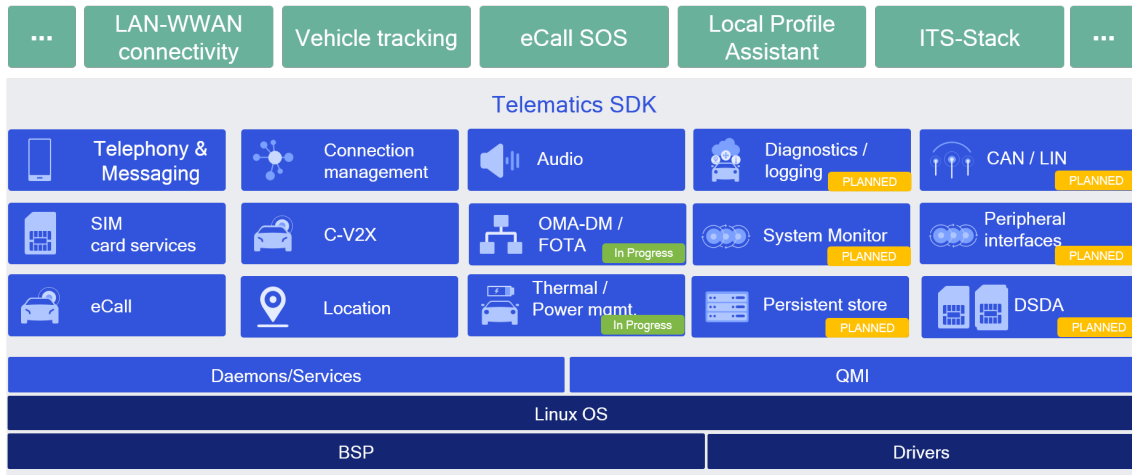


Figure 2-1 SDK-Overview

### 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 etc. Also constraint Time Uncertainty sub-system consists of API to enable/disable constraint time uncertainty 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 Mangement

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.

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 and jammer signals. 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. Location constraint time uncertainty API of Telematics SDK provide the mechanism to enable/disable constraint time uncertainty feature.

## 2.2.6 Data Services

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

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

## 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.

# 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

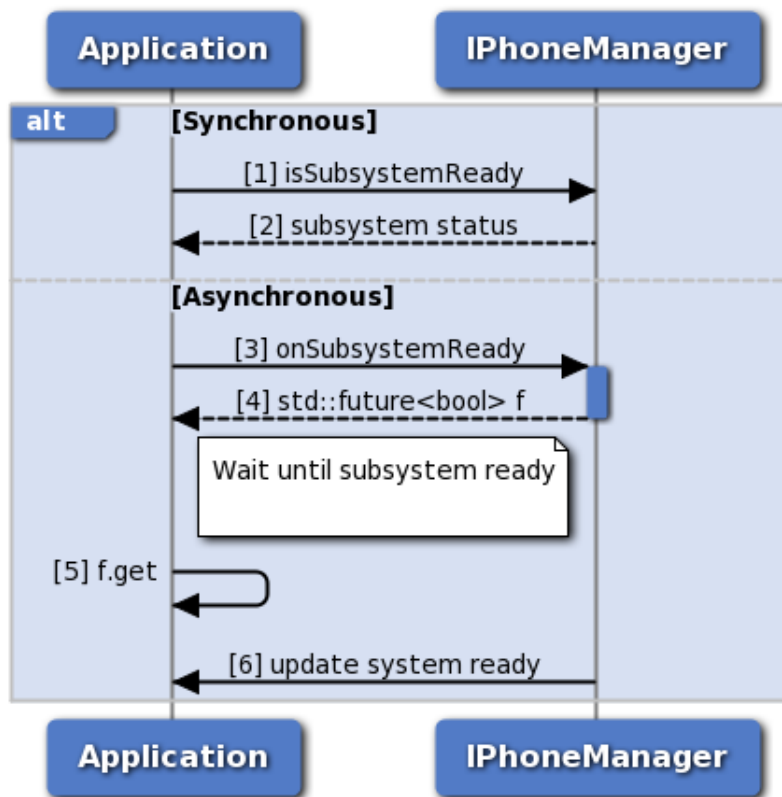


Figure 3-1 Phone manager initialization

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 Dial call flow

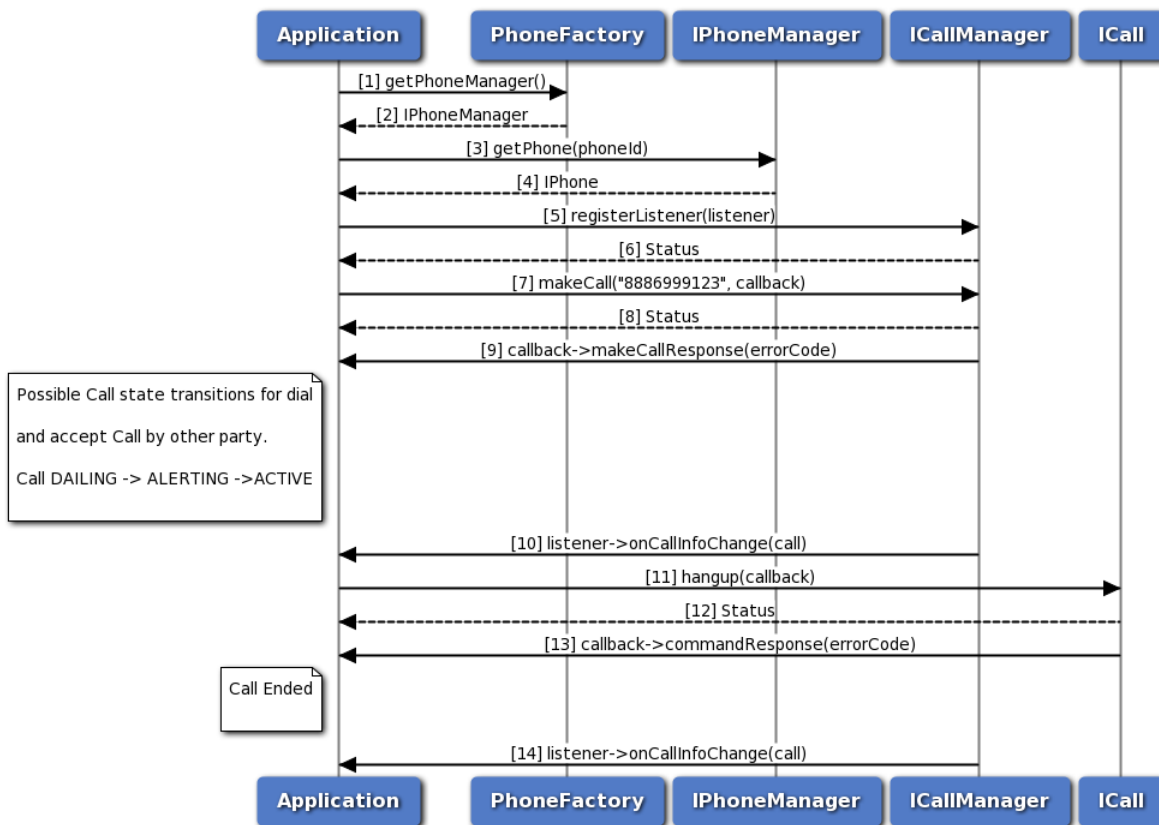


Figure 3-2 Dial call flow

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.

### 3.3 ECall call flow

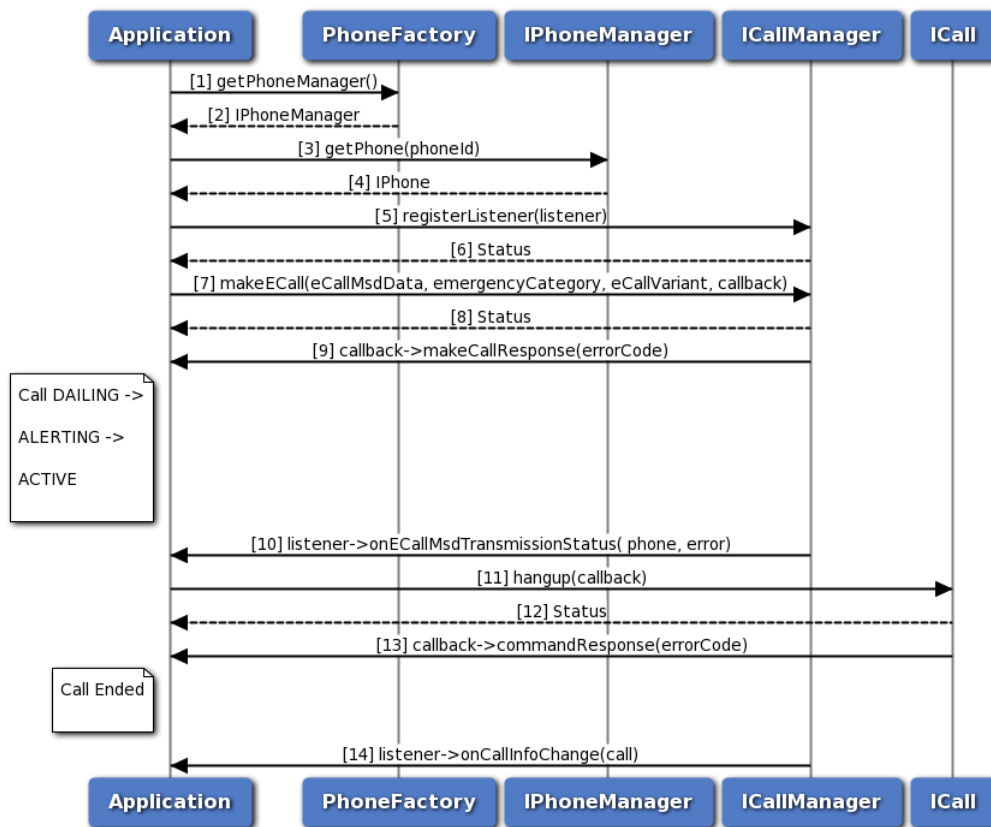


Figure 3-3 ECall call flow

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.

### 3.4 Signal strength call flow

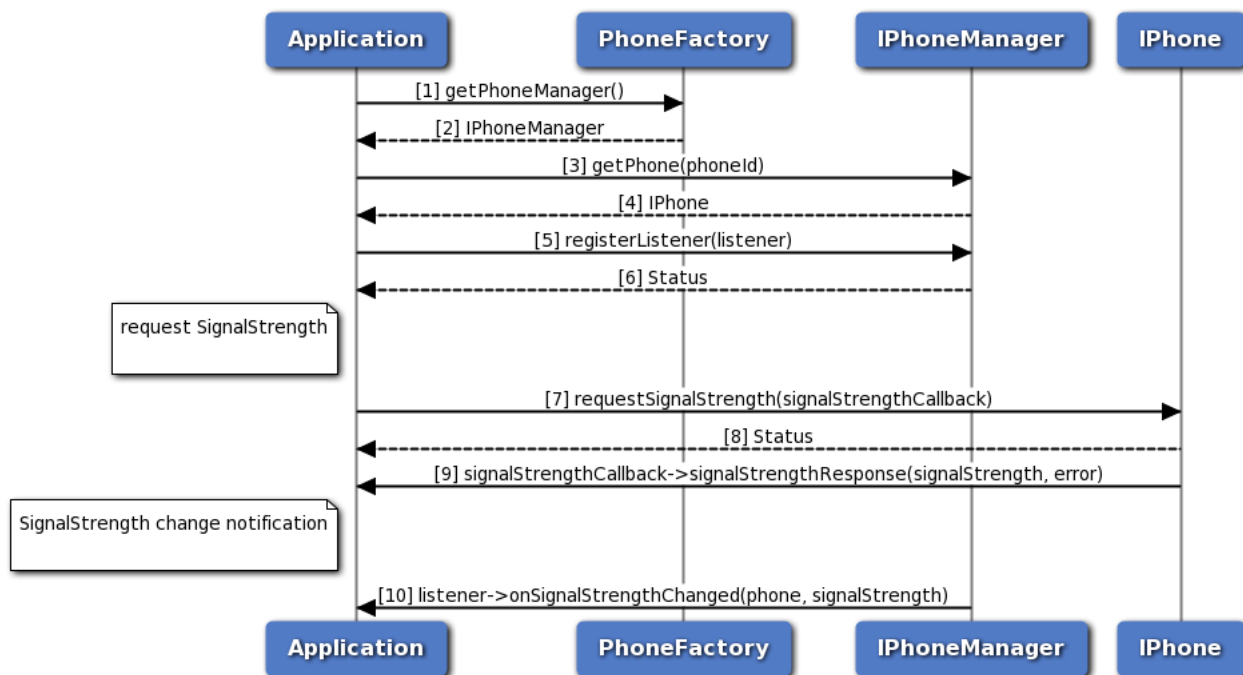


Figure 3-4 Signal strength call flow

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.

### 3.5 Answer, Reject, RejectWithSMS call flow

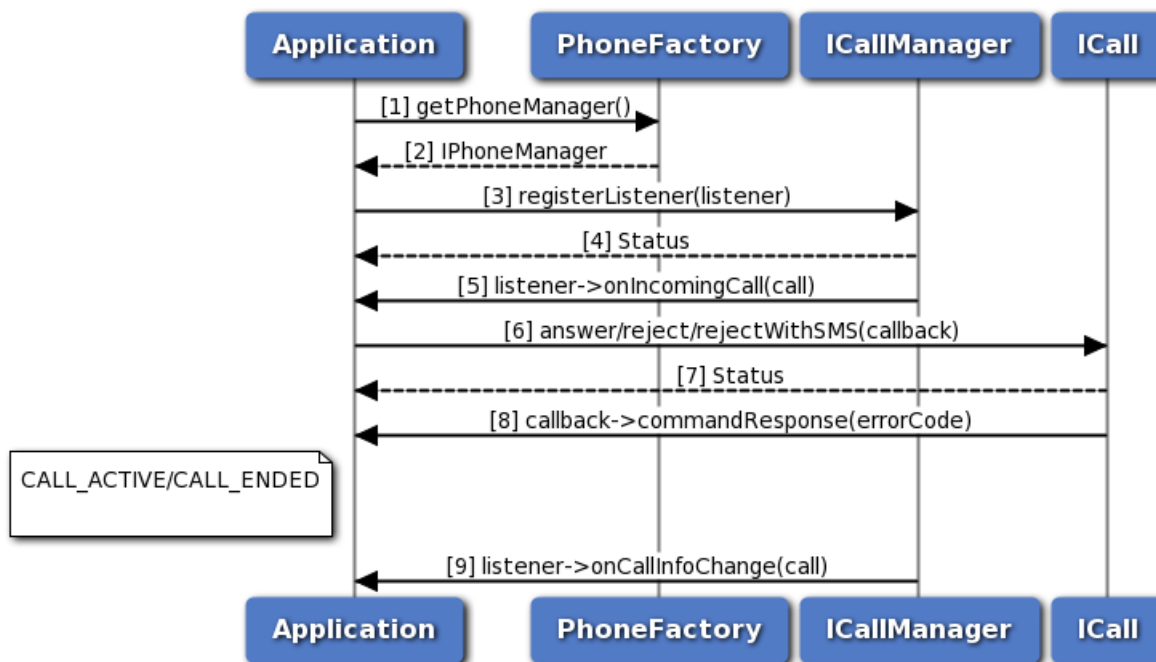
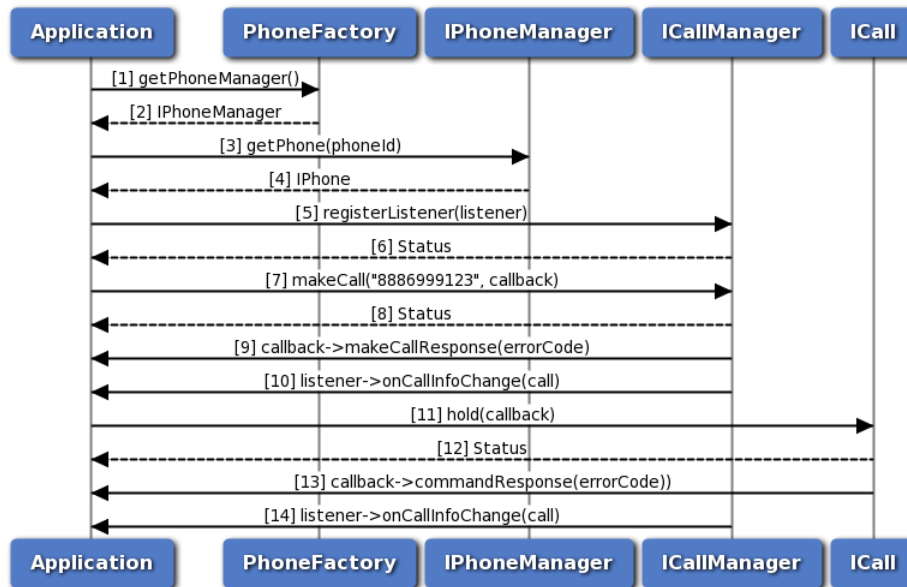


Figure 3-5 Answer, Reject, RejectWithSMS call flow

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.

### 3.6 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.

### 3.7 Hold, Conference, Swap call flow

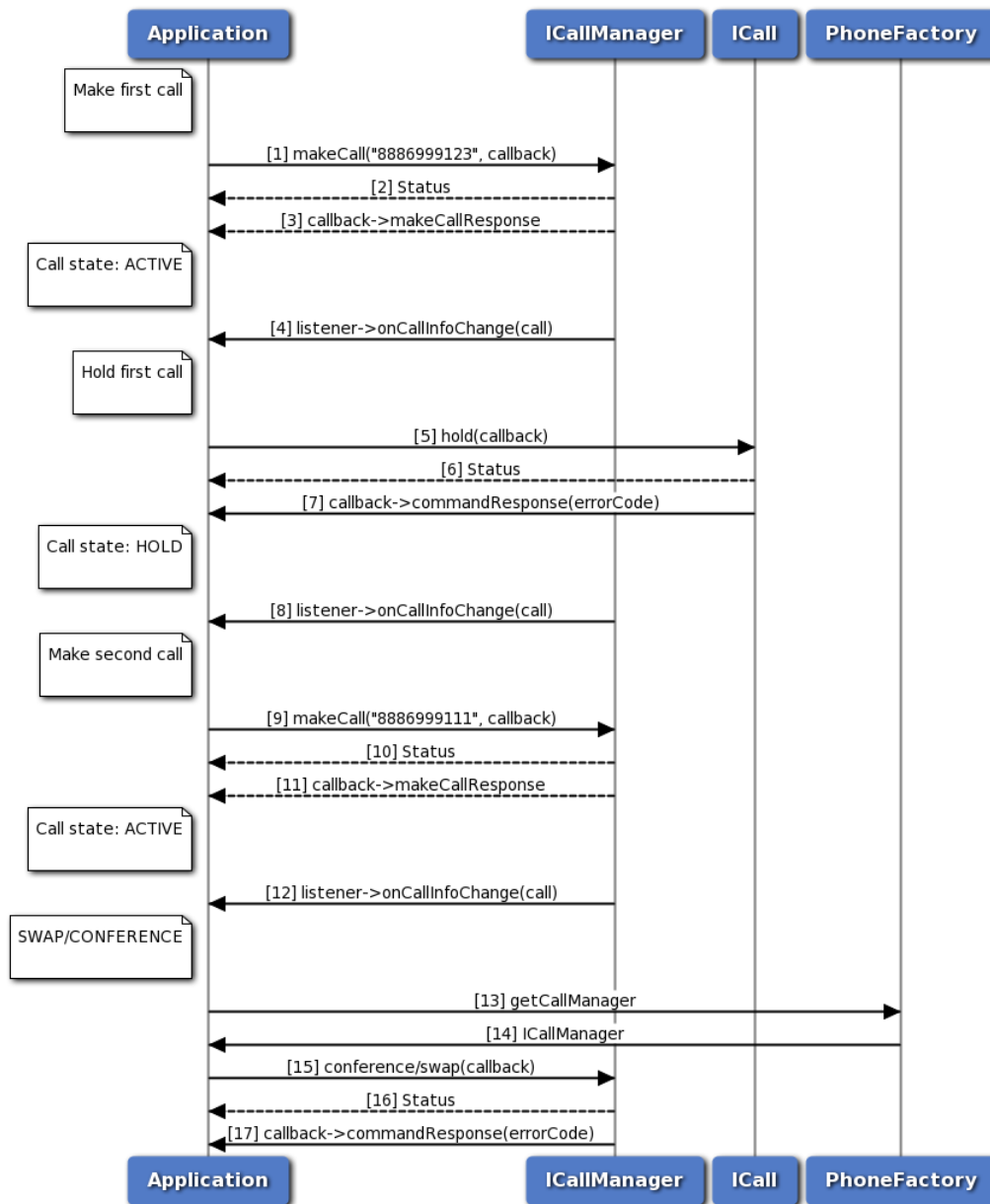


Figure 3-7 Hold, Conference, Swap call flow

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.



### 3.8 SMS call flow

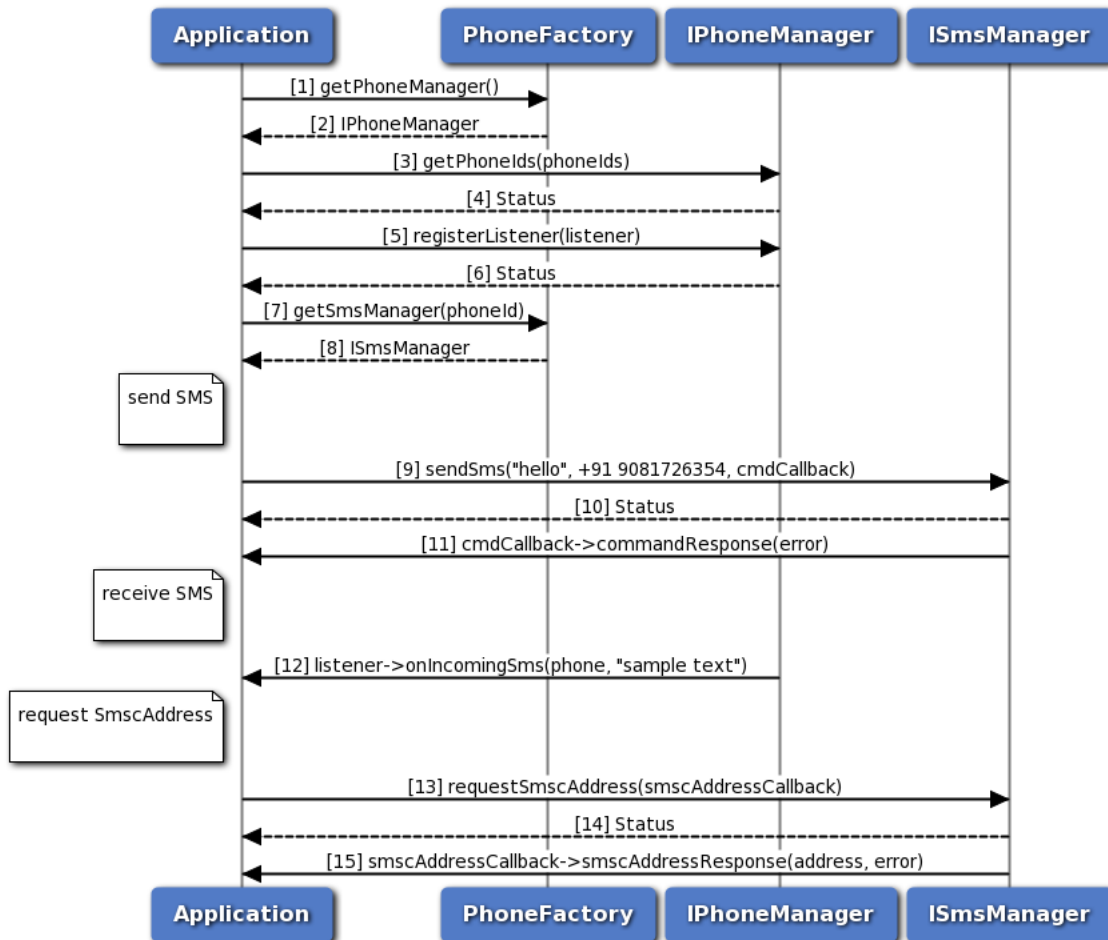


Figure 3-8 SMS call flow

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.

### 3.9 Get applications call flow

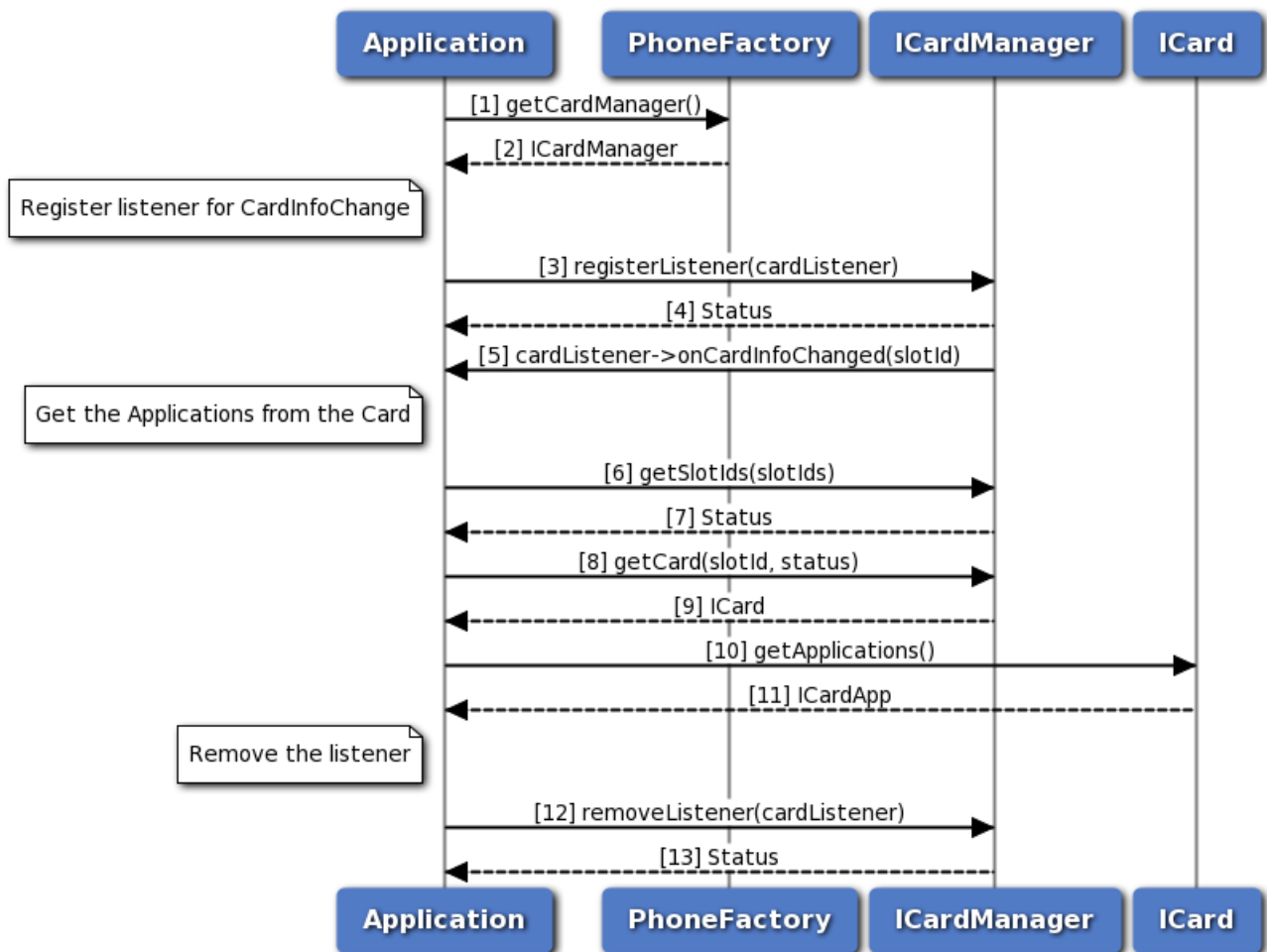


Figure 3-9 Get applications call flow

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.

## 3.10 Transmit APDU call flow

### 3.10.1 On logical channel

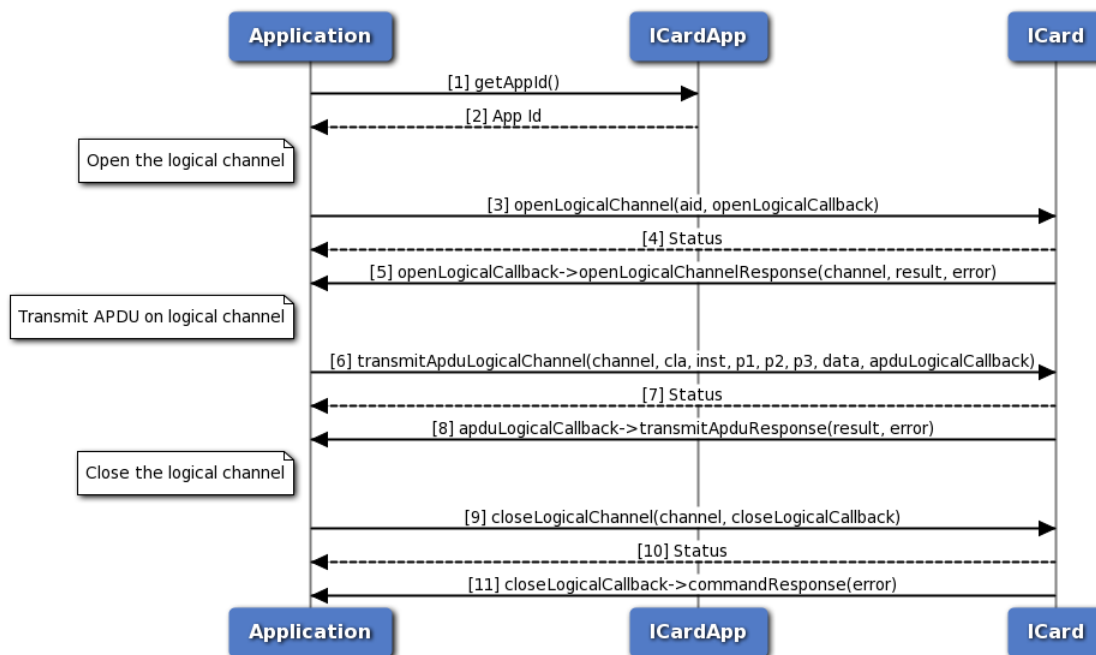


Figure 3-10 On logical channel

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 `TransmitApduResponseCallback`.
7. The application receives the status i.e. either `SUCCESS` or `FAILED` based on execution of `transmitApduLogicalChannel` 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.

### 3.10.2 On basic channel

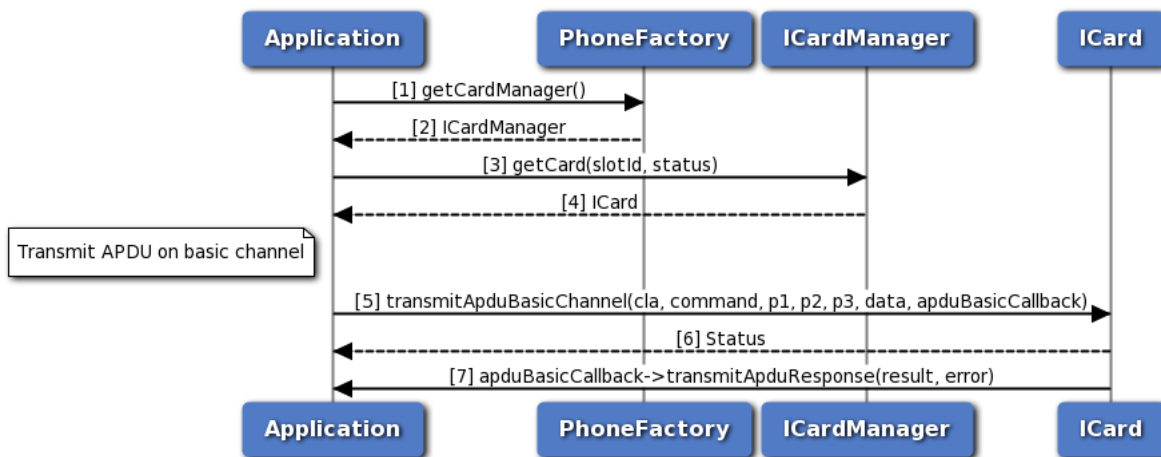


Figure 3-11 On basic channel

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.

- Optionally, the application receives the response which contains either result on success or error in case of failure.

### 3.11 SAP card manager call flow

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

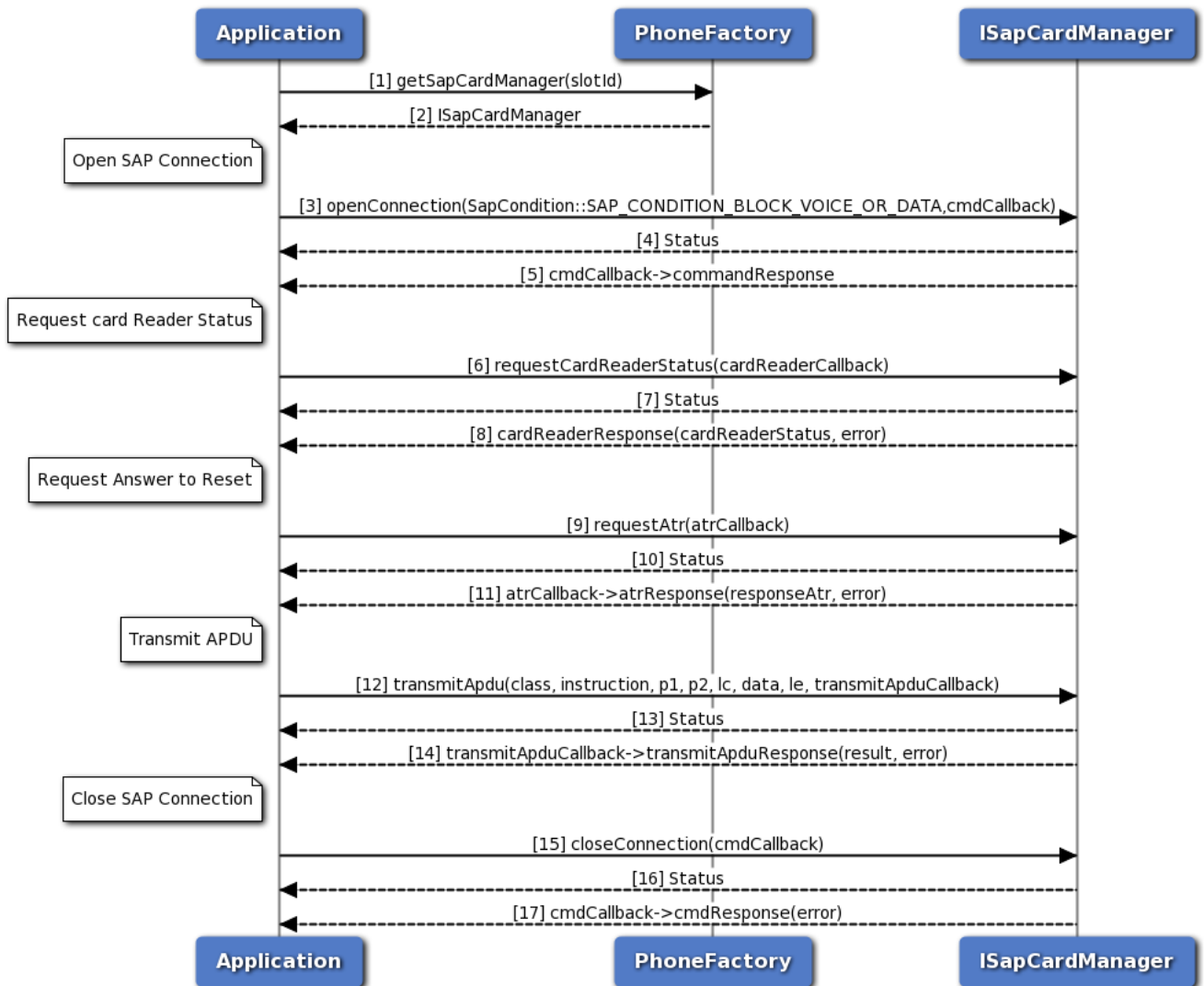


Figure 3-12 Request card reader status, Request ATR, Transmit APDU call flow

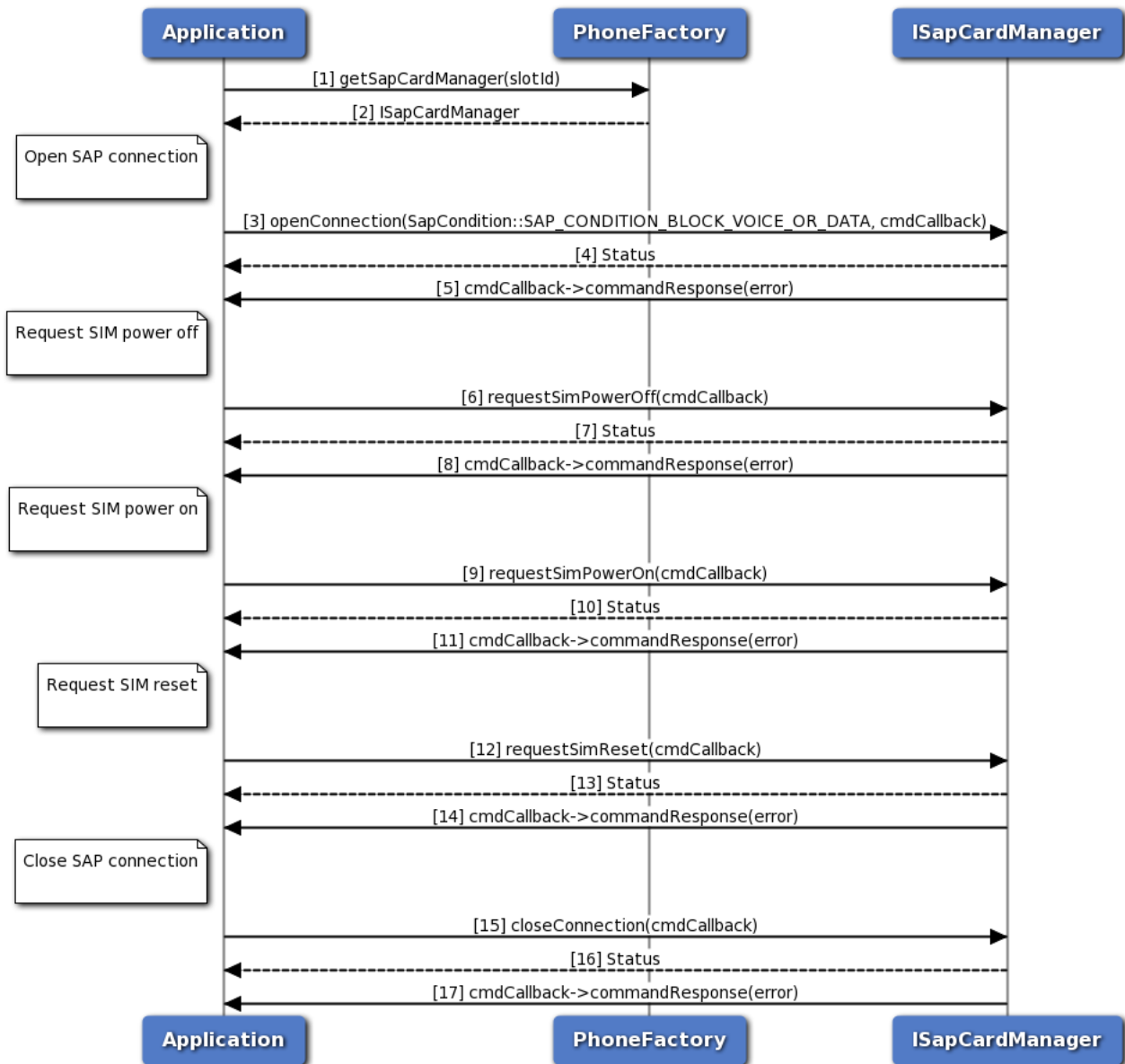
{ sap\_card\_operations.png, Request card reader status, Request ATR, Transmit APDU call flow, 80, 80, Request card reader status, Request ATR, Transmit APDU call flow }

- The application gets SapCardManager object corresponding to slotId using PhoneFactory.
- 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 IAtrResponseCallback.
10. The application receives the status i.e. either SUCCESS or FAILED based on execution of requestAtr 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.

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



**Figure 3-13 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.



### 3.12 Radio and Service state call flow

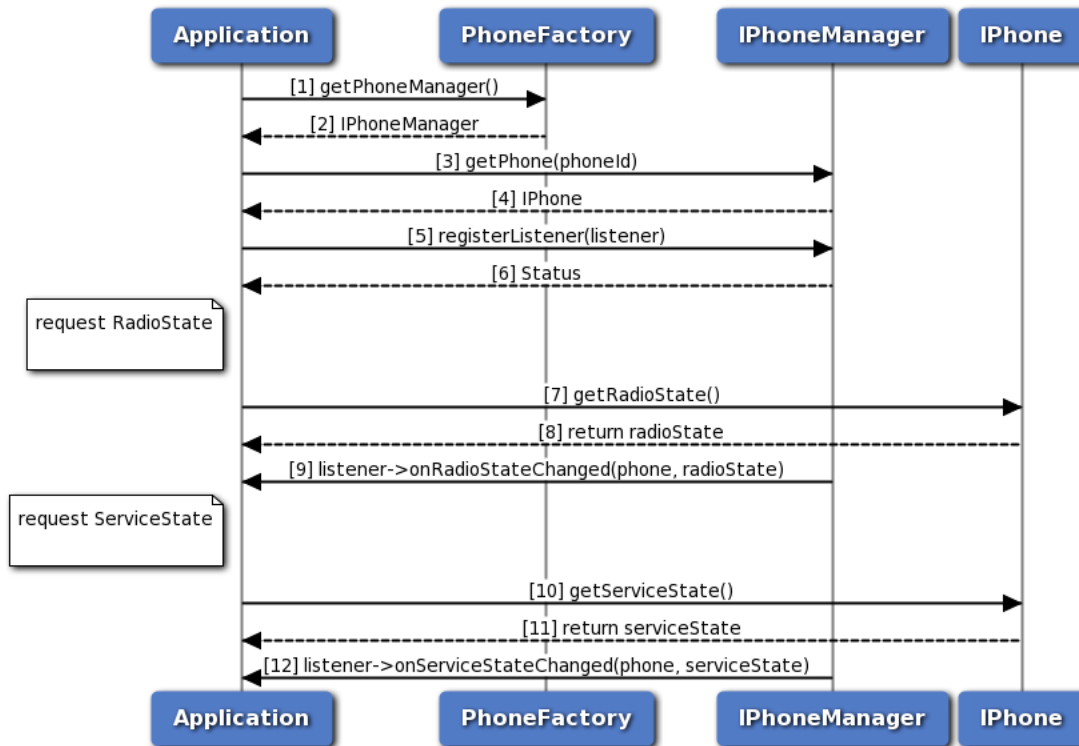


Figure 3-14 Radio and Service state call flow

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.

### 3.13 Subscription Call flow

### 3.13.1 Subscription initialization

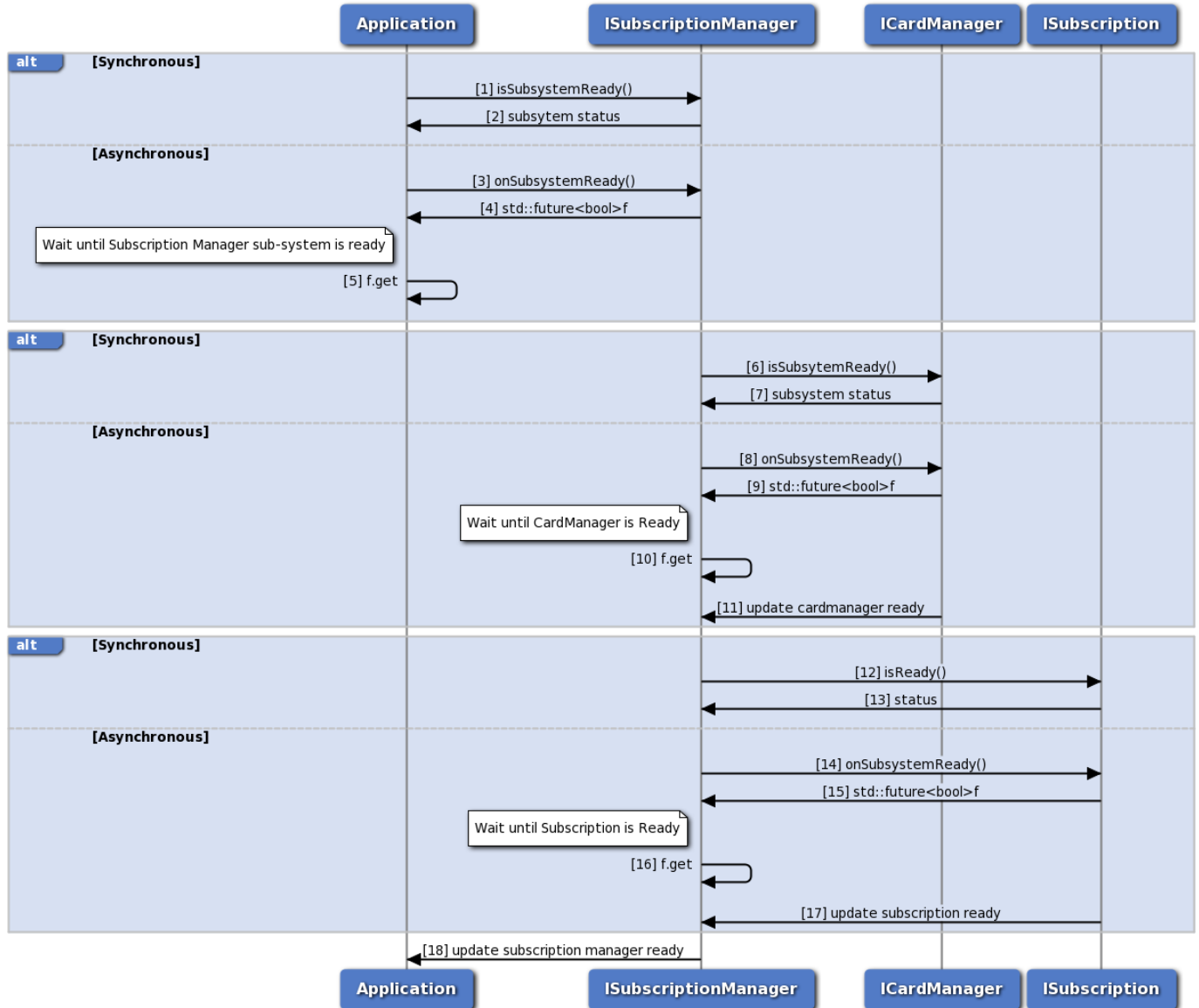


Figure 3-15 Subscription initialization call flow

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.

### 3.13.2 Subscription call flow

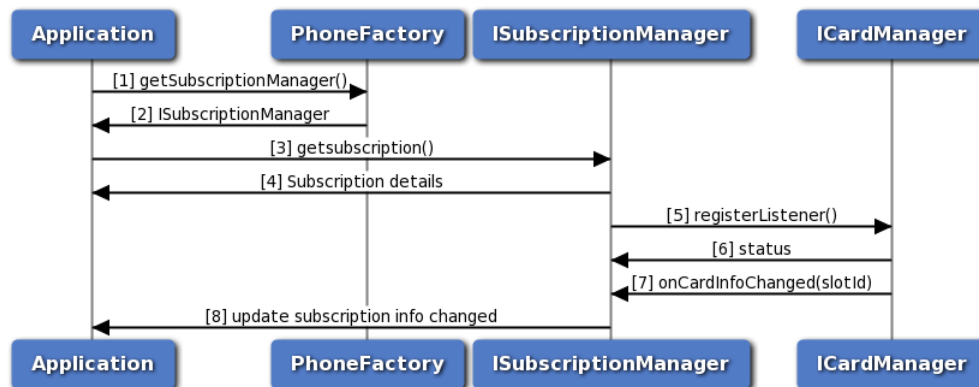


Figure 3-16 Subscription call flow

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.

### 3.14 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.

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

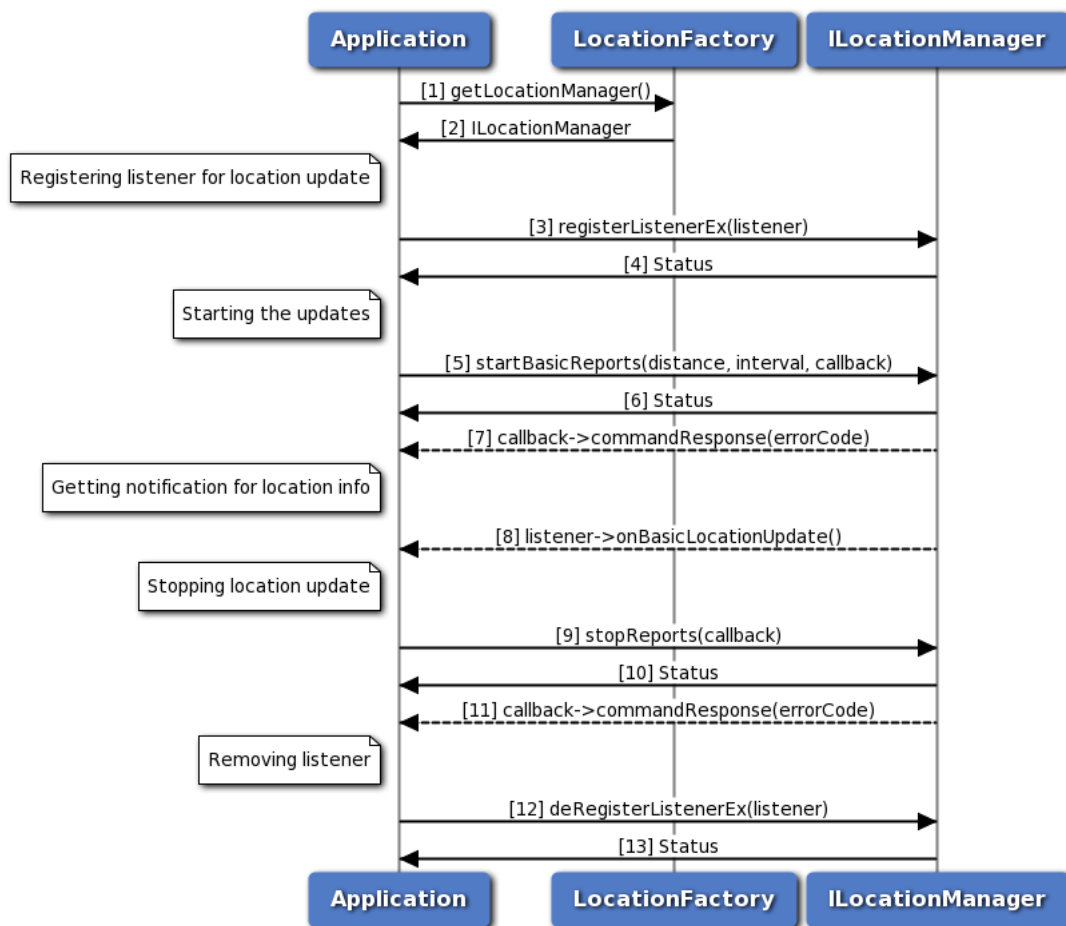
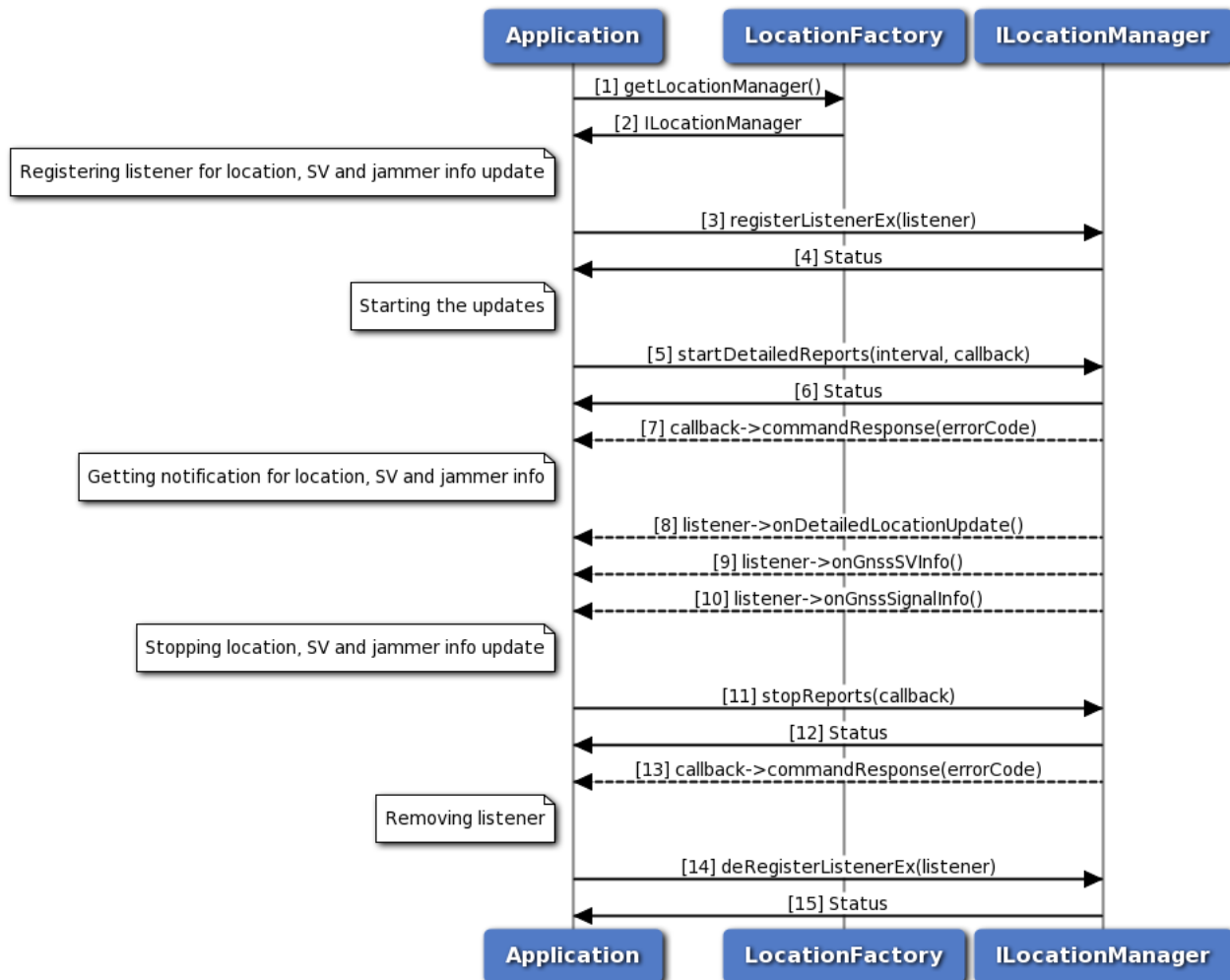


Figure 3-17 Call flow to register/remove listener for generating basic reports

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.

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



**Figure 3-18 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 and jammer signal 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 and jammer signal 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 jammer information etc.
11. Application stops receiving all the reports through stopReports API.
12. Status of stopReports i.e. either SUCCESS or FAILED will be returned to the application.
13. The response for stopReports is received by the application.
14. Application can remove listener and when the number of listeners are zero then location service will get stopped automatically.
15. Status of remove listener i.e. either SUCCESS or FAILED will be returned to the application.

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

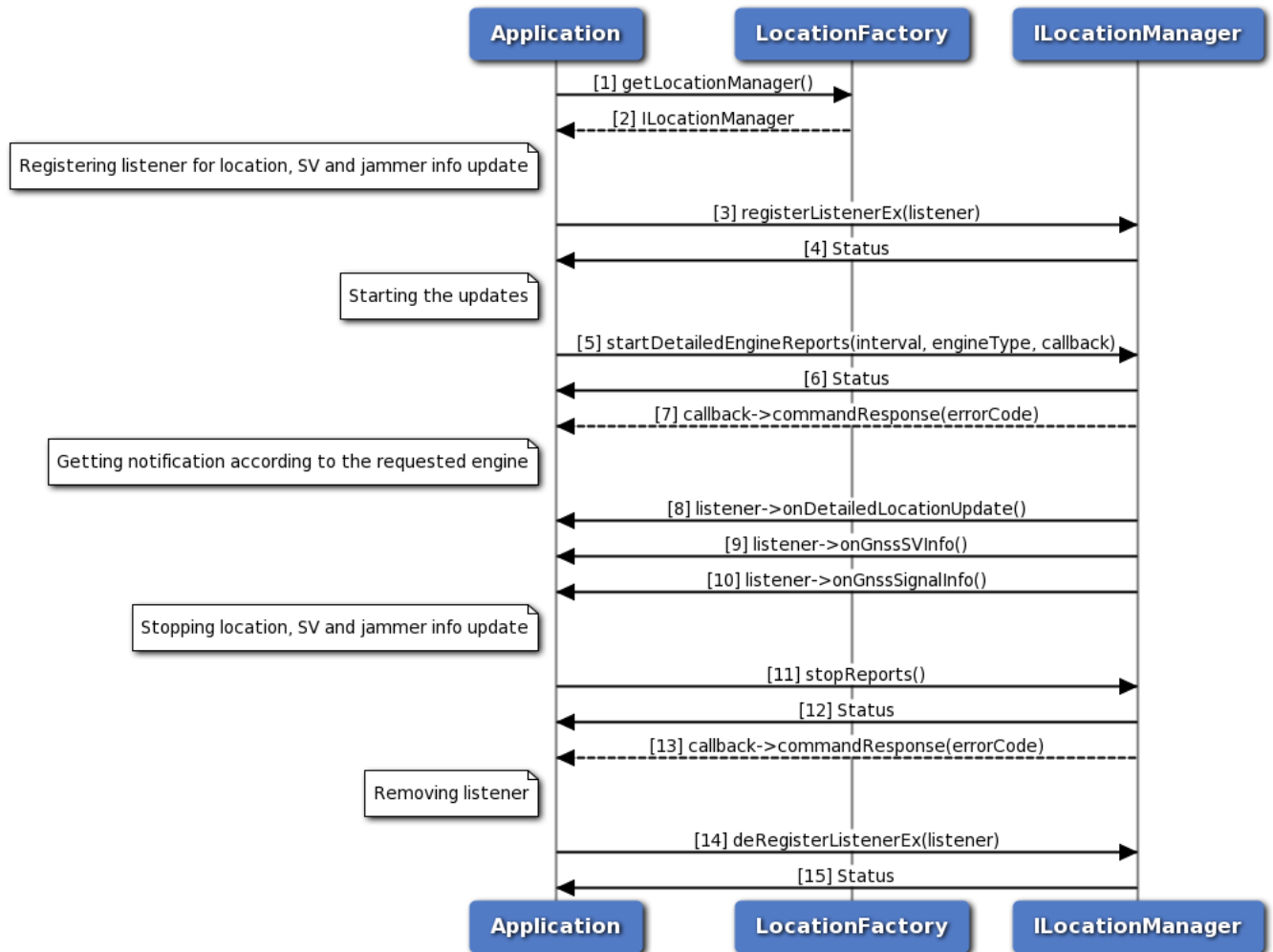
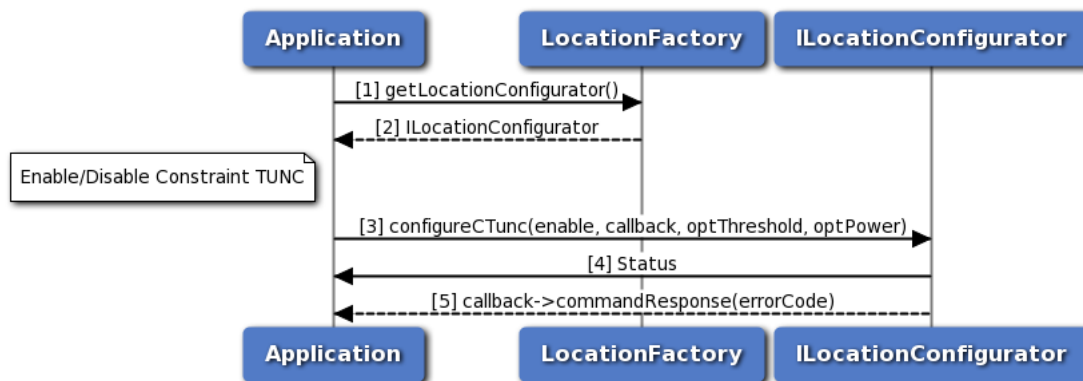


Figure 3-19 Call flow to register/remove listener for generating detailed engine reports

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 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 and jammer signal 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 jammer information etc depending on the requested SPE/PPE/Fused engine type.
11. Application stops receiving all the reports through stopReports API.
12. Status of stopReports i.e. either SUCCESS or FAILED will be returned to the application.
13. The response for stopReports is received by the application.
14. Application can remove listener and when the number of listeners are zero then location service will get stopped automatically.
15. Status of remove listener i.e. either SUCCESS or FAILED will be returned to the application.

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



**Figure 3-20 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.



## 3.15 Data Connection Manager Call Flow

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

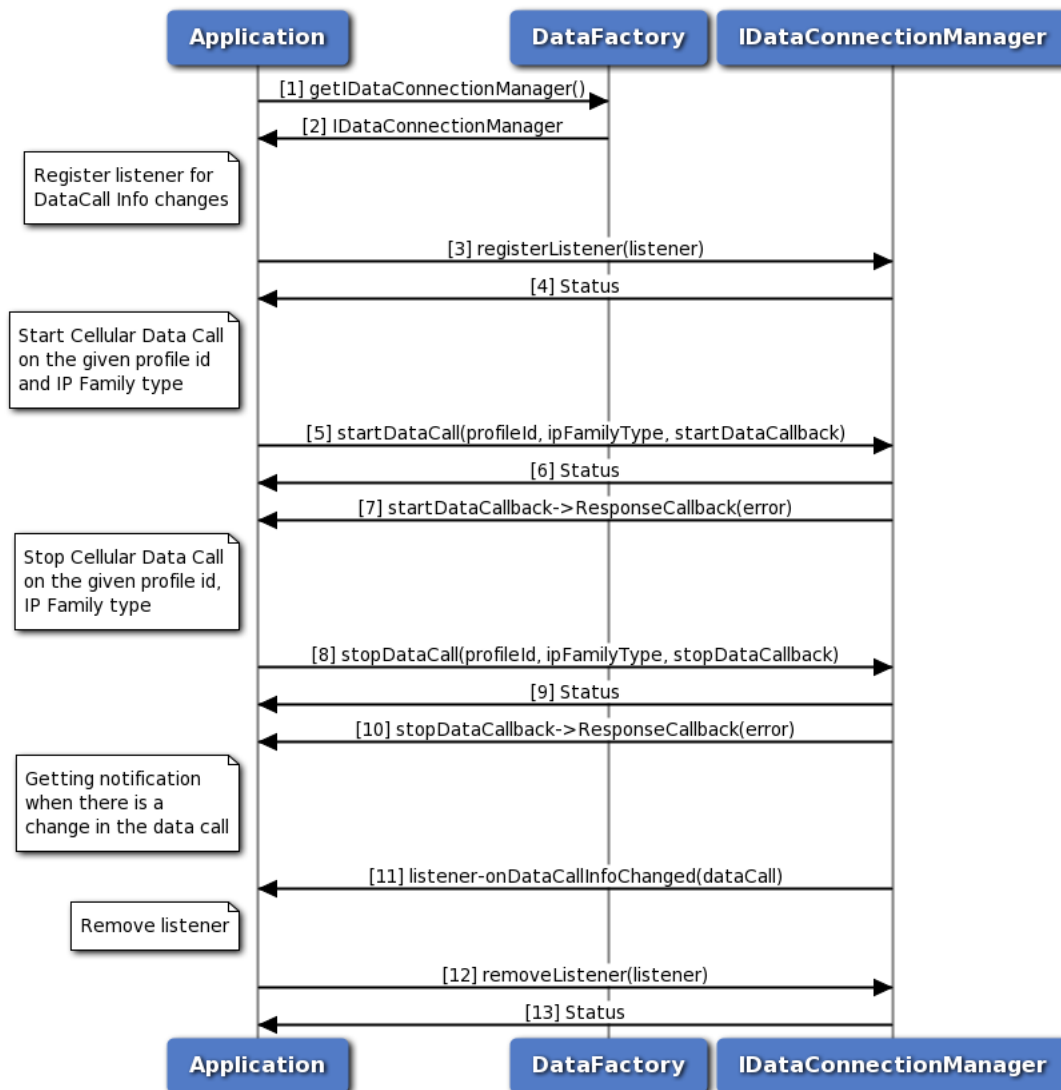


Figure 3-21 Start/Stop for data connection manager call flow

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.

### **3.16 Data Profile Manager Call Flow**

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

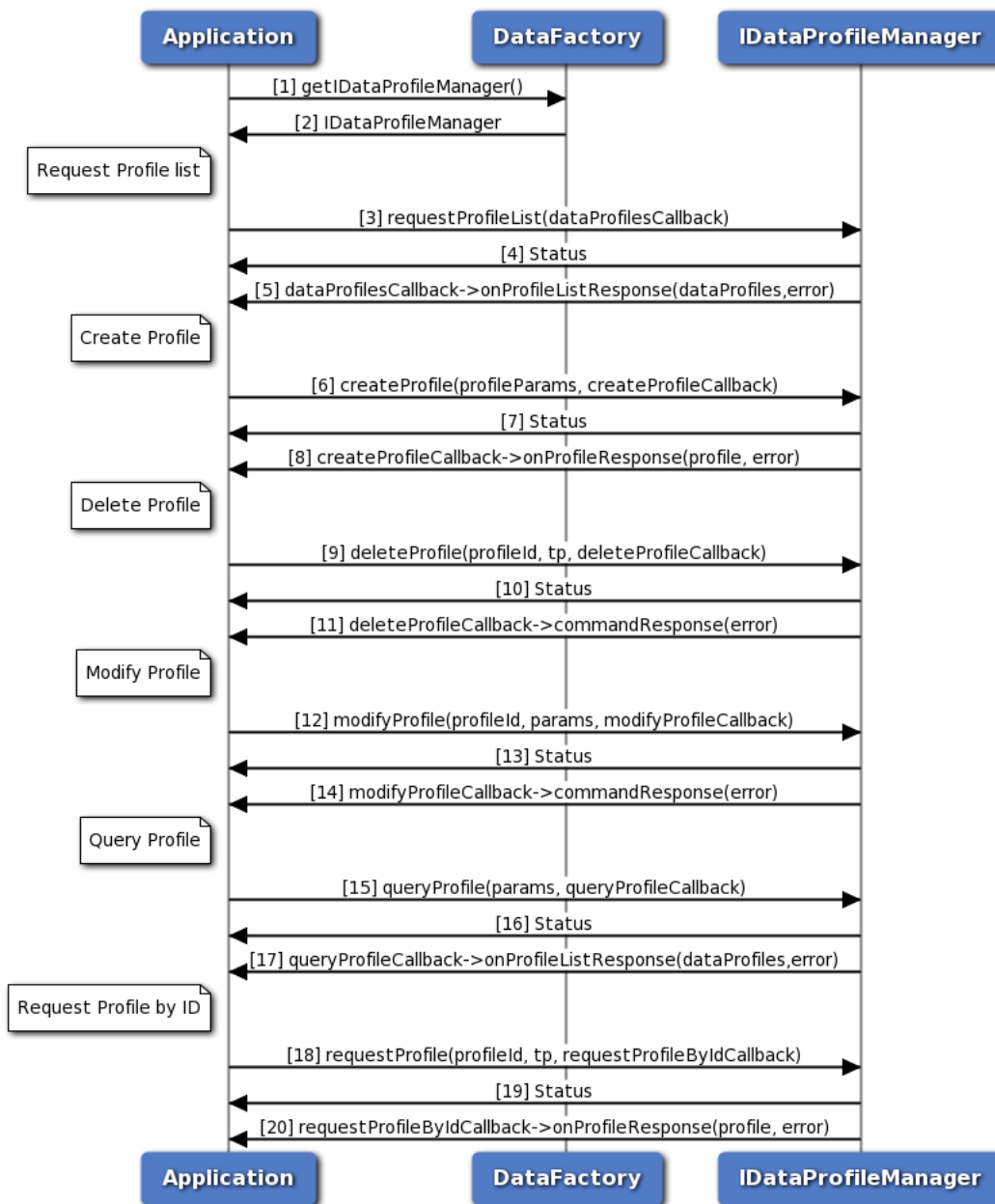


Figure 3-22 Request/Create/Delete/Modify for data profile manager call flow

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.

### 3.17 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.

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

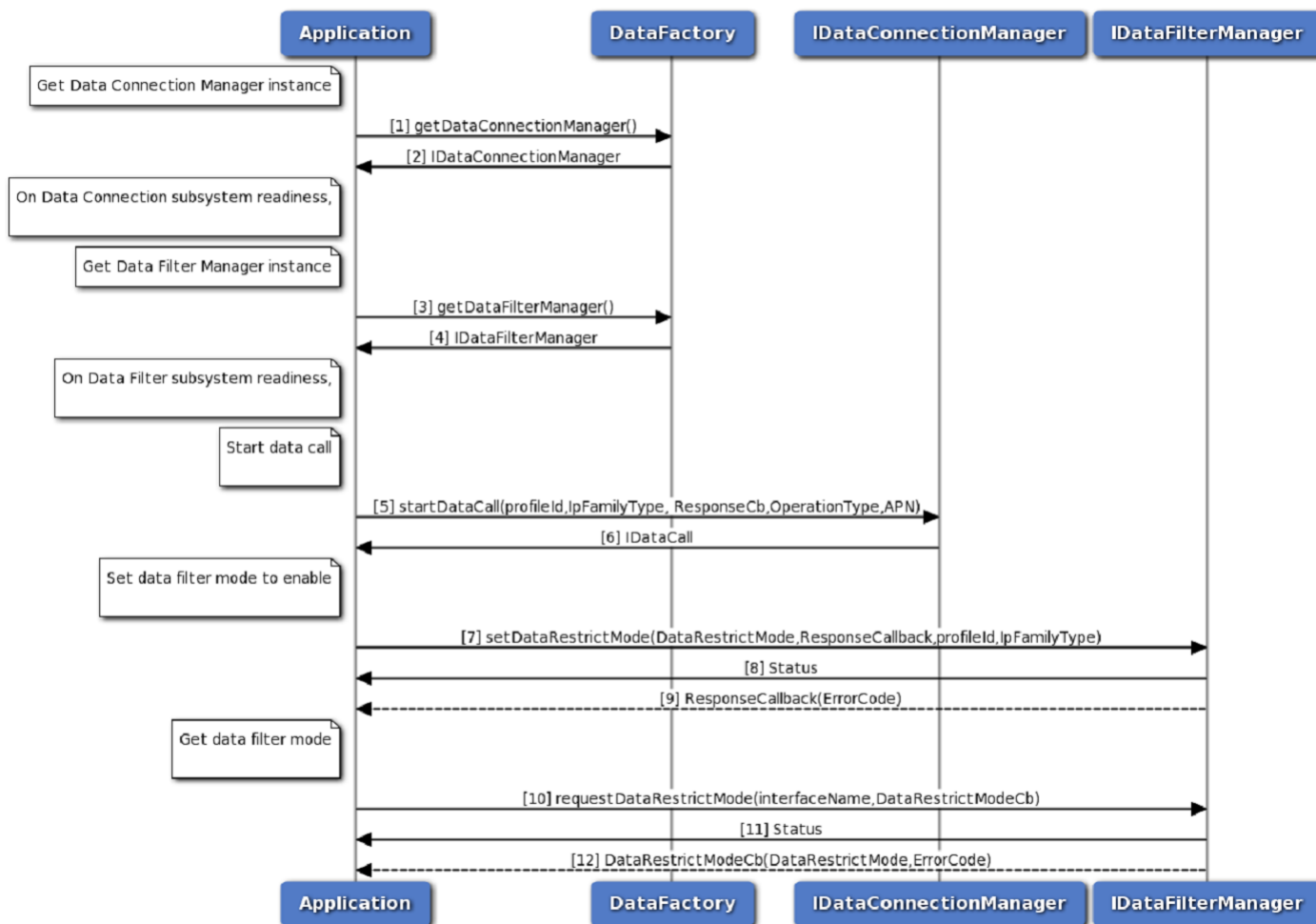


Figure 3-23 Get/Set data filter mode call flow

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.

### 3.17.2 Call flow to Add data restrict filter

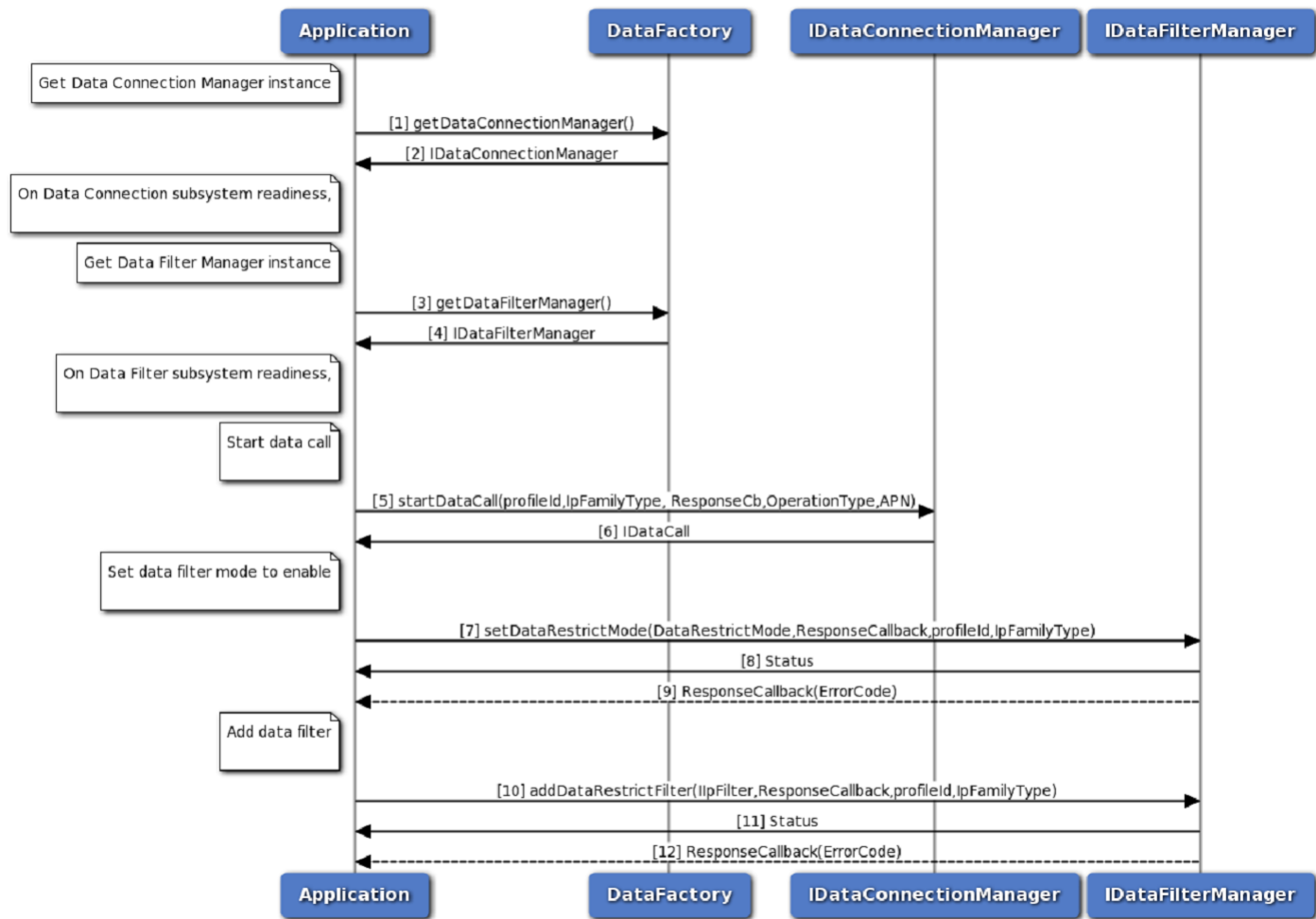


Figure 3-24 Add data restrict filter call flow

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.

### 3.17.3 Call flow to Remove data restrict filter

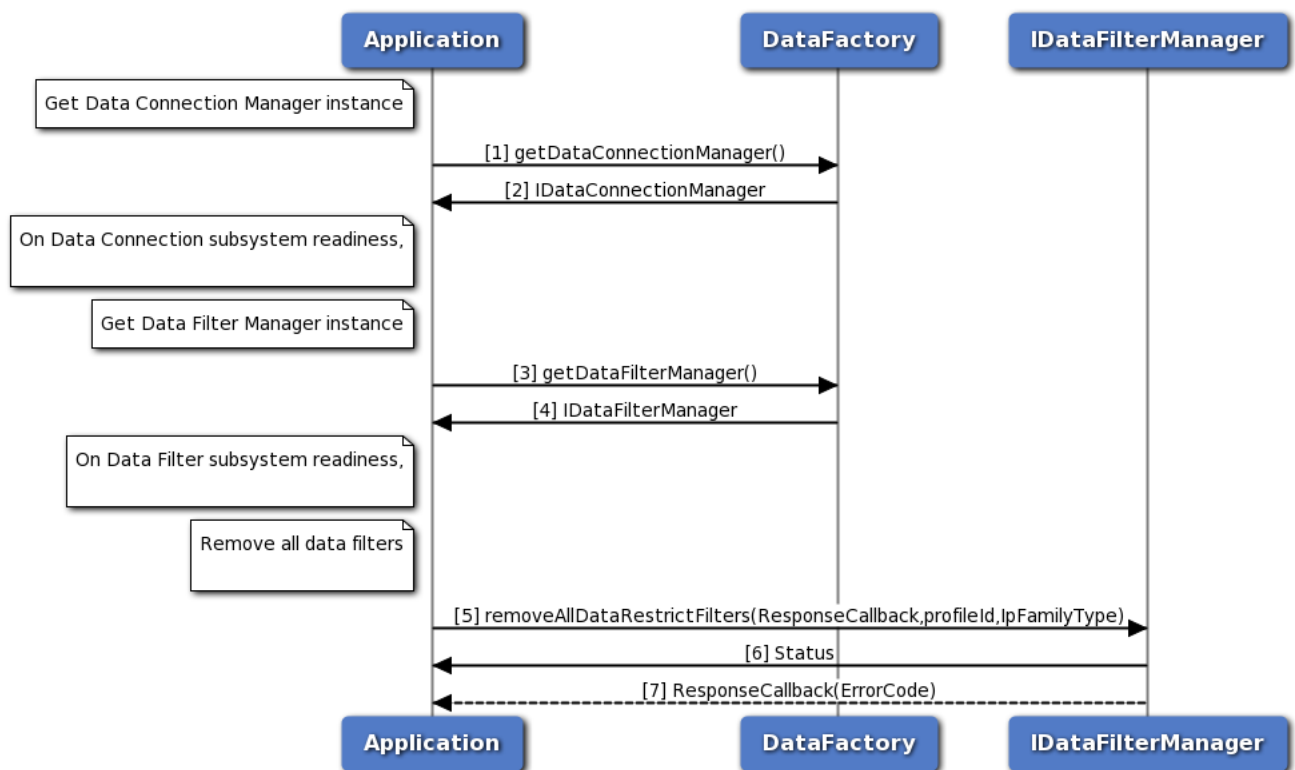


Figure 3-25 Remove data restrict filter call flow

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.

## **3.18 Network Selection Call Flow**

### **3.18.1 Network Selection Call Flow**

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



change in network selection mode.

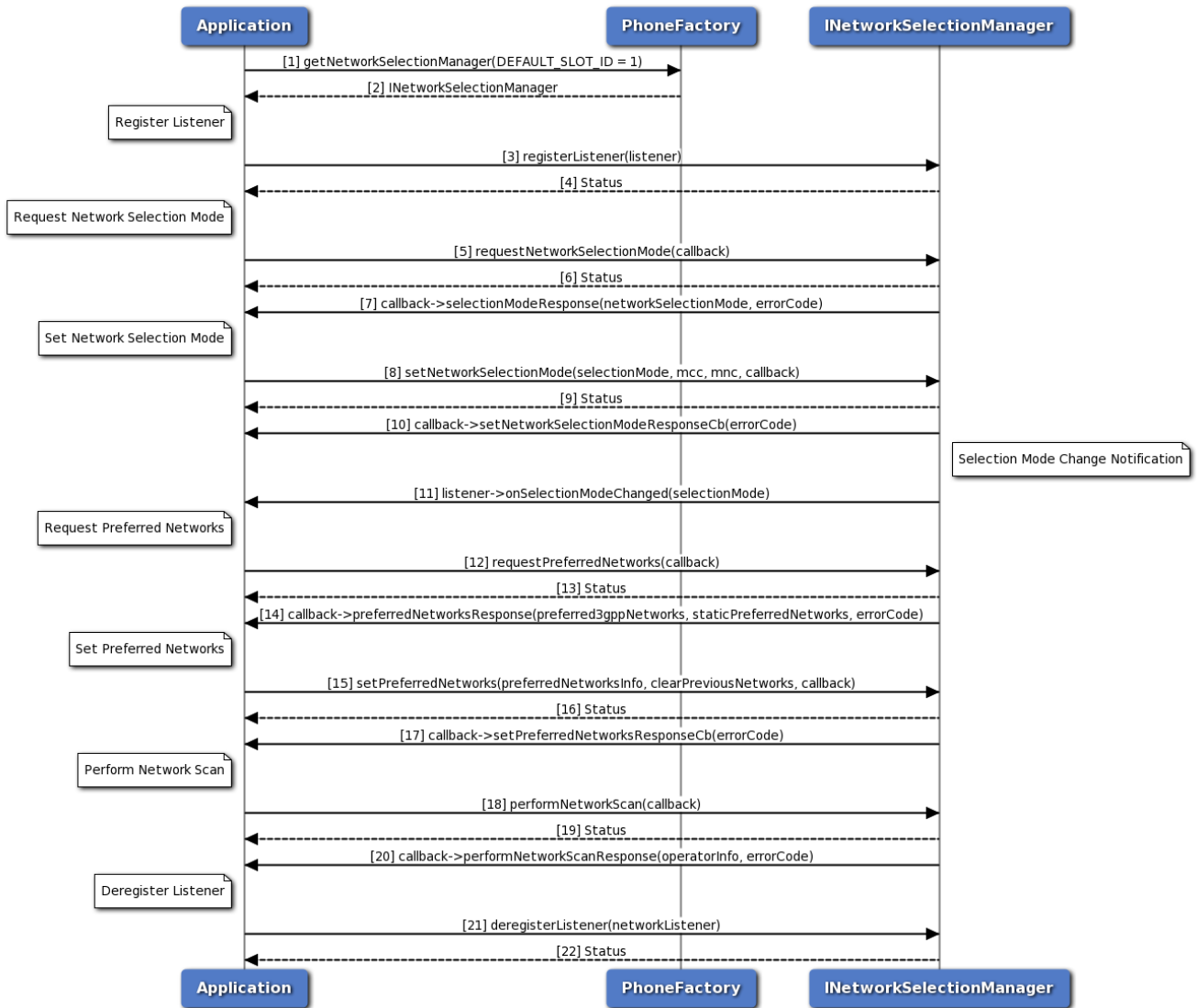


Figure 3-26 Network selection manager call flow

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.

### 3.19 Serving System Call Flow

### 3.19.1 Serving System 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.

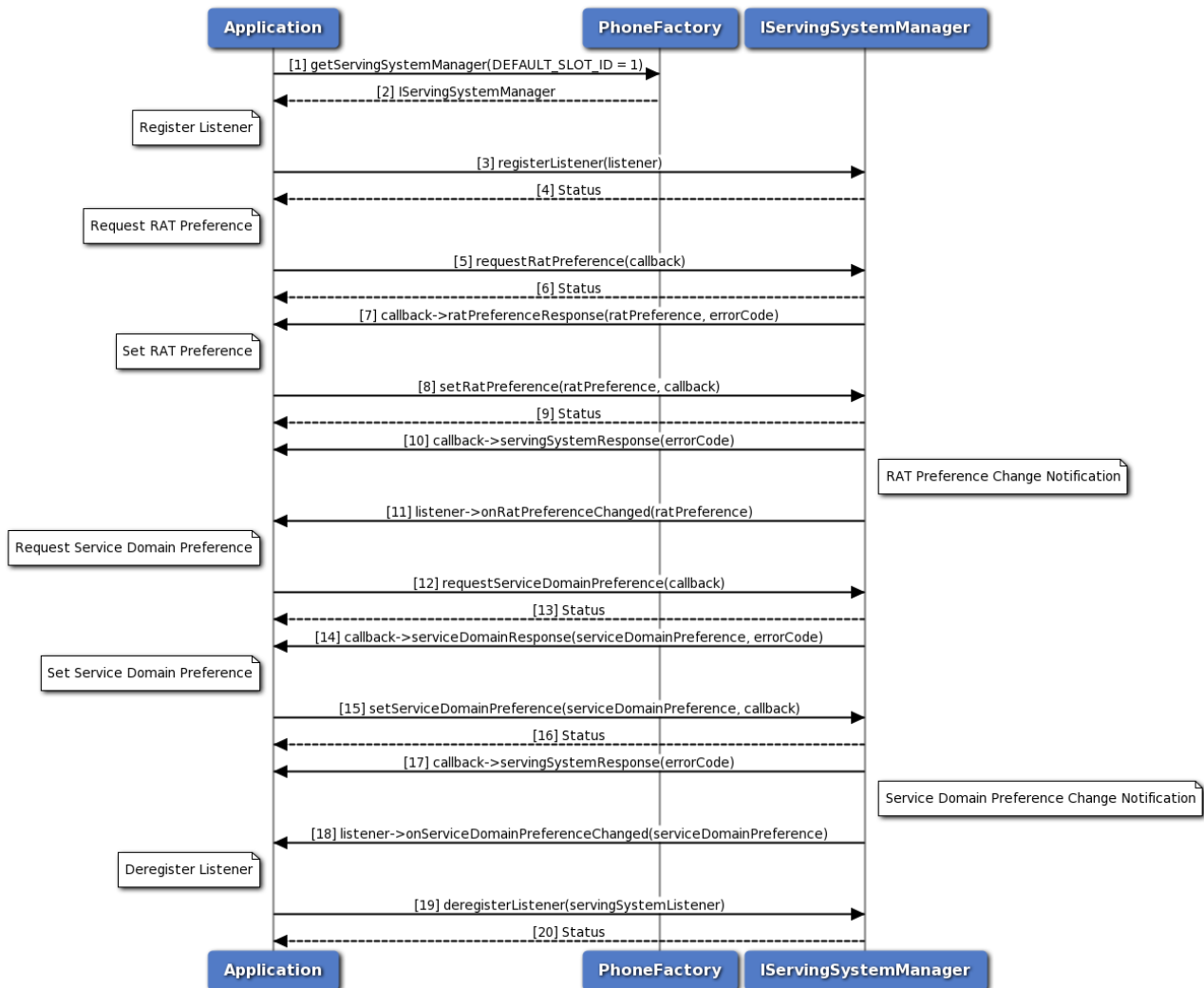


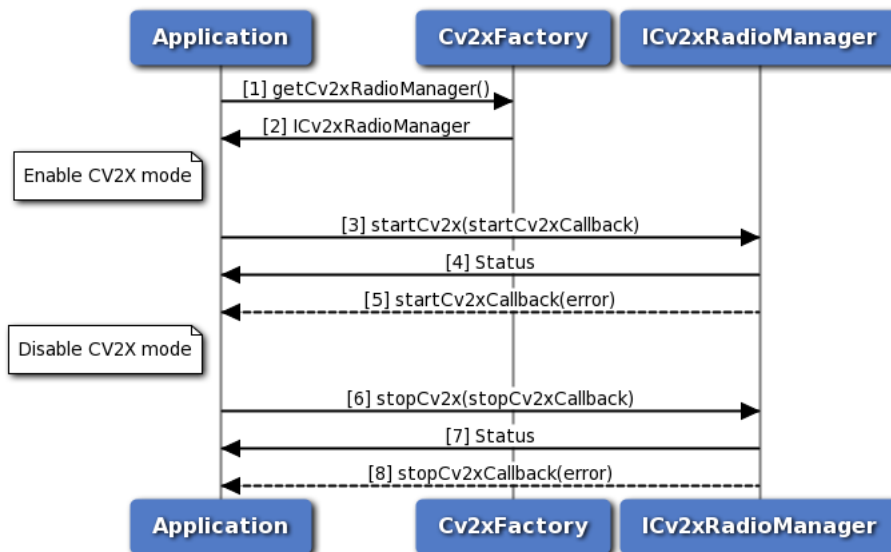
Figure 3-27 Serving System Manager Call Flow

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.

## 3.20 C-V2X

### 3.20.1 Start/Stop C-V2X Mode



**Figure 3-28 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.

### 3.20.2 C-V2X Radio Manager API

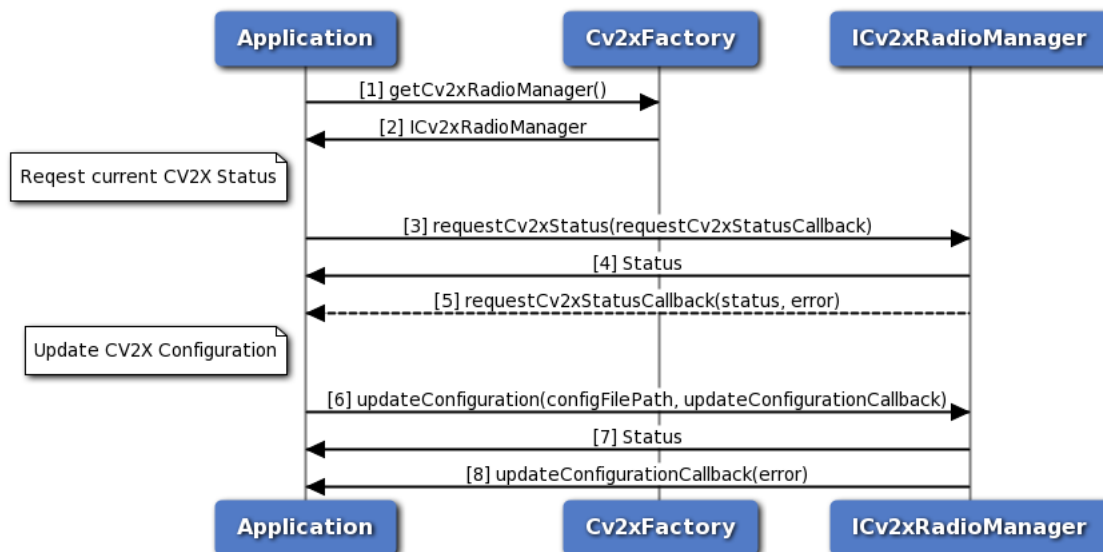


Figure 3-29 C-V2X Radio Manager call flow

#### 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.

### 3.20.3 C-V2X Radio Initialization

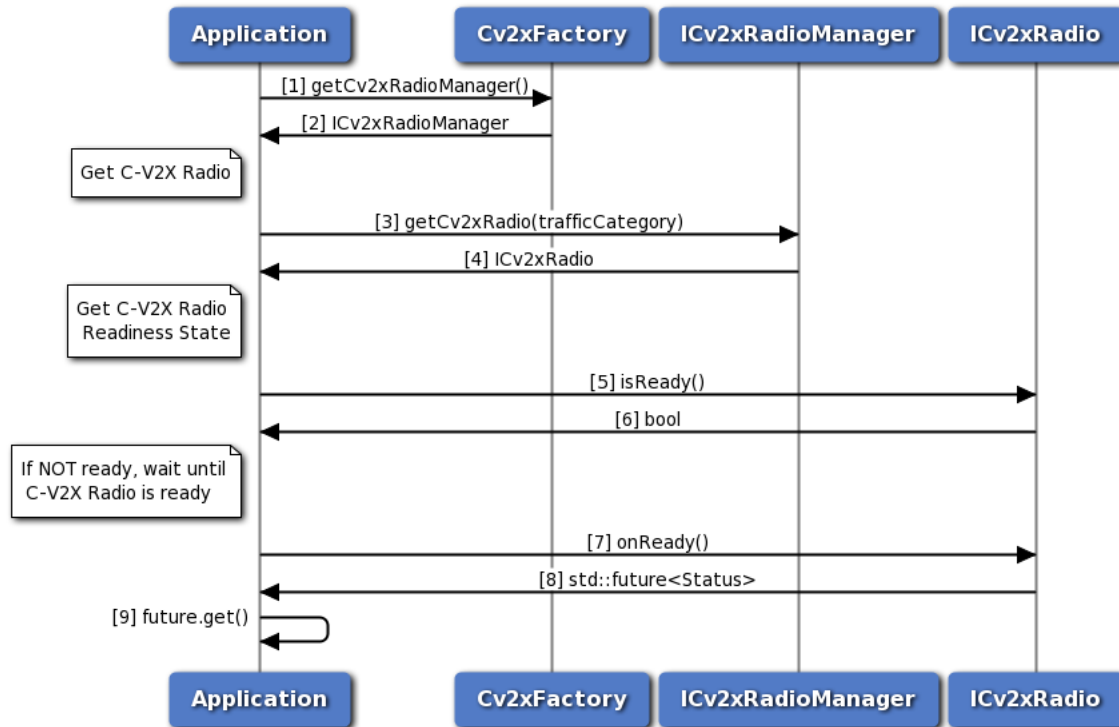
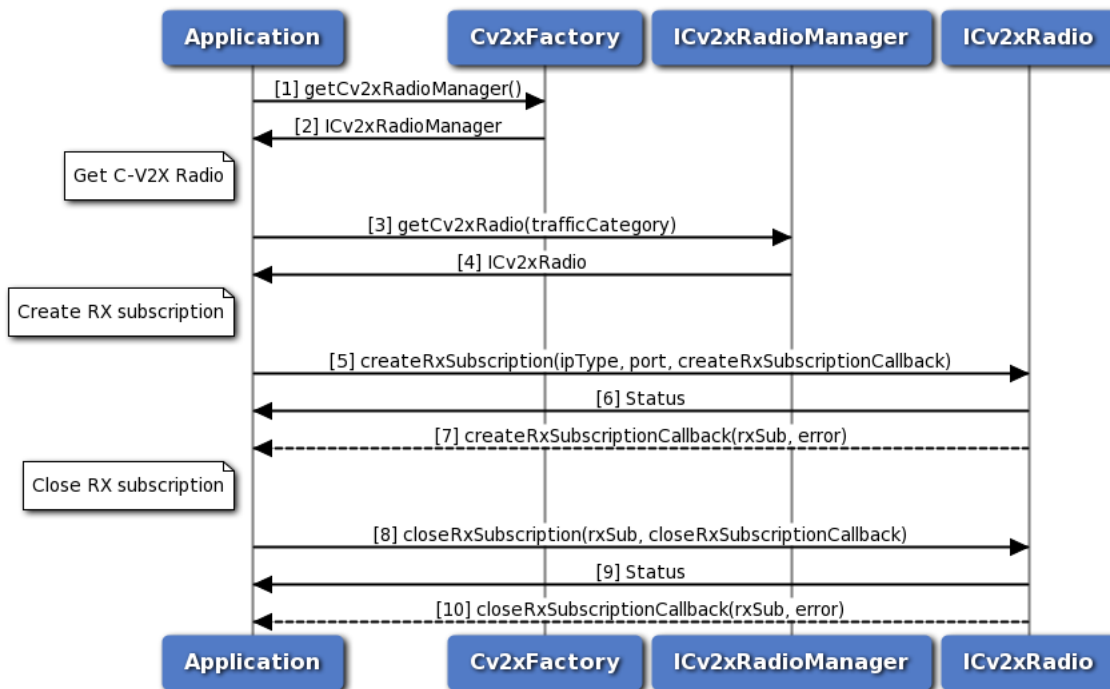


Figure 3-30 C-V2X Radio Initialization call flow

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).

### 3.20.4 C-V2X Radio RX subscription



**Figure 3-31 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.



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

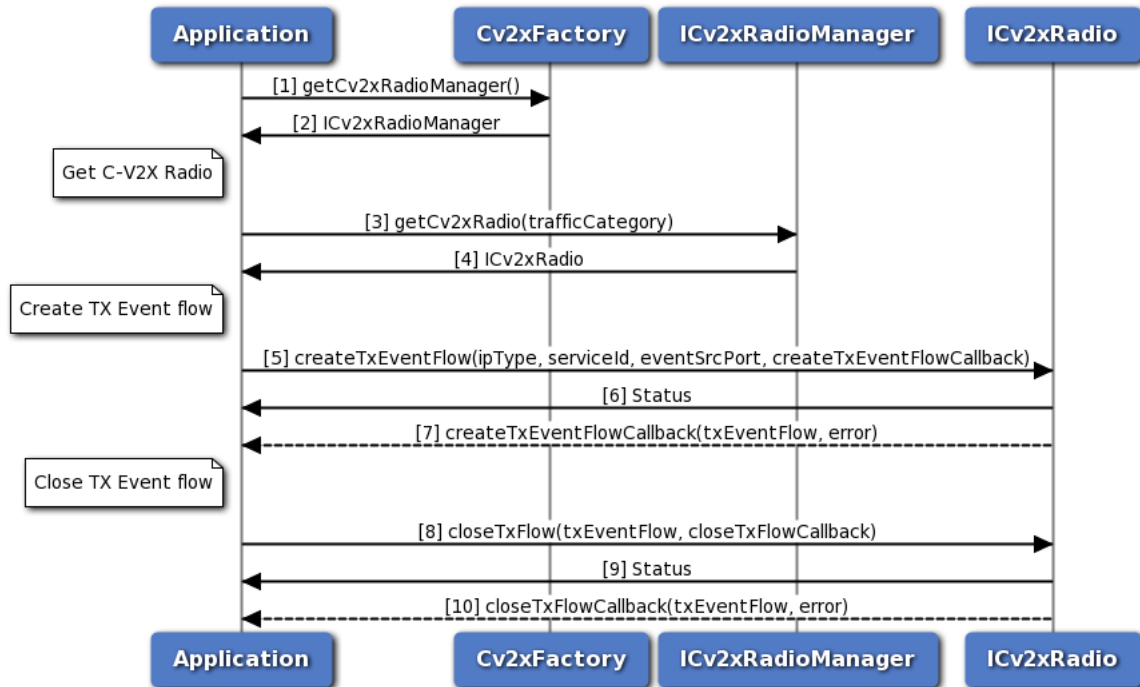
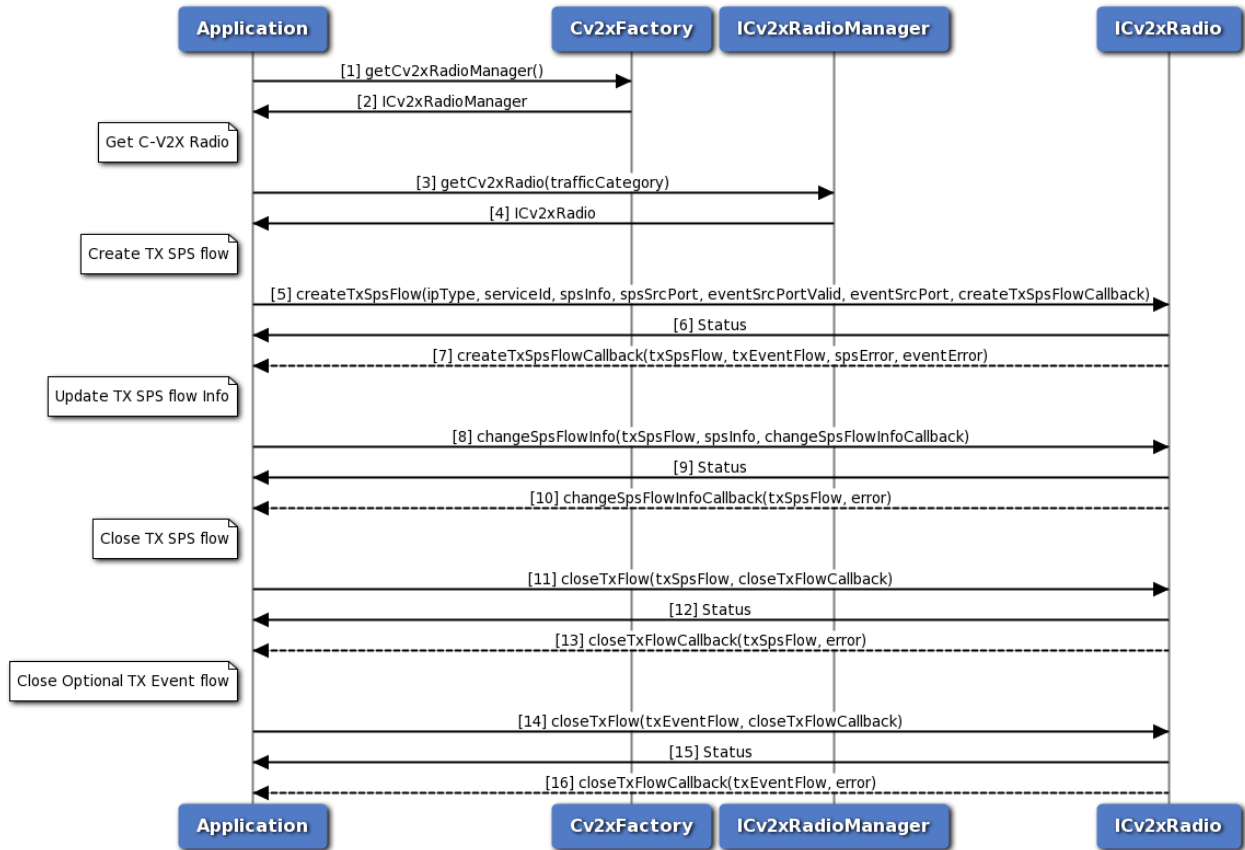


Figure 3-32 C-V2X Radio TX event-driven flow call flow

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.

### 3.20.6 C-V2X Radio TX SPS flow



**Figure 3-33 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.

## 3.21 Audio

### 3.21.1 Audio Manager API call flow

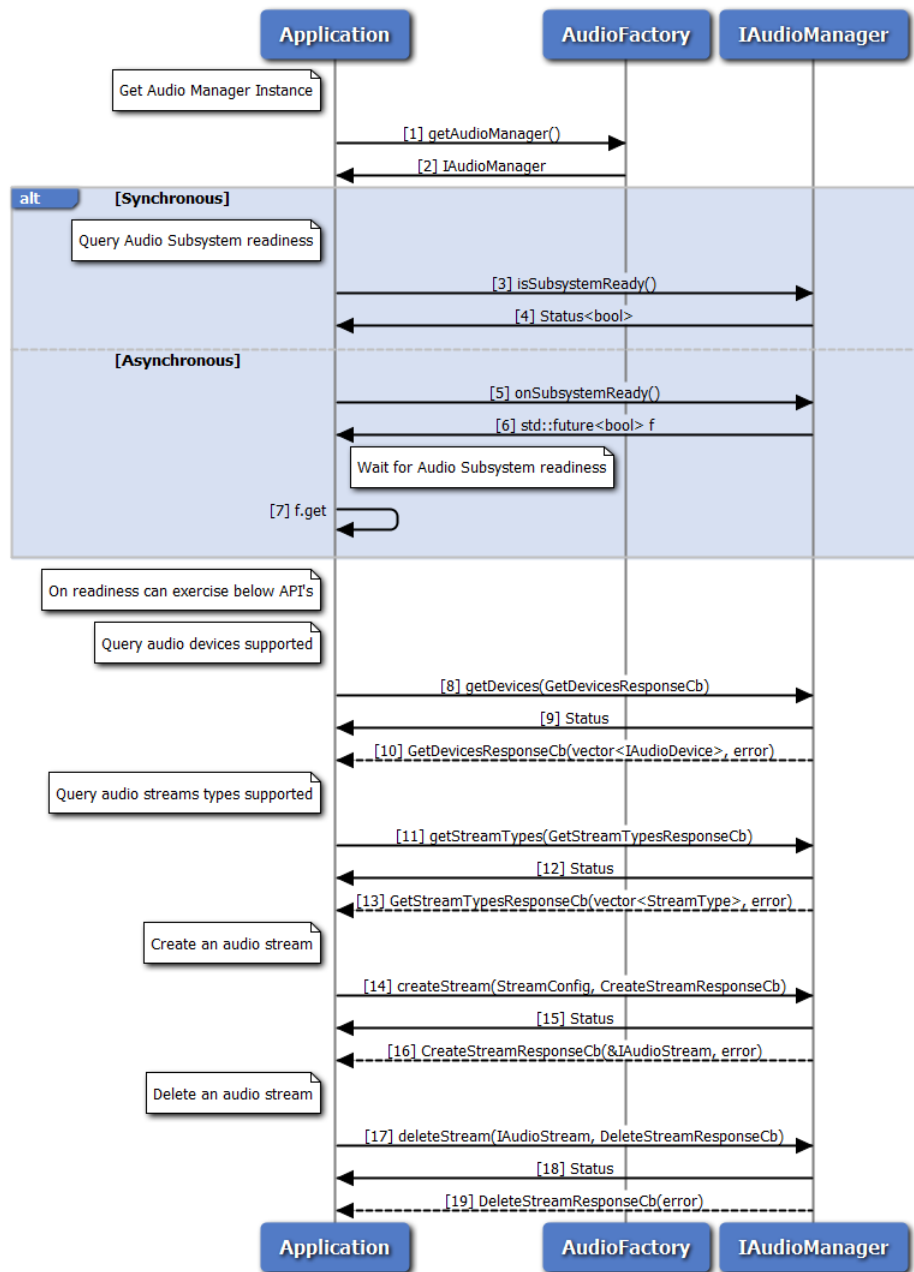


Figure 3-34 Audio Manager API call flow

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.

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

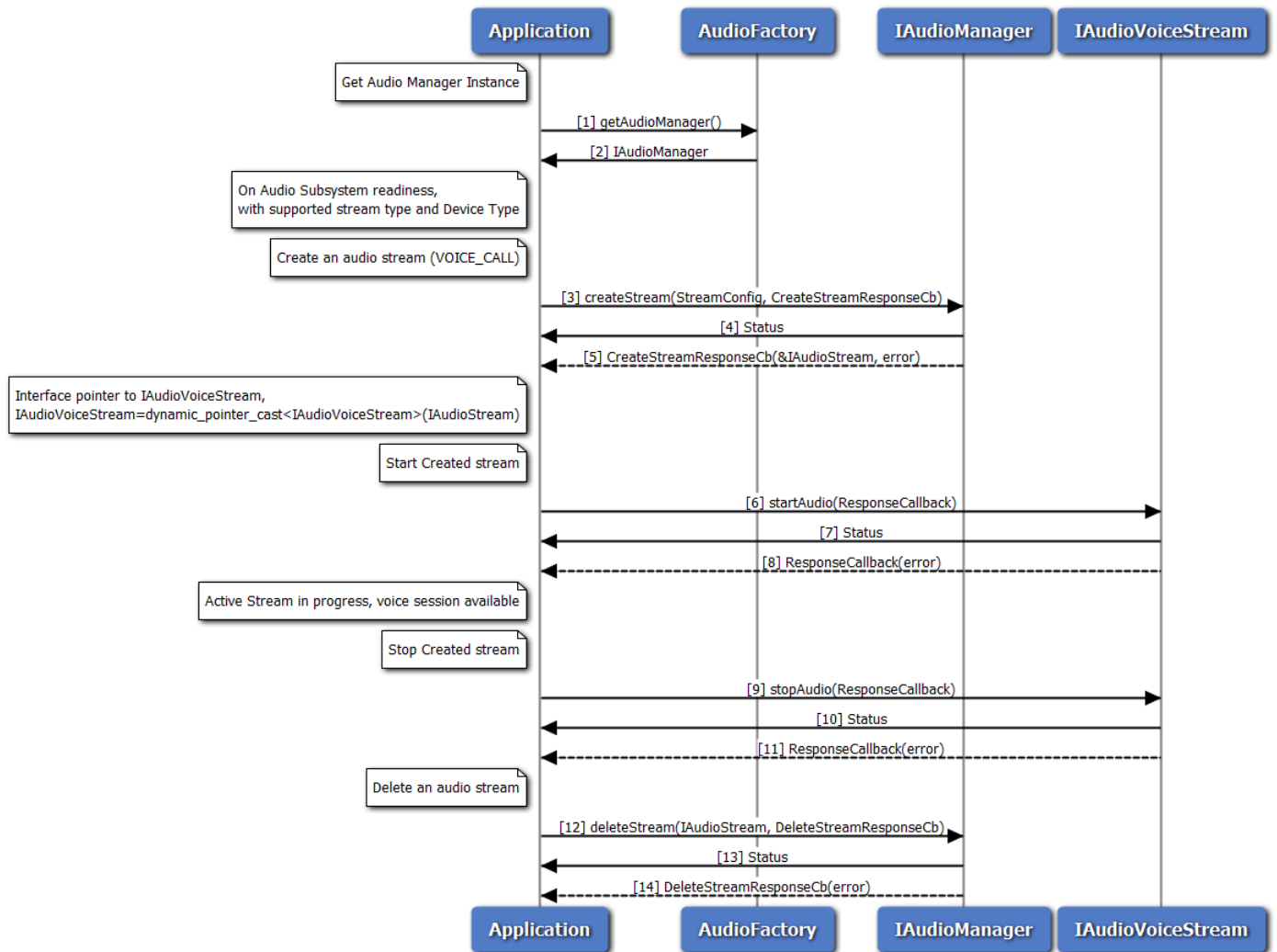


Figure 3-35 Audio Voice Call Start/Stop call flow

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.

### 3.21.3 Audio Voice Call Device Switch call flow

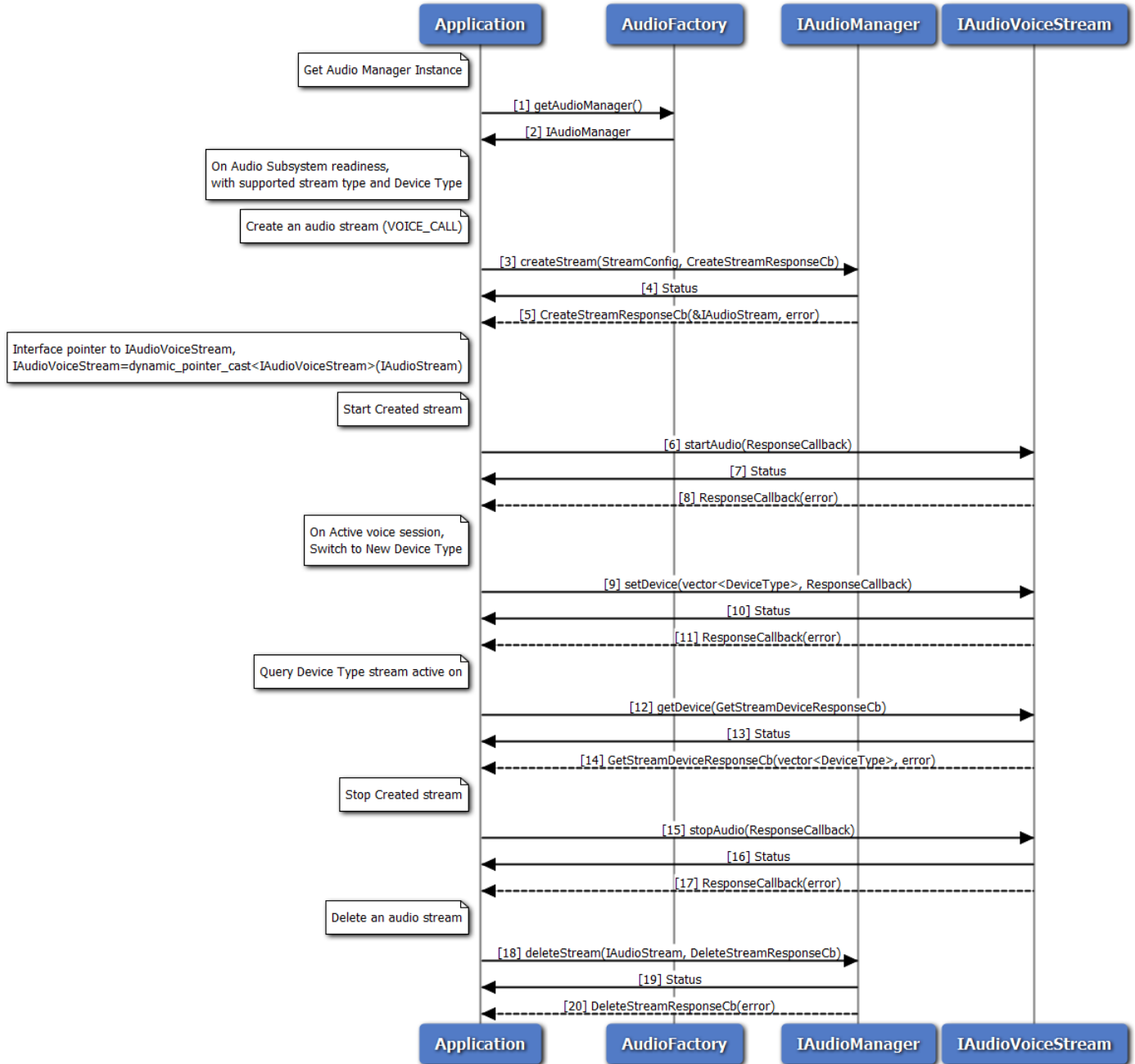


Figure 3-36 Audio Voice Call Device Switch call flow

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.

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

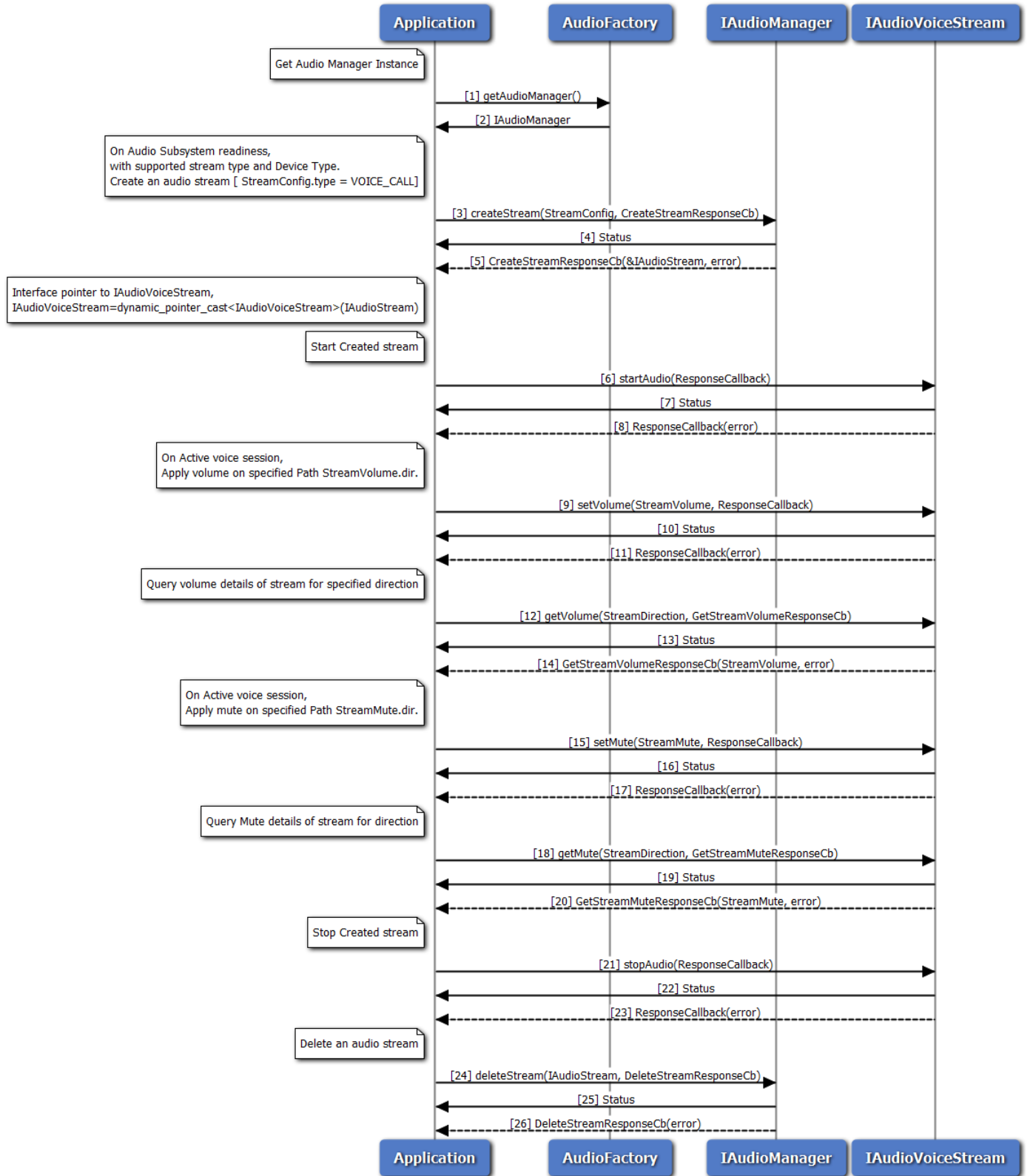


Figure 3-37 Audio Voice Call Volume/Mute control call flow

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.

### 3.21.5 Call flow to play DTMF tone

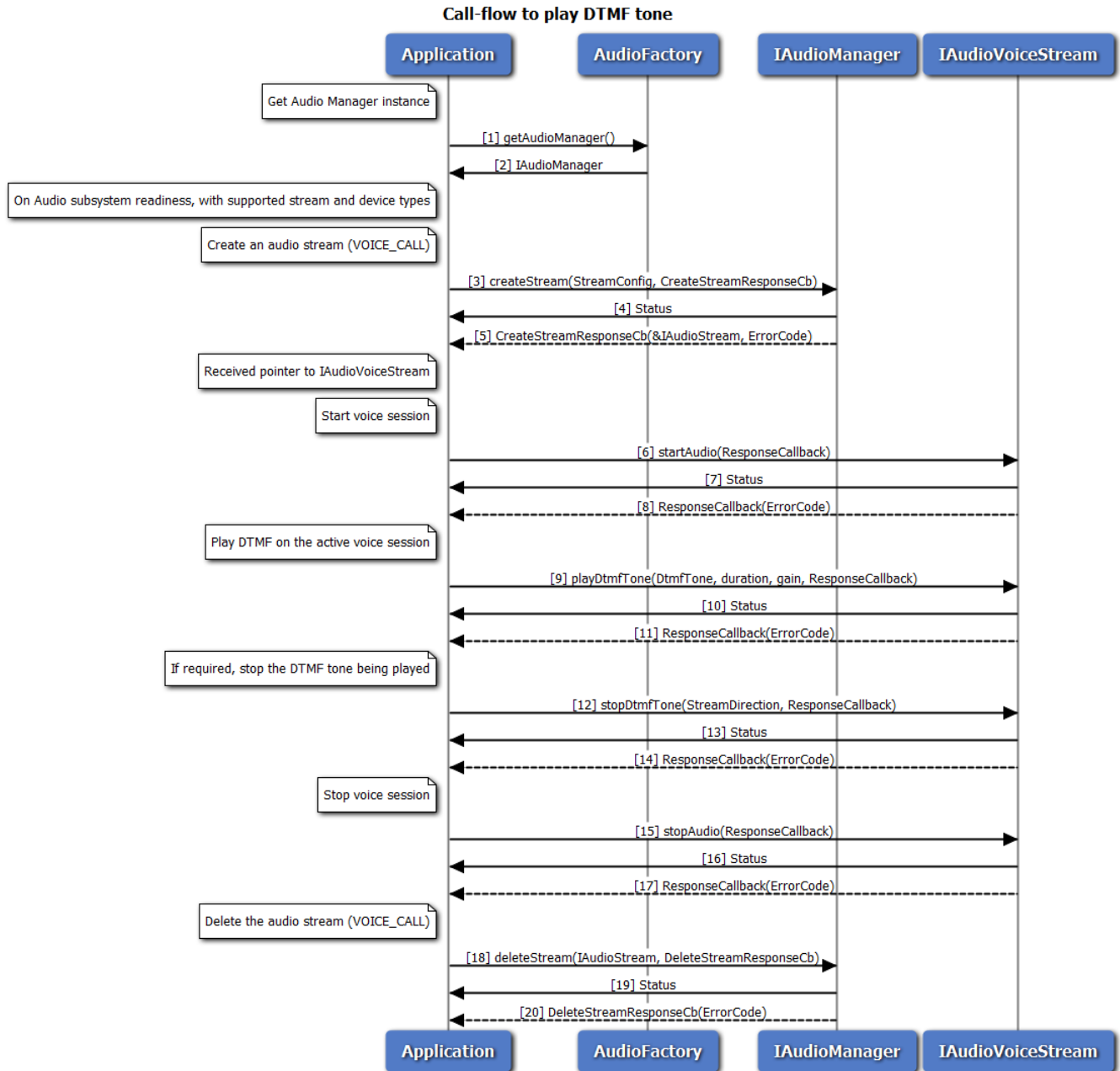
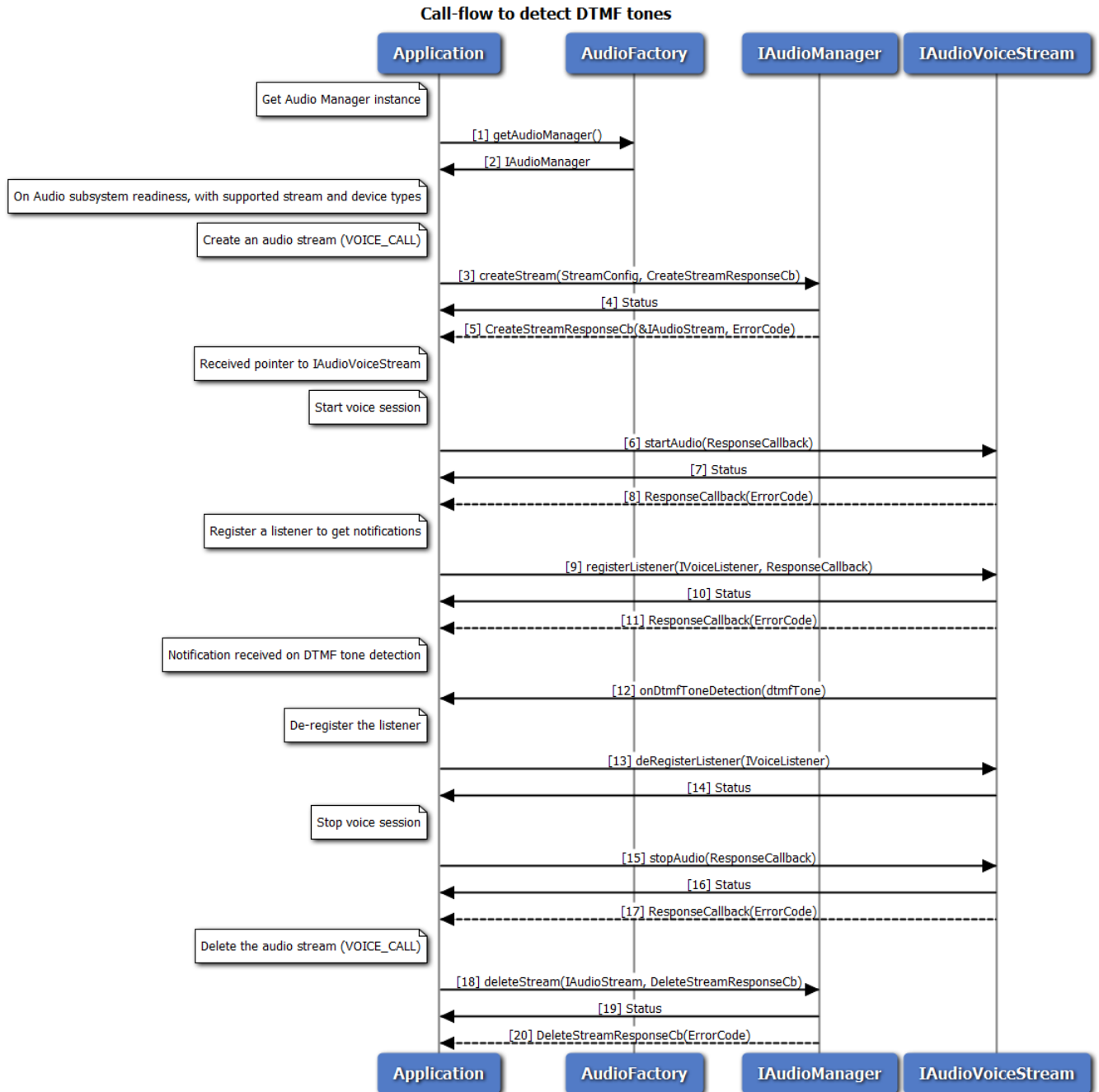


Figure 3-38 Call flow to play DTMF tone

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.

### 3.21.6 Call flow to detect DTMF tones



**Figure 3-39 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.

### 3.21.7 Audio Playback call flow

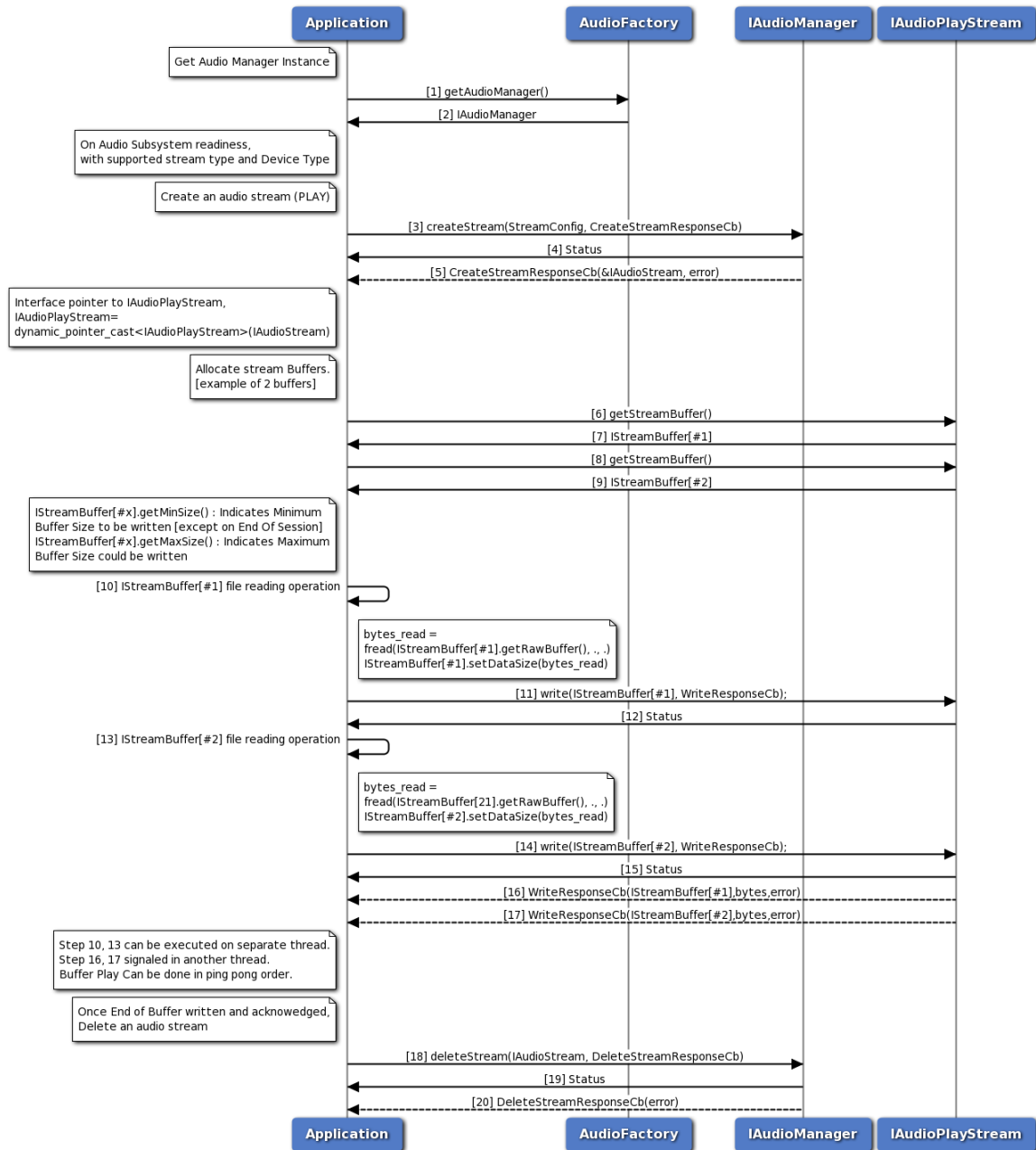


Figure 3-40 Audio Playback call flow

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.

### 3.21.8 Audio Capture call flow

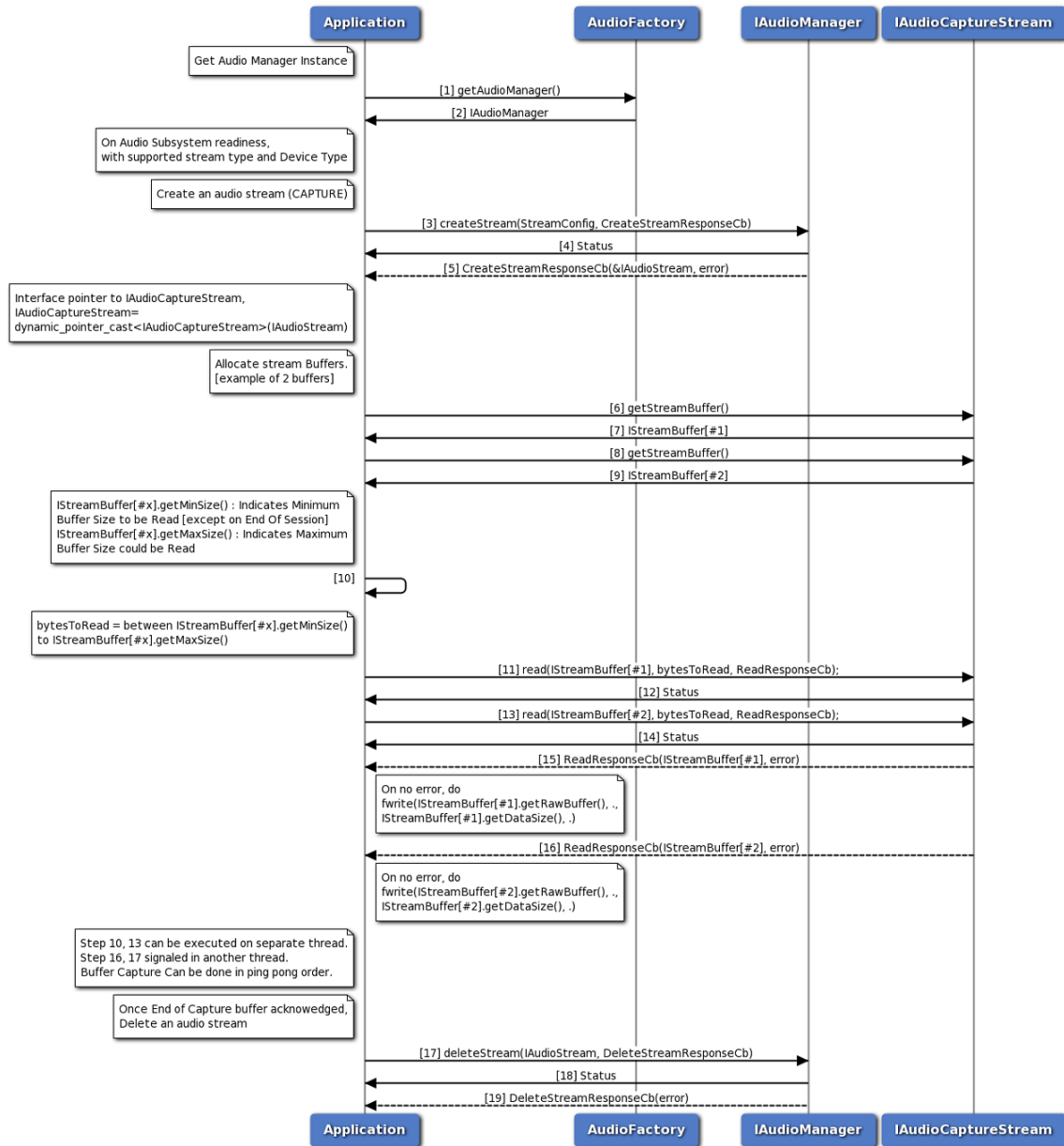
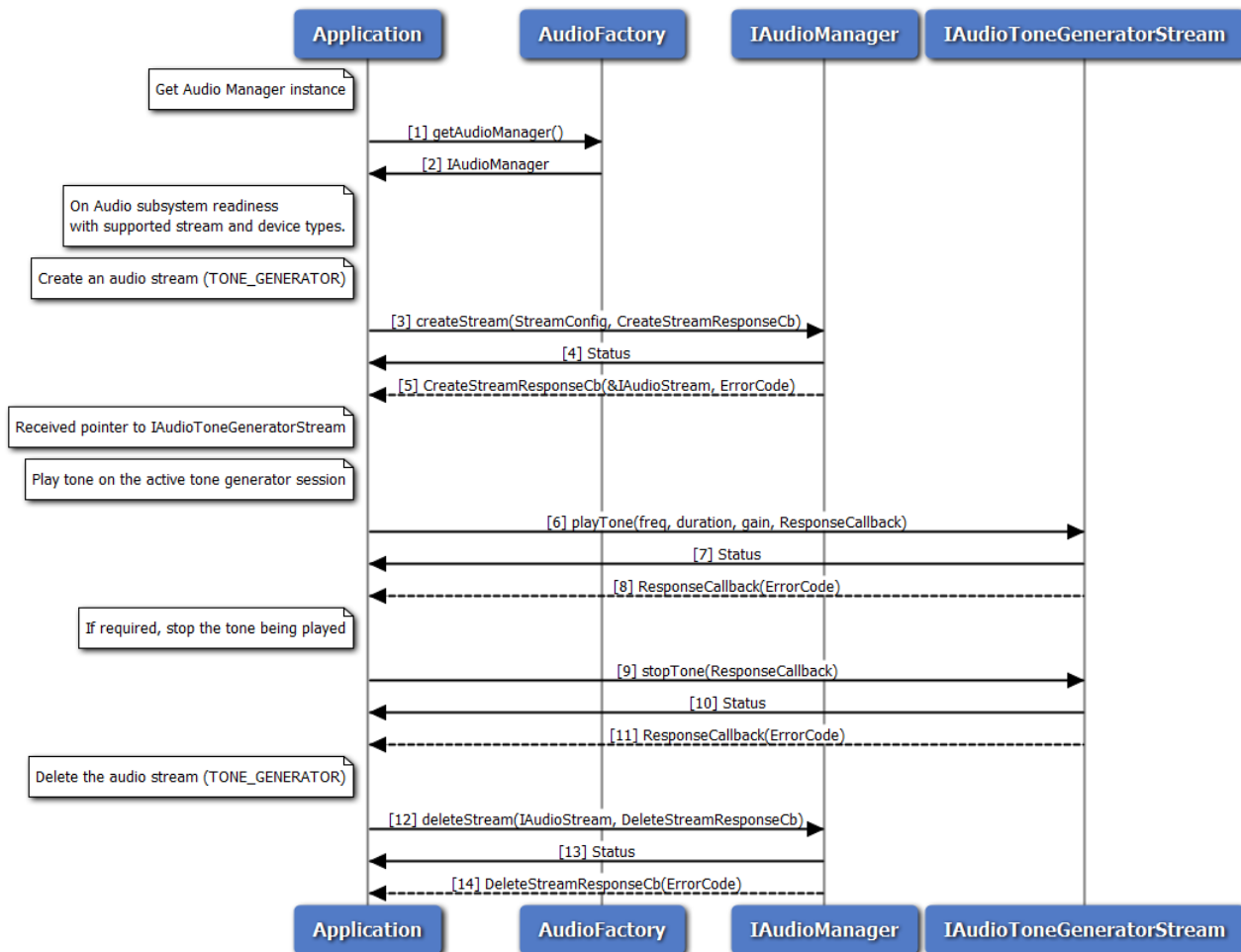


Figure 3-41 Audio Capture call flow

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.

### 3.21.9 Audio Tone Generator call flow



**Figure 3-42 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.

### 3.21.10 Audio Loopback call flow

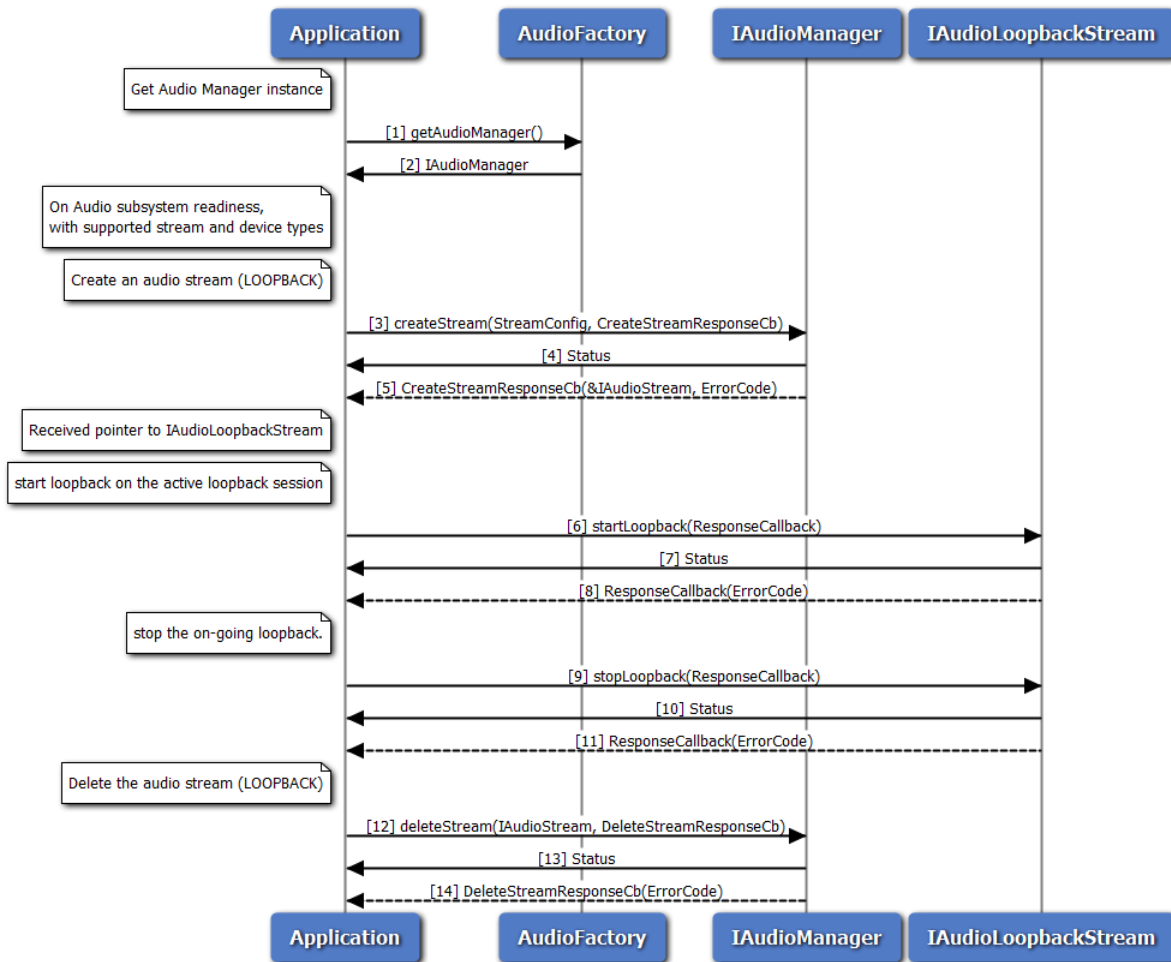


Figure 3-43 Call flow to start/stop loopback between source and sink devices

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.

### 3.21.11 Compressed audio format playback call flow

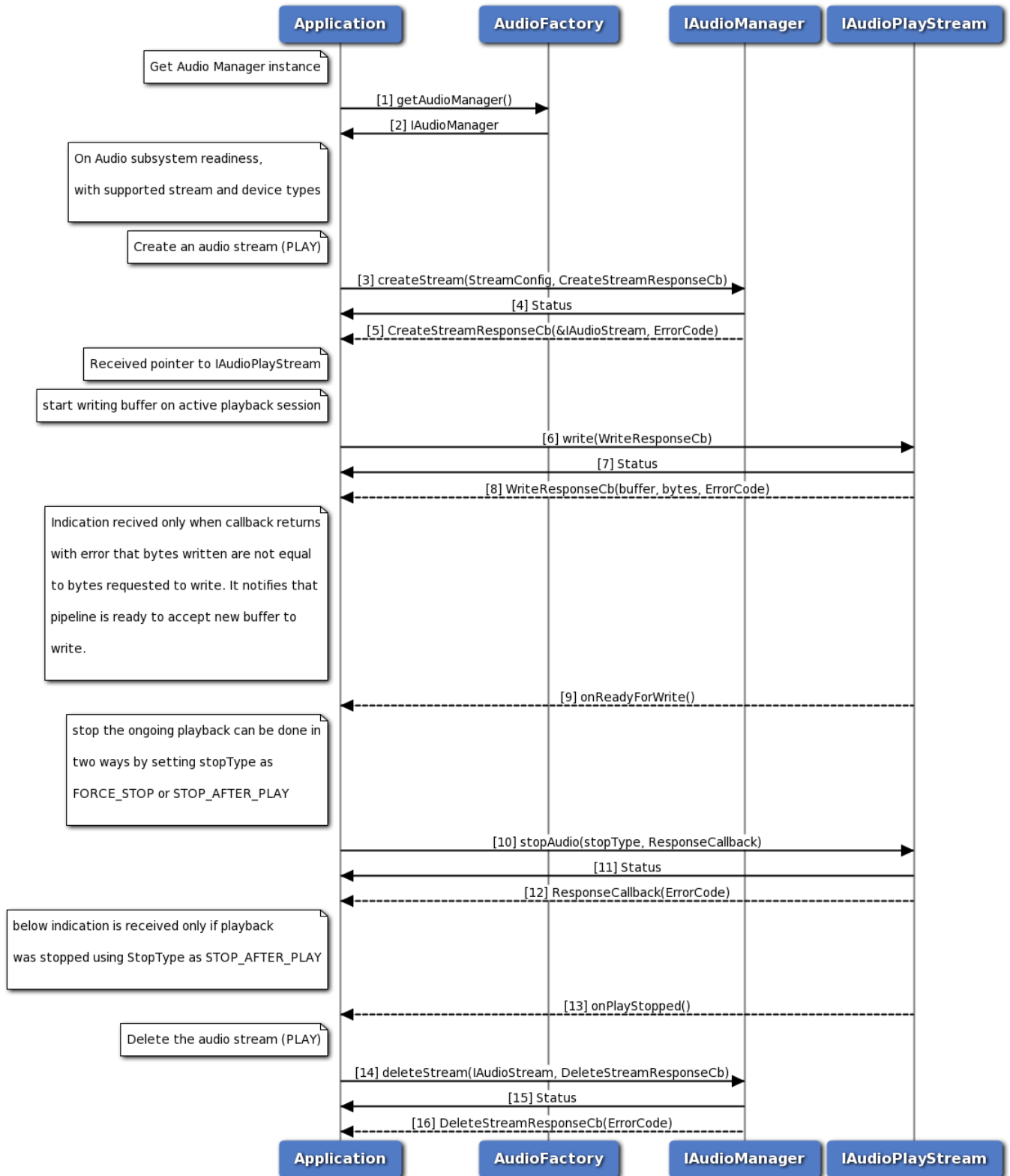


Figure 3-44 Call flow to play Compressed audio format

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.



### 3.21.12 Audio Transcoding Operation Callflow

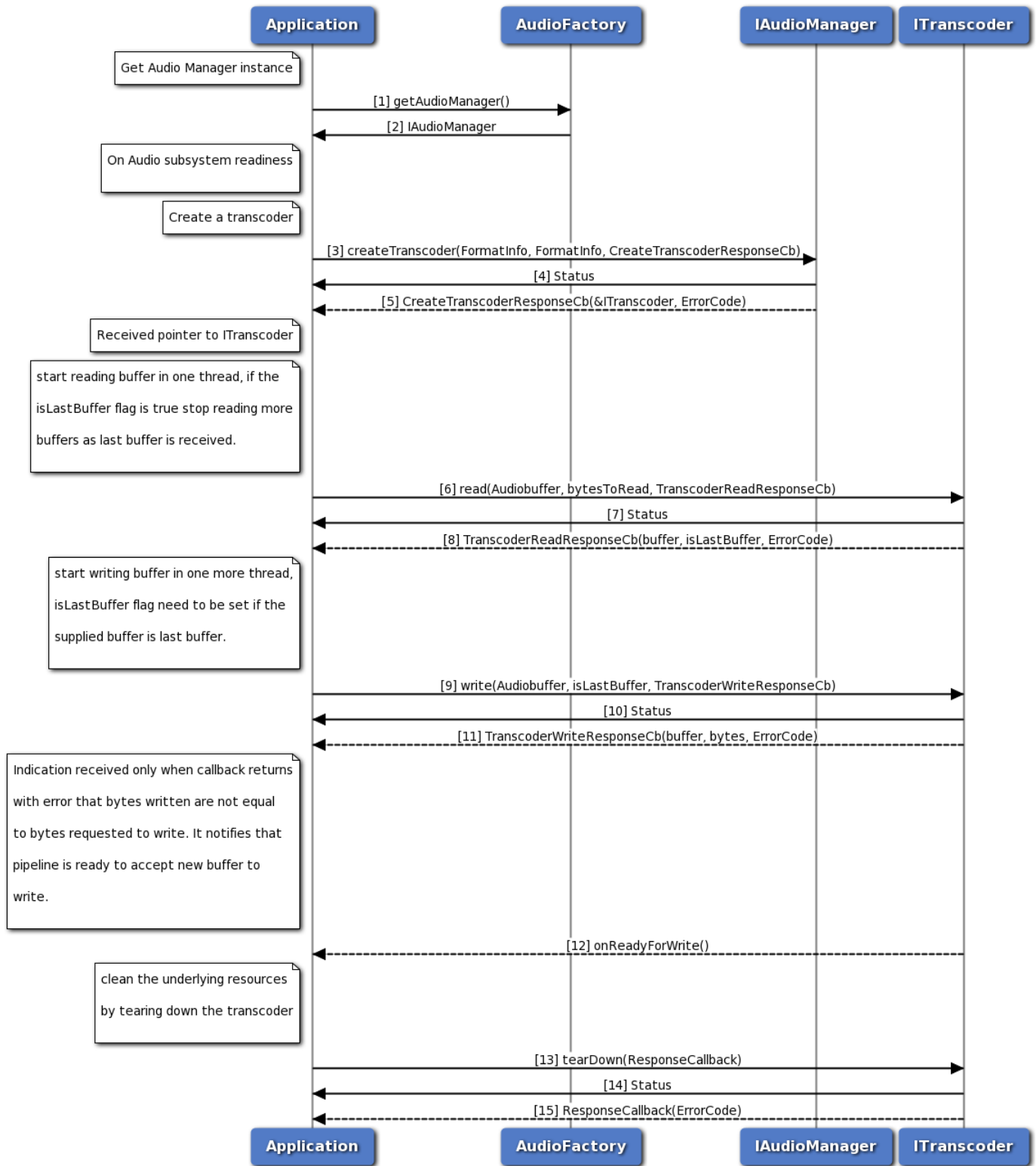


Figure 3-45 Audio Transcoding Operation Callflow

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.

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



Figure 3-46 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 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. On Readiness, Application requests to create a play stream with streamType as PLAY, voicePaths direction as TX or RX and no device is selected.
7. Application receives synchronous status which indicates if the createStream request was sent successfully.
8. 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.
9. Application requests to write buffer using write method on IAudioPlayStream.
10. Application receives synchronous status which indicates if the write request was sent successfully.
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. Application send request to stop playback using stopAudio method of IAudioPlayStream.
14. Application receives synchronous status which indicates if the stopAudio request was sent successfully.
15. Application is notified of the stopAudio request status (either SUCCESS or FAILED) via the application-supplied callback.
16. Application is notified via indication that playback is stopped if StopType is STOP\_AFTER\_PLAY.
17. Application requests delete audio play stream using deleteStream method.
18. Application receives synchronous Status which indicates if the deleteStream request was sent successfully.
19. Application is notified of the deleteStream request status(either SUCCESS or FAILED) via the application-supplied callback.
20. Application requests delete audio voice stream using deleteStream method.
21. Application receives synchronous Status which indicates if the deleteStream request was sent successfully.
22. Application is notified of the deleteStream request status(either SUCCESS or FAILED) via the application-supplied callback.

## 3.22 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.

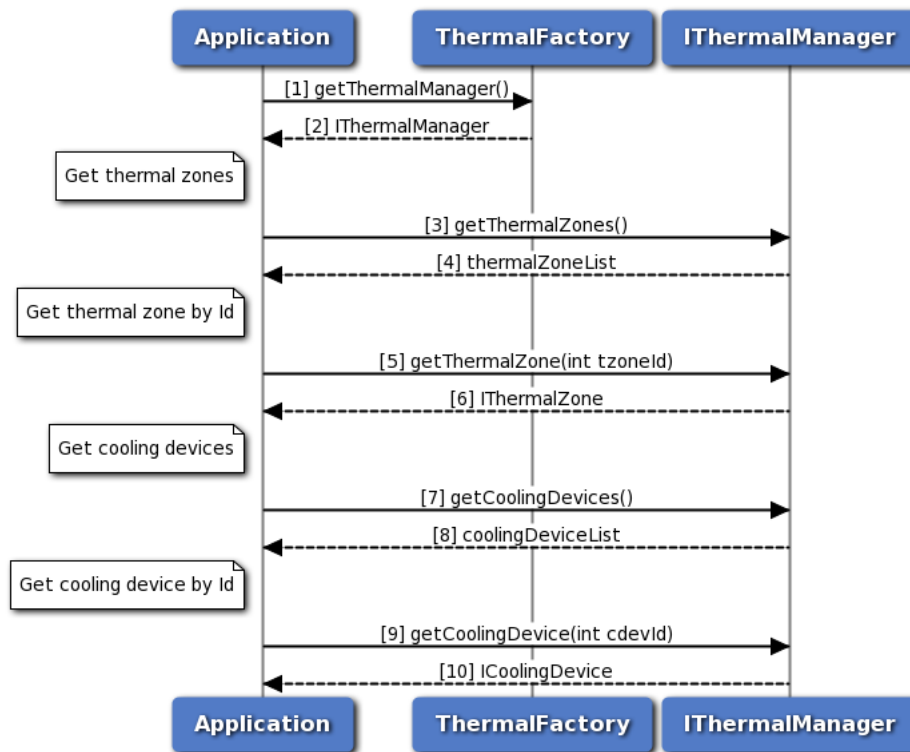


Figure 3-47 Thermal manager call flow

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.

## 3.23 Thermal shutdown management

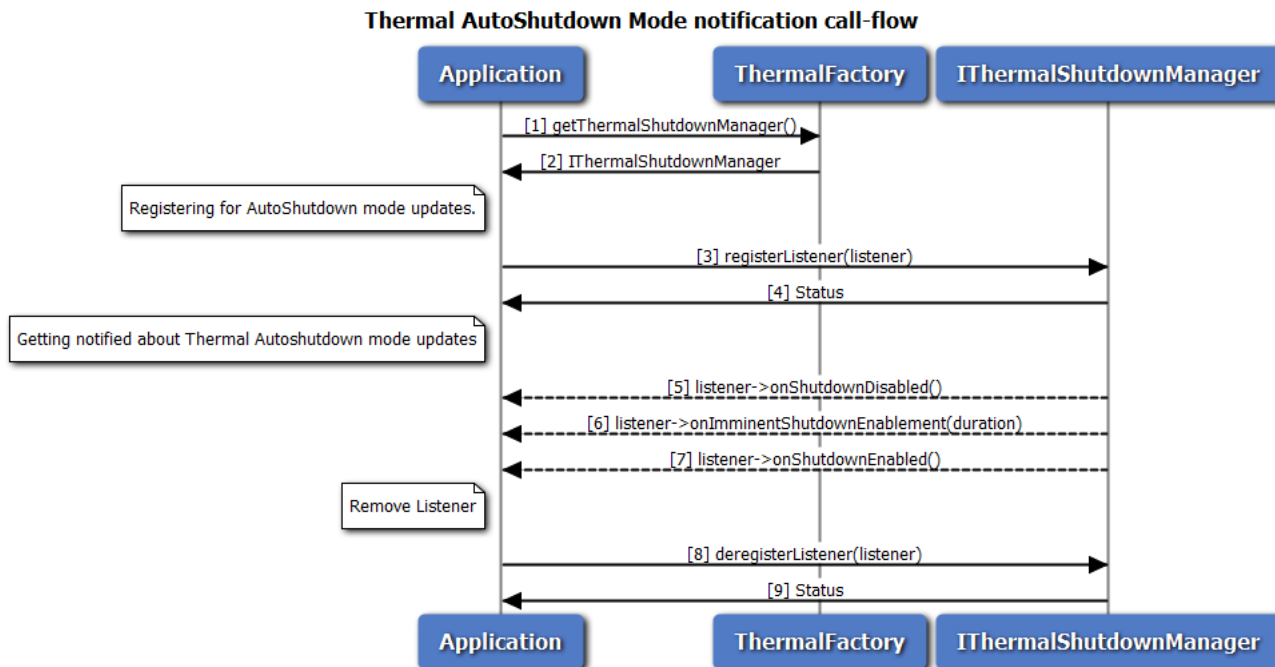
Thermal shutdown manager provides APIs to set/get auto thermal shutdown modes. It also has listener interface for notifications. Application will get the Thermal-shutdown manager object from thermal factory.

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.

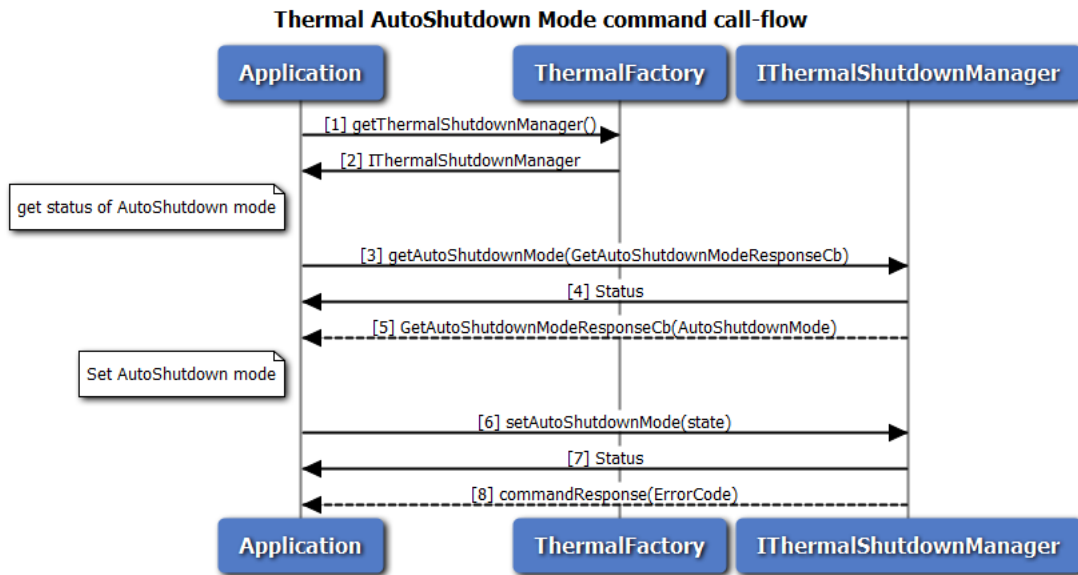
### 3.23.1 Call flow to register/remove listener for Thermal auto-shutdown mode updates.



**Figure 3-48 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.

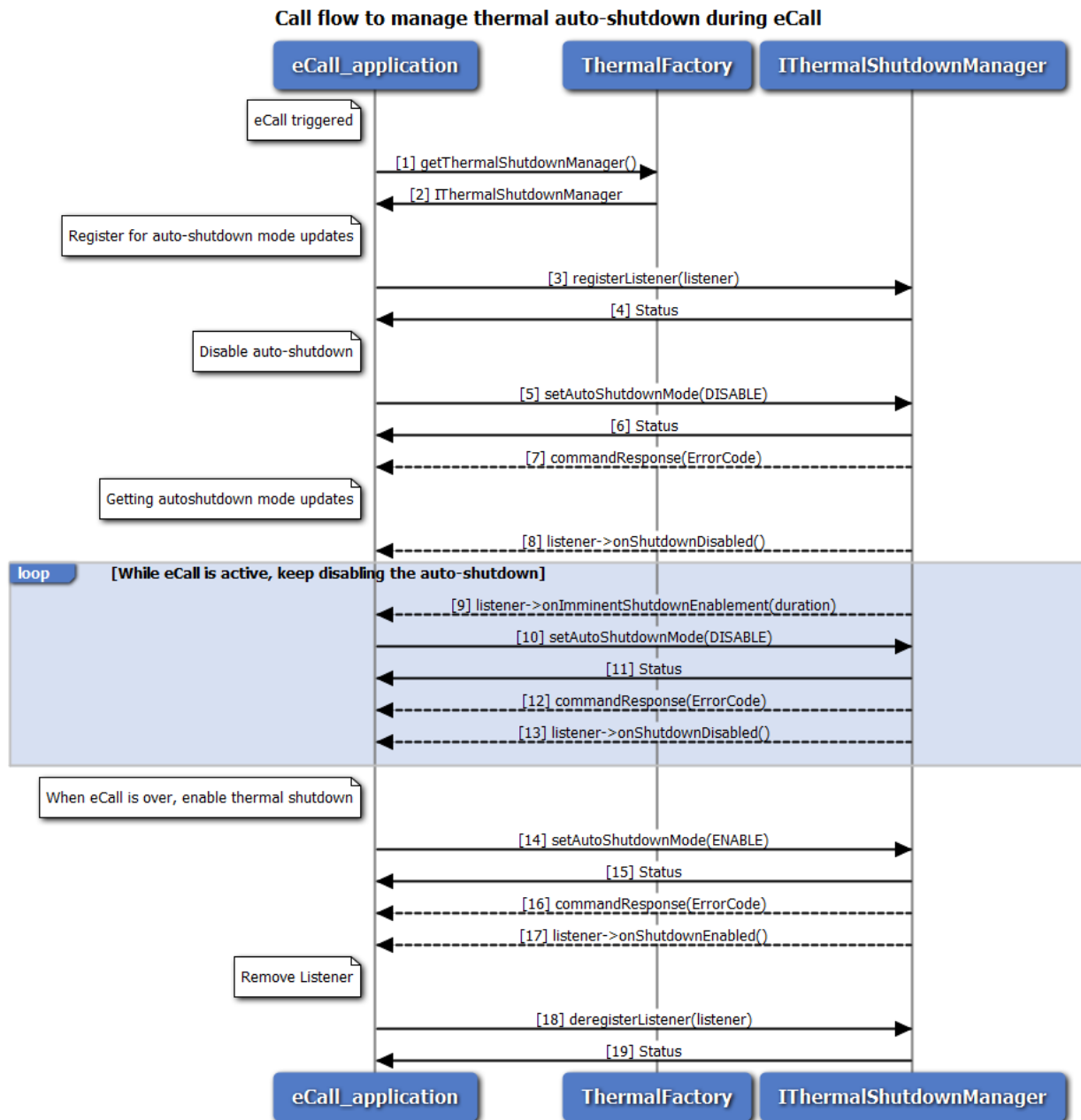
### 3.23.2 Call flow to set/get the Thermal auto-shutdown mode



**Figure 3-49 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.

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



**Figure 3-50 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



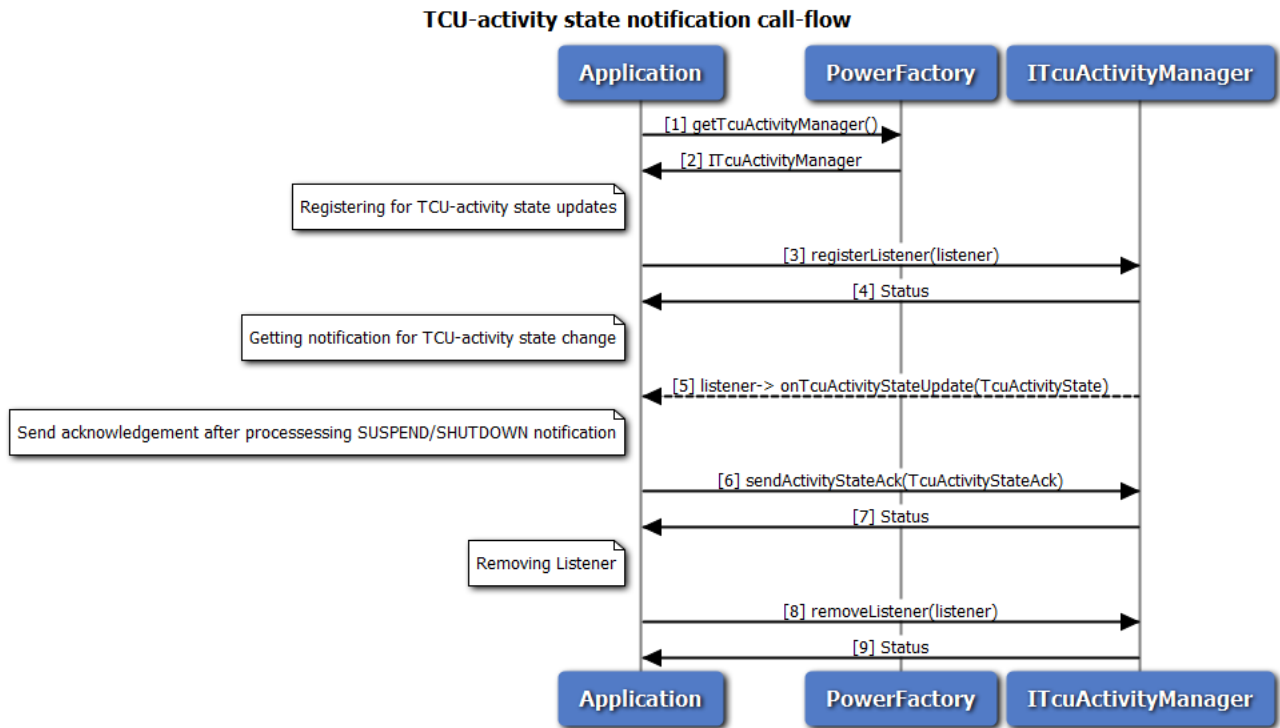
auto-shutdown during eCall.

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.

## 3.24 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.

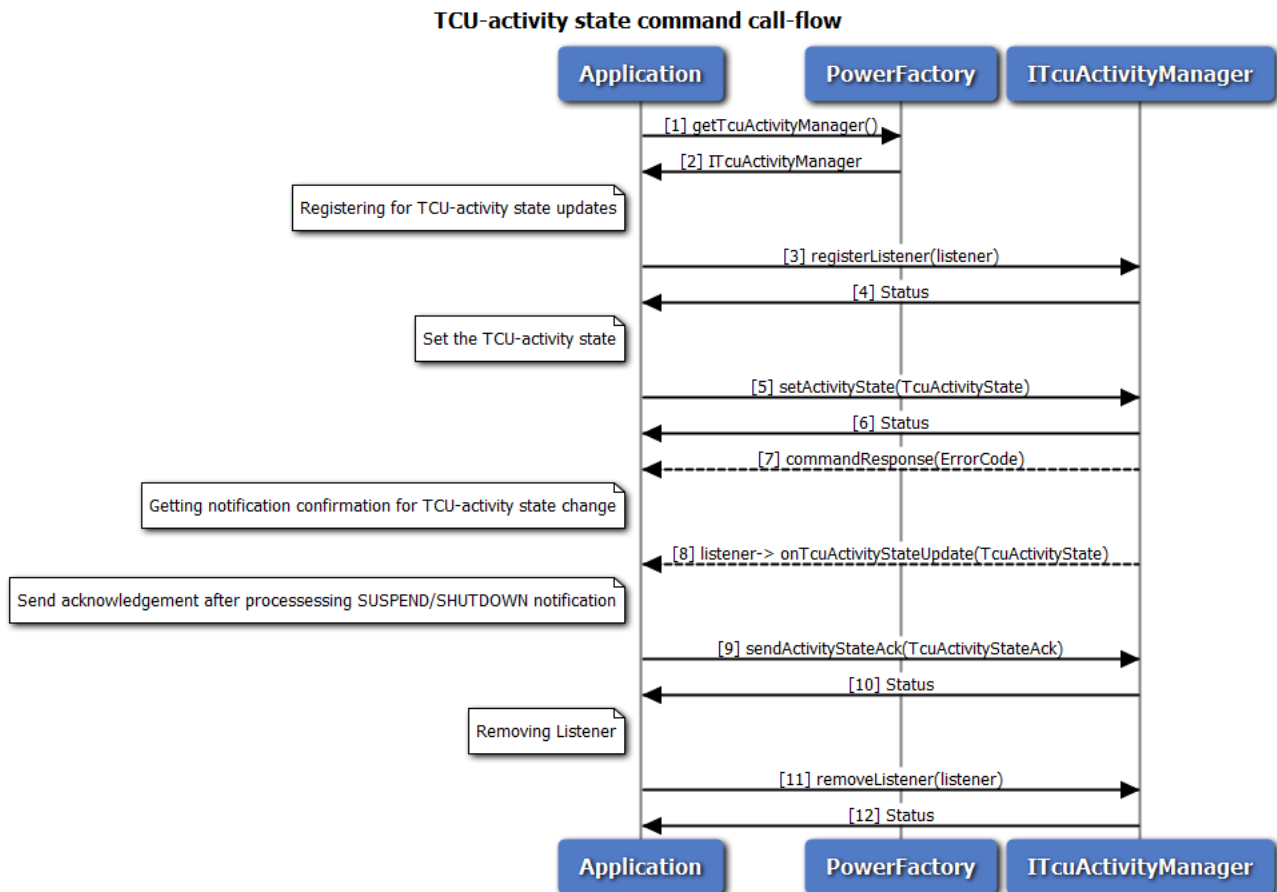
### 3.24.1 Call flow to register/remove listener for TCU-activity manager



**Figure 3-51 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.

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



**Figure 3-52 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.

### 3.25 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.

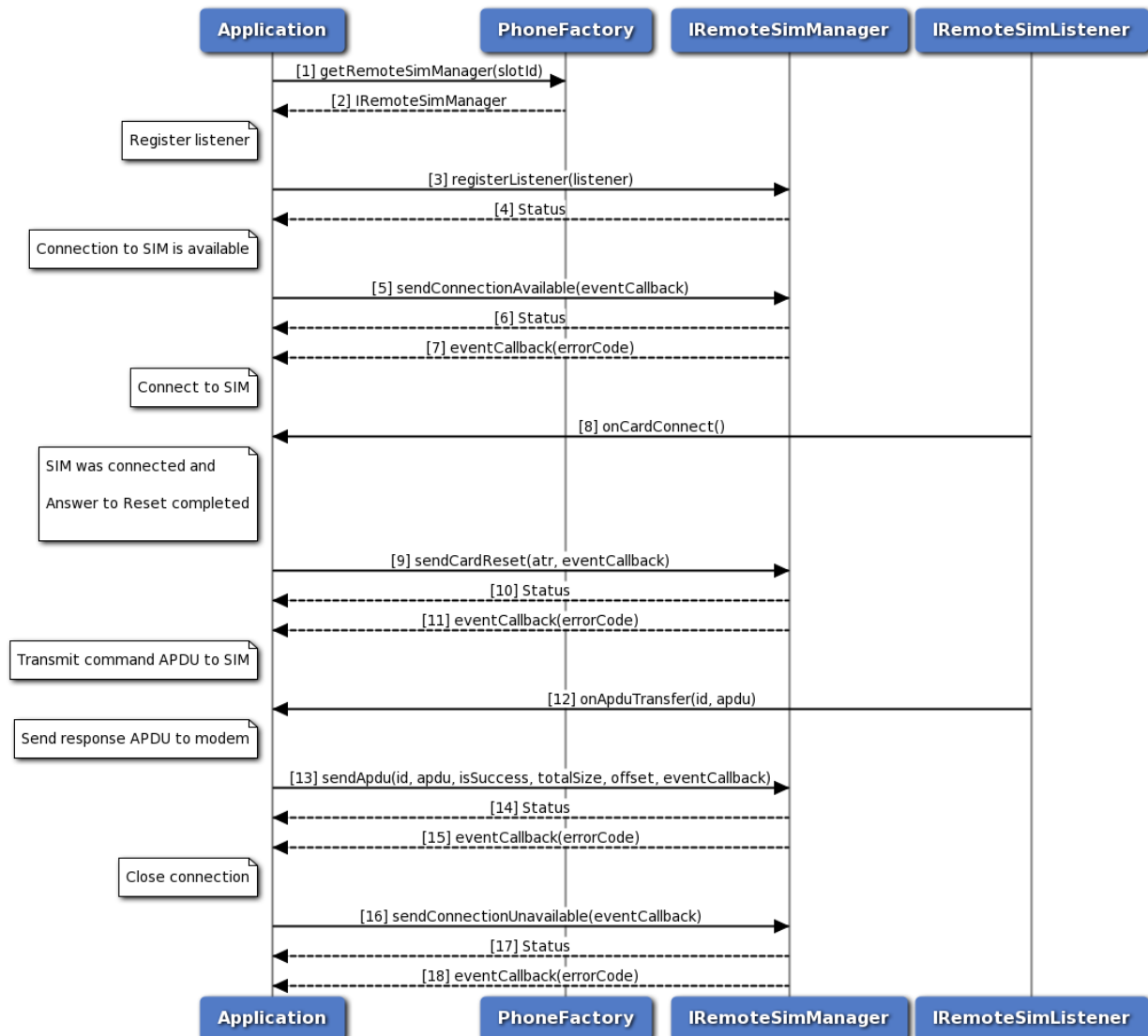


Figure 3-53 Remote SIM call flow

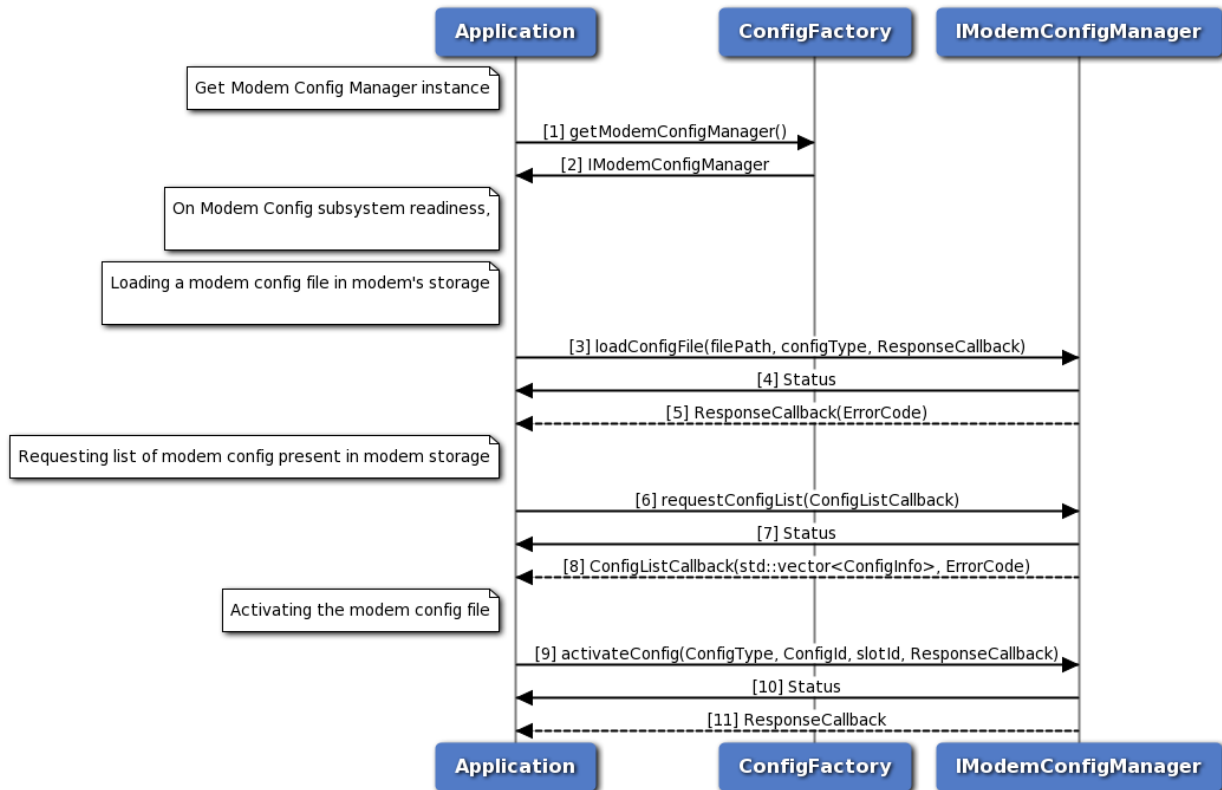
1. Application requests remote SIM manager object from phone factory, specifying a slot id.

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.

## 3.26 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.

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

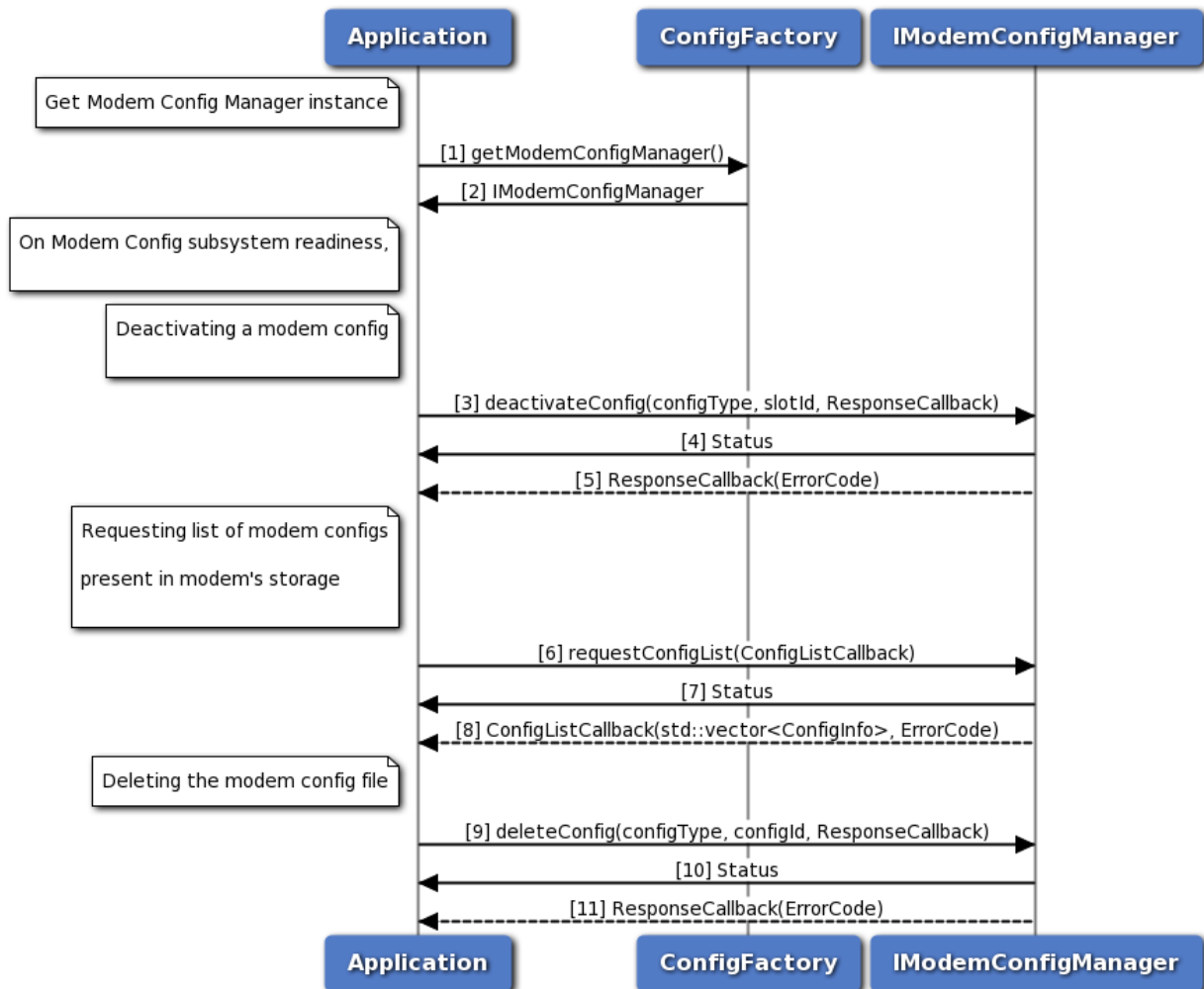


**Figure 3-54 Modem Config load and activate call flow**

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.

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

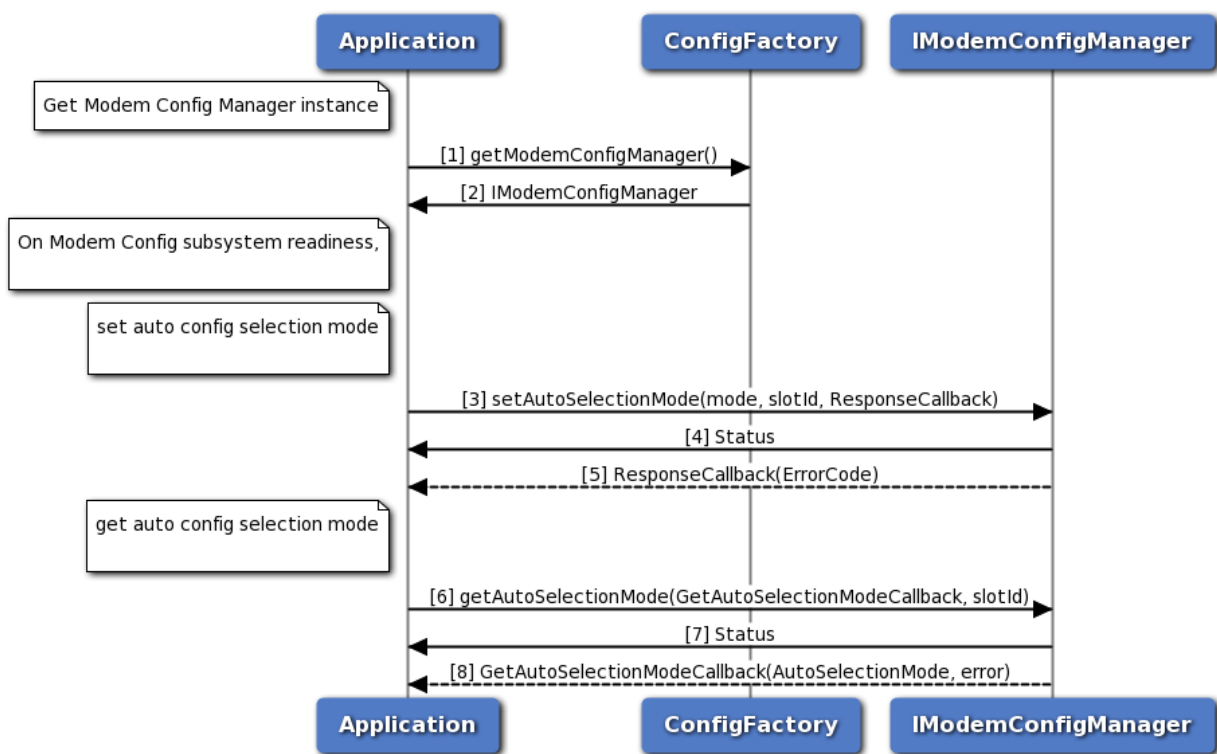


**Figure 3-55 Modem Config deactivate and delete Call Flow**

- 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.

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



**Figure 3-56 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.

# 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::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::loc::ILocationManager::getHorizontalAccuracyLevel ()=0`**

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.

**Global `telux::loc::ILocationManager::getMinIntervalForFinalReports ()=0`**

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.

**Global `telux::loc::ILocationManager::getPositionReportTimeout ()=0`**

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.

**Global `telux::loc::ILocationManager::registerListener (std::weak_ptr< ILocationListener > listener)=0`**

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.

**Global `telux::loc::ILocationManager::removeListener` (`std::weak_ptr< ILocationListener > listener`)=0**

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.

**Global `telux::loc::ILocationManager::setHorizontalAccuracyLevel` (`HorizontalAccuracyLevel accuracy=HorizontalAccuracyLevel::LOW`, `std::shared_ptr< telux::common::ICommandResponseCallback > callback=nullptr`)=0**

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.

**Global `telux::loc::ILocationManager::setMinIntervalForReports` (`uint32_t minInterval`, `std::shared_ptr< telux::common::ICommandResponseCallback > callback=nullptr`)=0**

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.

**Global `telux::loc::ILocationManager::setPositionReportTimeout` (`uint32_t timeout`, `std::shared_ptr< telux::common::ICommandResponseCallback > callback=nullptr`)=0**

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.

**Global `telux::tel::ICallListener::onECallMsdTransmissionStatus` (`int phoneId`, `telux::common::IErrorCodes::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::ICommandResponseCallback > 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::ServiceState`**

Use requestVoiceServiceState() API or to know the status of phone

# 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

## 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)`
- `std::shared_ptr< ICallManager > getCallManager ()`
- `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)`

##### 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 )`

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
----	----------------	---------------------------------

##### 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 ( )`

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

##### Returns

Pointer of [ICallManager](#) object.

#### 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.

##### Parameters

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.



## 5.3 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 telx::tel::WcdmaCellIdentity::getMcc ( )`

Get the Mobile Country Code.

##### Returns

Mcc value.

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

Get the Mobile Network Code.

##### Returns

Mnc value.

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

Get the location area code.

##### Returns

Location area code.

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

Get the cell identity.

##### Returns

Cell identity.

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

Get the primary scrambling code.

##### Returns

Primary scrambling code.

#### 5.3.1.4.2.6 `const int telx::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
ECallMsd↔ Optionals	optionals	Indicates presence of optionals in ECall MSD
uint8_t	message↔ Identifier	Starts with 1 for each new , increment on retransmission
ECallMsd↔ ControlBits	control	ECallMsdControlBits 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="#">ECallMode</a> ↔ <a href="#">Reason</a>	reason	Represents eCall operating mode change reason

**5.3.1.21 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](#) ()

**5.3.1.21.1 Constructors and Destructors**

**5.3.1.21.1.1** virtual [telux::tel::IPhone::~IPhone](#) ( ) [[virtual](#)]

**5.3.1.21.2 Member Function Documentation**

### 5.3.1.21.2.1 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.

### 5.3.1.21.2.2 virtual RadioState telux::tel::IPhone::getRadioState ( ) [pure virtual]

Get Radio state of device.

#### Returns

[RadioState](#)

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

### 5.3.1.21.2.3 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](#).

### 5.3.1.21.2.4 virtual ServiceState telux::tel::IPhone::getServiceState ( ) [pure virtual]

Get service state of the phone.

#### Returns

[ServiceState](#)

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

**5.3.1.21.2.5** `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](#).

**5.3.1.21.2.6** `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

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

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

#### Parameters

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

### 5.3.1.21.2.8 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

### 5.3.1.21.2.9 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.

### 5.3.1.21.2.10 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

#### Note

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

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

Get the eCall operating mode

#### Parameters

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

#### Returns

Status of requestECallOperatingMode i.e. success or suitable error

#### Note

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

### 5.3.1.22 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](#) ()

#### 5.3.1.22.1 Constructors and Destructors

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

#### 5.3.1.22.2 Member Function Documentation

##### 5.3.1.22.2.1 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

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

### 5.3.1.23 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](#) ()

#### 5.3.1.23.1 Constructors and Destructors

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

#### 5.3.1.23.2 Member Function Documentation

5.3.1.23.2.1 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>

### 5.3.1.24 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	

### 5.3.1.25 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	An array of struct which contains mask of RAT capabilities and slotId corresponding to each SIM

### 5.3.1.26 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](#) ()

### 5.3.1.26.1 Constructors and Destructors

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

### 5.3.1.26.2 Member Function Documentation

5.3.1.26.2.1 `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

5.3.1.26.2.2 `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

5.3.1.26.2.3 `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

5.3.1.26.2.4 `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

**5.3.1.26.2.5** `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>

**5.3.1.26.2.6** `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>

**5.3.1.26.2.7** `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> .
----	-------------	--

**5.3.1.26.2.8** `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>

#### Note

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

### 5.3.1.27 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 ()`

#### 5.3.1.27.1 Constructors and Destructors

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

#### 5.3.1.27.2 Member Function Documentation

5.3.1.27.2.1 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).

**5.3.1.27.2.2** `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.

**5.3.1.27.2.3** `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.

**5.3.1.27.2.4** `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.

**5.3.1.27.2.5** `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.

**5.3.1.27.2.6** `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.

**5.3.1.27.2.7** `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.

**5.3.1.27.2.8** `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.

**5.3.1.27.2.9** `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.

**Parameters**

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.

**5.3.1.27.2.10** `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.

**5.3.1.27.2.11** `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.

**5.3.1.28 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` ()

**5.3.1.28.1 Constructors and Destructors**

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

**5.3.1.28.2 Member Function Documentation**

**5.3.1.28.2.1** 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>

**5.3.1.29 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` ()

**5.3.1.29.1 Constructors and Destructors**

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

### 5.3.1.29.2 Member Function Documentation

#### 5.3.1.29.2.1 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>	<a href="#">OperatingMode</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::INTERNAL_ERR</a></li> <li><a href="#">telux::common::ErrorCode::NO_MEMORY</a></li> </ul>

### 5.3.1.30 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< [TdscdmaSignalStrengthInfo](#) > tdscdmaSignalStrengthInfo)
- std::shared\_ptr< [LteSignalStrengthInfo](#) > [getLteSignalStrength](#) ()
- std::shared\_ptr< [GsmSignalStrengthInfo](#) > [getGsmSignalStrength](#) ()
- std::shared\_ptr< [CdmaSignalStrengthInfo](#) > [getCdmaSignalStrength](#) ()
- std::shared\_ptr< [WcdmaSignalStrengthInfo](#) > [getWcdmaSignalStrength](#) ()
- std::shared\_ptr< [TdscdmaSignalStrengthInfo](#) > [getTdscdmaSignalStrength](#) ()

#### 5.3.1.30.1 Constructors and Destructors

##### 5.3.1.30.1.1 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< [TdscdmaSignalStrengthInfo](#) > [tdscdmaSignalStrengthInfo](#) )

#### 5.3.1.30.2 Member Function Documentation

**5.3.1.30.2.1** `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.

**5.3.1.30.2.2** `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.

**5.3.1.30.2.3** `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.

**5.3.1.30.2.4** `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.

**5.3.1.30.2.5** `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.



### 5.3.1.31 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

#### 5.3.1.31.1 Constructors and Destructors

**5.3.1.31.1.1** telux::tel::LteSignalStrengthInfo::LteSignalStrengthInfo ( int *lteSignalStrength*, int *lteRsrp*, int *lteRsrq*, int *lteRssnr*, int *lteCqi*, int *timingAdvance* )

#### 5.3.1.31.2 Member Function Documentation

**5.3.1.31.2.1** 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.

**5.3.1.31.2.2** 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.

**5.3.1.31.2.3 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.

**5.3.1.31.2.4 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.

**5.3.1.31.2.5 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.

**5.3.1.31.2.6 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).

**Returns**

LteCqI.

**5.3.1.31.2.7 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).

**Returns**

Timing advance value.

### 5.3.1.32 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 ()

#### 5.3.1.32.1 Constructors and Destructors

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

#### 5.3.1.32.2 Member Function Documentation

5.3.1.32.2.1 `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.

5.3.1.32.2.2 `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.

5.3.1.32.2.3 `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.

#### 5.3.1.32.2.4 `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).

##### Returns

GSM bit error rate.

#### 5.3.1.32.2.5 `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).

##### Returns

timing advance.

### 5.3.1.33 `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`

#### 5.3.1.33.1 Constructors and Destructors

5.3.1.33.1.1 `telux::tel::CdmaSignalStrengthInfo::CdmaSignalStrengthInfo ( int cdmaDbm, int cdmaEcio, int evdoDbm, int evdoEcio, int evdoSignalNoiseRatio )`

#### 5.3.1.33.2 Member Function Documentation

**5.3.1.33.2.1 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.

**5.3.1.33.2.2 const int telux::tel::CdmaSignalStrengthInfo::getDbm ( ) const**

Get the signal strength in dBm.

**Returns**

Minimum value of Evdo dBm and Cdma dBm.

**5.3.1.33.2.3 const int telux::tel::CdmaSignalStrengthInfo::getCdmaEcIo ( ) const**

Get the CDMA Ec/Io in dB.

**Returns**

CDMA Ec/Io.

**5.3.1.33.2.4 const int telux::tel::CdmaSignalStrengthInfo::getEvdoEcIo ( ) const**

Get the EVDO Ec/Io in dB.

**Returns**

EVDO Ec/Io.

**5.3.1.33.2.5 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.

**5.3.1.34 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

### 5.3.1.34.1 Constructors and Destructors

**5.3.1.34.1.1** `telux::tel::WcdmaSignalStrengthInfo::WcdmaSignalStrengthInfo ( int signalStrength, int bitErrorRate )`

### 5.3.1.34.2 Member Function Documentation

**5.3.1.34.2.1** `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.

**5.3.1.34.2.2** `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.

**5.3.1.34.2.3** `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.

#### 5.3.1.34.2.4 `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).

#### Returns

WCDMA bit error rate.

#### 5.3.1.35 `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`

#### 5.3.1.35.1 Constructors and Destructors

##### 5.3.1.35.1.1 `telux::tel::TdscdmaSignalStrengthInfo::TdscdmaSignalStrengthInfo ( int rscp )`

#### 5.3.1.35.2 Member Function Documentation

##### 5.3.1.35.2.1 `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.

#### 5.3.1.36 `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](#) ()

### 5.3.1.36.1 Constructors and Destructors

5.3.1.36.1.1 `telux::tel::VoiceServiceInfo::VoiceServiceInfo ( VoiceServiceState voiceServiceState, VoiceServiceDenialCause denialCause )`

### 5.3.1.36.2 Member Function Documentation

5.3.1.36.2.1 `VoiceServiceState telux::tel::VoiceServiceInfo::getVoiceServiceState ( )`

Get voice service state.

#### Returns

[VoiceServiceState](#)

5.3.1.36.2.2 `VoiceServiceDenialCause telux::tel::VoiceServiceInfo::getVoiceServiceDenialCause ( )`

Get Voice service denial cause

#### Returns

[VoiceServiceDenialCause](#)

5.3.1.36.2.3 `bool telux::tel::VoiceServiceInfo::isEmergency ( )`

Check if phone service is in emergency mode (i.e Only emergency numbers are allowed)

5.3.1.36.2.4 `bool telux::tel::VoiceServiceInfo::isInService ( )`

Check if phone is registered to home network or roaming network, phone is in service mode

5.3.1.36.2.5 `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** Success

**FAILURE** Generic failure

**MSD\_TRANSMISSION\_STARTED** MSD Transmission Started

**NACK\_OUT\_OF\_ORDER** Out of order NACK message detected

**ACK\_OUT\_OF\_ORDER** Out of order ACK message detected

**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\_GLOASS** eCall operating mode changed due to ERA-GLONASS operation

### 5.3.2.10 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

### 5.3.2.11 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

### 5.3.2.12 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

### 5.3.2.13 enum telux::tel::RATCapability [strong]

Defines all available RAT capabilities for each subscription

#### Enumerator

**AMPS**  
**CDMA**  
**HDR**  
**GSM**  
**WCDMA**  
**LTE**  
**TDS**

### 5.3.2.14 enum telux::tel::VoiceServiceTechnology [strong]

Defines all voice support available on device

#### Enumerator

**VOICE\_Tech\_GW\_CSFB**  
**VOICE\_Tech\_1x\_CSFB**  
**VOICE\_Tech\_VOLTE**

### 5.3.2.15 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

### 5.3.2.16 enum telux::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**

### 5.3.2.17 enum telux::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

### 5.3.2.18 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 `CallDirection::INCOMING`.

#### Parameters

in	<i>callback</i>	<p>- optional callback pointer to get the response of answer request below are possible error codes for callback response</p> <ul style="list-style-type: none"> <li>• <code>SUCCESS</code></li> <li>• <code>RADIO_NOT_AVAILABLE</code></li> <li>• <code>NO_MEMORY</code></li> <li>• <code>MODEM_ERR</code></li> <li>• <code>INTERNAL_ERR</code></li> <li>• <code>INVALID_STATE</code></li> <li>• <code>INVALID_CALL_ID</code></li> <li>• <code>INVALID_ARGUMENTS</code></li> <li>• <code>OPERATION_NOT_ALLOWED</code></li> <li>• <code>GENERIC_FAILURE</code></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 call. Only applicable for [CallDirection::INCOMING](#).

##### Parameters

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 [CallDirection::INCOMING](#).

##### 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 telux::common::Status telux::tel::ICall::hangup ( std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]`

Hang up the active call.

**Parameters**

in	<i>callback</i>	<p>- optional callback pointer to get the response of hangup 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 hangup i.e. success or suitable error code.

**5.4.1.1.2.7** `virtual telux::common::Status telux::tel::ICall::playDtmfTone ( char tone, std::shared_ptr< telux::common::ICommandResponseCallback > 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 telux::common::Status telux::tel::ICall::startDtmfTone ( char *tone*, std::shared\_ptr< telux::common::ICommandResponseCallback > *callback* = nullptr ) [pure virtual]**

Starts a continuous DTMF tone. To terminate the continuous DTMF tone, stopDtmfTone API needs to be invoked explicitly.

#### 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 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 dialing 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 `~ICallListener ()`

### 5.4.1.2.1 Constructors and Destructors

5.4.1.2.1.1 virtual `telx::tel::ICallListener::~ICallListener ( ) [virtual]`

### 5.4.1.2.2 Member Function Documentation

5.4.1.2.2.1 virtual void `telx::tel::ICallListener::onIncomingCall ( std::shared_ptr< ICall > call ) [virtual]`

This function is called when device receives an incoming call.

#### Parameters

in	<i>call</i>	- Pointer to <a href="#">ICall</a> instance
----	-------------	---

5.4.1.2.2.2 virtual void `telx::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 `telx::tel::ICallListener::onECallMsdTransmissionStatus ( int phoneId, telx::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 `telx::tel::ICallListener::onECallMsdTransmissionStatus ( int phoneId, telx::tel::ECallMsdTransmissionStatus msdTransmissionStatus ) [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>msdTransmission↔ Status</i>	- Indicates MSD Transmission status ECallMsdTransmissionStatus

**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::Status makeCall](#) (int phoneId, const std::string &dialNumber, std::shared\_ptr< [IMakeCallCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status makeECall](#) (int phoneId, const [ECallMsdData](#) &eCallMsdData, int category, int variant, std::shared\_ptr< [IMakeCallCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status makeECall](#) (int phoneId, const std::string dialNumber, const [ECallMsdData](#) &eCallMsdData, 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 updateECallMsd](#) (int phoneId, const [ECallMsdData](#) &eCallMsd, std::shared\_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status updateECallMsd](#) (int phoneId, const std::vector< uint8\_t > &msdPdu, [telux::common::ResponseCallback](#) 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::Status telux::tel::ICallManager::makeCall ( int phoneId, const std::string & dialNumber, std::shared_ptr< IMakeCallCallback > callback = nullptr ) [pure virtual]`

Initiate a voice call.

#### 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>	Optional callback pointer to get the response of makeCall request. Possible(not exhaustive) error codes for callback response <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.

5.4.1.3.2.2 `virtual telux::common::Status telux::tel::ICallManager::makeECall ( int phoneId, const E↔ CallMsdData & eCallMsdData, int category, int variant, std::shared_ptr< IMakeCallCallback > callback = nullptr ) [pure virtual]`

Initiate an emergency call to the emergency number(e.g. 112)



**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.

```
5.4.1.3.2.3 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 emergency call to the specified phone number. It is similar to a regular voice call, except that it facilitates MSD transmission.

**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>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>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.

```
5.4.1.3.2.4 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 emergency call with raw MSD pdu, to the emergency number(e.g. 112)

**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.

**5.4.1.3.2.5 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 emergency call with raw MSD pdu, to the specified phone number. It is similar to a regular voice call, except that it facilitates MSD transmission.

### 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.

**5.4.1.3.2.6** `virtual telx::common::Status telx::tel::ICallManager::updateECallMsd ( int phoneId, const ECallMsdData & eCallMsd, std::shared_ptr< telx::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.

**5.4.1.3.2.7** `virtual telx::common::Status telx::tel::ICallManager::updateECallMsd ( int phoneId, const std::vector< uint8_t > & msdPdu, telx::common::ResponseCallback callback ) [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>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.

**5.4.1.3.2.8** `virtual std::vector<std::shared_ptr<ICall> > telux::tel::ICallManager::getInProgressCalls ( ) [pure virtual]`

Get in-progress calls.

#### Returns

List of active calls.

**5.4.1.3.2.9** `virtual telux::common::Status telux::tel::ICallManager::conference ( std::shared_ptr< ICall > call1, std::shared_ptr< ICall > call2, std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]`

Merge two calls in a conference.

#### Parameters

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.

**5.4.1.3.2.10** `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.

**5.4.1.3.2.11** `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.

**5.4.1.3.2.12** `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 class telux::tel::SmsMessage

A Short Message Service message.

##### Public member functions

- [SmsMessage](#) (std::string text, std::string sender, std::string receiver, [SmsEncoding](#) encoding, std::string pdu)
- const std::string & [getText](#) () const
- const std::string & [getSender](#) () const
- const std::string & [getReceiver](#) () const
- [SmsEncoding](#) [getEncoding](#) () const
- const std::string & [getPdu](#) () const
- const std::string [toString](#) () const

##### 5.5.1.2.1 Constructors and Destructors

5.5.1.2.1.1 **telux::tel::SmsMessage::SmsMessage ( std::string text, std::string sender, std::string receiver, [SmsEncoding](#) encoding, std::string pdu )**

##### 5.5.1.2.2 Member Function Documentation

**5.5.1.2.2.1 const std::string& telux::tel::SmsMessage::getText ( ) const**

Get the message body.

**Returns**

String containing SMS message.

**5.5.1.2.2.2 const std::string& telux::tel::SmsMessage::getSender ( ) const**

Get the originating address (sender) of this SMS message.

**Returns**

String containing sender address.

**5.5.1.2.2.3 const std::string& telux::tel::SmsMessage::getReceiver ( ) const**

Get the destination address (receiver) of this SMS message.

**Returns**

String containing receiver address

**5.5.1.2.2.4 SmsEncoding telux::tel::SmsMessage::getEncoding ( ) const**

Get encoding used for this SMS message.

**Returns**

SMS message encoding used.

**5.5.1.2.2.5 const std::string& telux::tel::SmsMessage::getPdu ( ) const**

Get the raw PDU for the message.

**Returns**

String containing raw pdu data.

**5.5.1.2.2.6 const std::string telux::tel::SmsMessage::toString ( ) const**

Get the text related informative representation of this object.

**Returns**

String containing informative string.

### 5.5.1.3 class telux::tel::ISmsManager

SMS Manager class is the primary interface to send and receive SMS messages. It allows to send an SMS in several formats and sizes.

#### Public member functions

- virtual `telux::common::Status 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)=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` ()

#### 5.5.1.3.1 Constructors and Destructors

5.5.1.3.1.1 virtual telux::tel::ISmsManager::~ISmsManager ( ) [virtual]

#### 5.5.1.3.2 Member Function Documentation

5.5.1.3.2.1 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 destination address.

#### Parameters

in	<i>message</i>	Message or payload 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, This callback gives possible error codes.
in	<i>deliveryCallback</i>	Optional callback pointer to get message delivery status

#### Returns

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

**5.5.1.3.2.2 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.

**5.5.1.3.2.3 virtual telux::common::Status telux::tel::ISmsManager::setSmscAddress ( const std::string & *smscAddress*, 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>smscAddress</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.

**5.5.1.3.2.4 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.

**5.5.1.3.2.5 virtual int telux::tel::ISmsManager::getPhoneId ( ) [pure virtual]**

Get associated phone id for this SMSManager.

**Returns**

PhoneId.

**5.5.1.3.2.6 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.

**5.5.1.3.2.7 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.

**5.5.1.4 class telux::tel::ISmsListener**

A listener class for monitoring incoming SMS. 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 [onIncomingSms](#) (int phoneId, std::shared\_ptr< [SmsMessage](#) > message)
- virtual [~ISmsListener](#) ()

### 5.5.1.4.1 Constructors and Destructors

5.5.1.4.1.1 virtual telux::tel::ISmsListener::~ISmsListener ( ) [virtual]

### 5.5.1.4.2 Member Function Documentation

5.5.1.4.2.1 virtual void telux::tel::ISmsListener::onIncomingSms ( int *phoneId*, std::shared\_ptr< SmsMessage > *message* ) [virtual]

This function is called when device receives an incoming message

#### Parameters

in	<i>phoneId</i>	Unique identifier per phone
in	<i>SmsMessage</i>	Pointer to <a href="#">SmsMessage</a> object

### 5.5.1.5 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

### 5.5.1.5.1 Member Function Documentation

5.5.1.5.1.1 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** Message is made up of GSM7 septets

**GSM8** Message is made up of GSM8 septets

**UCS2** Message is made up of UCS2 septets

**UNKNOWN** Message encoding is unknown



## 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.

**Note**

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

**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.

**Note**

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

#### 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.

**Parameters**

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.

**Note**

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

#### 5.6.1.1.2.7 virtual telux::common::Status telux::tel::ICardApp::queryPin1LockState ( QueryPin1LockResponseCb callback ) [pure virtual]

Query Pin1 lock state.

**Parameters**

in	<i>callback</i>	Callback function to get the response of query pin1 lock state.
----	-----------------	---

**Note**

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

#### 5.6.1.1.2.8 virtual telux::common::Status telux::tel::ICardApp::queryFdnLockState ( QueryFdnLockResponseCb callback ) [pure virtual]

Query FDN lock state.

**Parameters**

in	<i>callback</i>	Callback function to get the response of query fdn lock state.
----	-----------------	--

**Note**

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

**5.6.1.1.2.9** virtual telux::common::Status telux::tel::lCardApp::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.

#### Note

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

### 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**

out	<i>count</i>	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**

out	<i>slotIds</i>	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=nullptr)=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>	<a href="#">CardState</a> - 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 p3 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 transmitApduLogicalChannel i.e. success or suitable status code.

**5.6.1.4.1.6** `virtual telux::common::Status telux::tel::ICard::transmitApduBasicChannel ( 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 p3 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 transmitApduBasicChannel 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

**Note**

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

#### 5.6.1.4.1.9 virtual telux::common::Status telux::tel::ICard::requestEid ( EidResponseCallback = nullptr ) [pure virtual]

Request eUICC identifier 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**

**Note**

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

#### 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::ICardChannelCallback::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>	<a href="#">IccResult</a> of open logical channel.

in	<i>error</i>	ErrorCode of the request.
----	--------------	---------------------------

### 5.6.1.6 class telux::tel::ICardCommandCallback

#### 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::ICardCommandCallback::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>	<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>

### 5.6.1.7 class telux::tel::ICardListener

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	slotId	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 [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 `~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 `telux::common::Status telux::tel::ISapCardManager::getState` ( `SapState & sapState` ) [`pure virtual`]

Get SIM access profile (SAP) client connection state.

#### Parameters

out	<i>sapState</i>	<a href="#">SapState</a> of the SIM Card
-----	-----------------	--

#### Returns

Status of `getState` i.e. success or suitable status code.

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

5.6.1.9.2.2 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 <code>requestSapState</code> .
-----	-----------------	---

#### Returns

Status of `requestSapState` i.e. success or suitable status code.

**5.6.1.9.2.3** `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.

**5.6.1.9.2.4** `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>• SUCCESS</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

**5.6.1.9.2.5** `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.

**5.6.1.9.2.6** `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 lc 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.

**5.6.1.9.2.7** `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.



**5.6.1.9.2.8** `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

<i>in</i>	<i>callback</i>	Optional callback to get the response for SIM power on.
-----------	-----------------	---

#### Returns

Status of requestSimPowerOn i.e. success or suitable status code.

**5.6.1.9.2.9** `virtual telux::common::Status telux::tel::ISapCardManager::requestSimReset ( std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]`

Send the SAP SIM reset request.

#### Parameters

<i>in</i>	<i>callback</i>	Optional callback to get the response for SIM reset
-----------	-----------------	---

#### Returns

Status of requestSimReset i.e. success or suitable status code.

**5.6.1.9.2.10** `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

<i>in</i>	<i>callback</i>	Optional callback to get the response for card reader status
-----------	-----------------	--

#### Returns

Status of requestCardReaderStatus i.e. success or suitable status code.

**5.6.1.9.2.11** `virtual int telux::tel::ISapCardManager::getSlotId ( ) [pure virtual]`

Get associated slot id for the SapCardManager.

#### Returns

SlotId

### 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

## 5.6.2 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::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

### 5.6.2.3 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

**5.6.2.4 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

**5.6.2.5 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

**5.6.2.6 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::ILocationConfigurator

[ILocationConfigurator](#) allows for the enablement/disablement of the time uncertainty. It also allows to set the threshold and the required power level for the `configureCTunc` API.

##### 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 `~ILocationConfigurator` ()

##### 5.7.1.1.1 Constructors and Destructors

5.7.1.1.1 virtual `telux::loc::ILocationConfigurator::~ILocationConfigurator` ( ) [`virtual`]

Destructor of [ILocationConfigurator](#)

##### 5.7.1.1.2 Member Function Documentation

5.7.1.1.2.1 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.

5.7.1.1.2.2 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.

**5.7.1.1.2.3 virtual telux::common::Status telux::loc::LocationConfigurator::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.

#### 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.

### 5.7.1.2 struct telux::loc::GnssKinematicsData

Specifies kinematics related information.

#### Data fields

Type	Field	Description
<a href="#">Kinematic↔ DataValidity</a>	<a href="#">bodyFrame↔ DataMask</a>	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

### 5.7.1.3 struct telux::loc::TimeInfo

#### Data fields

Type	Field	Description
GnssTime↔ Validity	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.

### 5.7.1.4 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
TimeValidity	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
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.

Type	Field	Description
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

### 5.7.1.5 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="#">GlonassTimeInfo</a>	glo	

### 5.7.1.6 struct telux::loc::SystemTime

#### Data fields

Type	Field	Description
<a href="#">GnssSystem</a>	gnssSystem↔ TimeSrc	Specifies GNSS system time reported. Mandatory field
<a href="#">SystemTimeInfo</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

### 5.7.1.7 struct telux::loc::GnssMeasurementInfo

#### Data fields

Type	Field	Description
<a href="#">GnssSignal</a>	gnssSignalType	GnssSignalType mask
<a href="#">GnssSystem</a>	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

### 5.7.1.8 struct telux::loc::GnssData



**Data fields**

Type	Field	Description
<a href="#">GnssDataValidity</a>	<a href="#">gnssDataMask</a> [ <a href="#">GnssDataSignalTypes::GNSS_DATA_MAX_NUMBER_OF_SIGNAL_TYPES</a> ]	bitwise OR of <a href="#">GnssDataValidityType</a>
double	<a href="#">jammerInd</a> [ <a href="#">GnssDataSignalTypes::GNSS_DATA_MAX_NUMBER_OF_SIGNAL_TYPES</a> ]	Jammer Indication Each index represents the measurement for the signal type in <a href="#">GnssDataSignalTypes</a>
double	<a href="#">agc</a> [ <a href="#">GnssDataSignalTypes::GNSS_DATA_MAX_NUMBER_OF_SIGNAL_TYPES</a> ]	Automatic gain control Each index represents the measurement for the signal type in <a href="#">GnssDataSignalTypes</a>

**5.7.1.9 class [telux::loc::IGpsTime](#)**

[IGpsTime](#) provides interface to get current GPS week and elapsed time in current GPS week.

**Public member functions**

- virtual [uint32\\_t](#) [getWeek](#) ()=0
- virtual [uint32\\_t](#) [getTimeOfWeekMsec](#) ()=0

**5.7.1.9.1 Member Function Documentation****5.7.1.9.1.1 virtual [uint32\\_t](#) [telux::loc::IGpsTime::getWeek](#) ( ) [pure virtual]**

Retrieves current GPS week as calculated from midnight, Jan 6, 1980.

**Returns**

Unsigned 32-bit integer containing week number.

#### 5.7.1.9.1.2 virtual uint32\_t telux::loc::IGpsTime::getTimeOfWeekMsec ( ) [pure virtual]

Retrieves elapsed time in the current GPS week starting from 12:00 am on Sunday.

#### Returns

Unsigned 32-bit integer containing time in milliseconds.

### 5.7.1.10 class telux::loc::ISensorDataUsage

Specifies the sensors used for calculating the fixes and the type of measurements which were aided by sensor data.

#### Public member functions

- virtual [SensorType](#) getSensorType ()=0
- virtual [Measurement](#) getMeasurement ()=0

#### 5.7.1.10.1 Member Function Documentation

##### 5.7.1.10.1.1 virtual SensorType telux::loc::ISensorDataUsage::getSensorType ( ) [pure virtual]

Retrieves which sensors were used in calculating the position in the position report.

#### Returns

[SensorType](#) if available.

##### 5.7.1.10.1.2 virtual Measurement telux::loc::ISensorDataUsage::getMeasurement ( ) [pure virtual]

Retrieves which measurements were aided by sensor data.

#### Returns

Measurement types if available.

### 5.7.1.11 class telux::loc::ILocationInfo

[ILocationInfo](#) provides interface to get basic position related information like latitude, longitude, altitude, timestamp and other information like time stamp, session status,.

**Public member functions**

- virtual [PositionTech](#) getPositionTechnology ()=0
- virtual double [getLatitude](#) ()=0
- virtual double [getLongitude](#) ()=0
- virtual double [getAltitude](#) ()=0
- virtual float [getHeading](#) ()=0
- virtual float [getVerticalUncertainty](#) ()=0
- virtual uint64\_t [getTimeStamp](#) ()=0
- virtual float [getSpeedUncertainty](#) ()=0
- virtual float [getHeadingUncertainty](#) ()=0
- virtual float [getAltitudeMeanSeaLevel](#) ()=0
- virtual float [getPositionDop](#) ()=0
- virtual float [getHorizontalDop](#) ()=0
- virtual float [getVerticalDop](#) ()=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 void [getSVIds](#) (std::vector< uint16\_t > &idsOfUsedSVs)=0
- virtual [SbasCorrection](#) getSbasCorrection ()=0
- virtual [SessionStatus](#) getSessionStatus ()=0
- virtual [telux::common::Status](#) getLeapSeconds (uint8\_t &leapSeconds)=0
- virtual std::shared\_ptr< [IGpsTime](#) > [getGpsTime](#) ()=0
- virtual [telux::common::Status](#) getCircularHorizontalUncertainty (float &circularHorizontalUncertainty)=0
- virtual [telux::common::Status](#) getHorizontalConfidence (uint8\_t &horizontalConfidence)=0
- virtual float [getHorizontalSpeed](#) ()=0
- virtual [telux::common::Status](#) getVerticalConfidence (uint8\_t &verticalConfidence)=0
- virtual float [getVerticalSpeed](#) ()=0
- virtual [telux::common::Status](#) getSensorDataUsage (std::shared\_ptr< [ISensorDataUsage](#) > &sensorDataUsage)=0

- virtual `telux::common::Status getFixId (uint32_t &fixId)=0`
- virtual `telux::common::Status getVelocityEastNorthUp (std::vector< float > &velocityEastNorthUp)=0`
- virtual `telux::common::Status getVelocityUncertaintyEastNorthUp (std::vector< float > &velocityUncertaintyEastNorthUp)=0`

### 5.7.1.11.1 Member Function Documentation

#### 5.7.1.11.1.1 virtual `PositionTech telux::loc::ILocationInfo::getPositionTechnology ( ) [pure virtual]`

Retrieves technology used in computing this fix.

#### Returns

Position technology.

#### 5.7.1.11.1.2 virtual `double telux::loc::ILocationInfo::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.

#### 5.7.1.11.1.3 virtual `double telux::loc::ILocationInfo::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.

**5.7.1.11.1.4 virtual double telux::loc::ILocationInfo::getAltitude ( ) [pure virtual]**

Retrieves altitude above the WGS 84 reference ellipsoid.

- Units: Meters

**Returns**

Altitude if available else returns NaN.

**5.7.1.11.1.5 virtual float telux::loc::ILocationInfo::getHeading ( ) [pure virtual]**

Retrieves heading.

- Units: Degrees
- Range: 0 to 359.999

**Returns**

Heading if available else returns NaN.

**5.7.1.11.1.6 virtual float telux::loc::ILocationInfo::getVerticalUncertainty ( ) [pure virtual]**

Retrieves the vertical uncertainty.

- Units: Meters

**Returns**

Vertical uncertainty if available else returns NaN.

**5.7.1.11.1.7 virtual uint64\_t telux::loc::ILocationInfo::getTimeStamp ( ) [pure virtual]**

Retrieves UTC timeStamp for the location fix.

- Units: Milliseconds since Jan 1, 1970

**Returns**

TimeStamp in seconds if available else returns 0 (as UTC timeStamp has elapsed since January 1, 1970, it cannot be 0)

**5.7.1.11.1.8 virtual float telux::loc::!LocationInfo::getSpeedUncertainty ( ) [pure virtual]**

Retrieves 3-D speed uncertainty.

- Units: Meters per Second

**Returns**

Speed uncertainty if available else returns NaN.

**5.7.1.11.1.9 virtual float telux::loc::!LocationInfo::getHeadingUncertainty ( ) [pure virtual]**

Retrieves heading uncertainty.

- Units: Degrees
- Range: 0 to 359.999

**Returns**

Heading uncertainty if available else returns NaN.

**5.7.1.11.1.10 virtual float telux::loc::!LocationInfo::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.

**5.7.1.11.1.11 virtual float telux::loc::!LocationInfo::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)

**5.7.1.11.1.12 virtual float telux::loc::!LocationInfo::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)

**5.7.1.11.1.13 virtual float telux::loc::ILocationInfo::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)

**5.7.1.11.1.14 virtual float telux::loc::ILocationInfo::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

**5.7.1.11.1.15 virtual LocationReliability telux::loc::ILocationInfo::getHorizontalReliability ( ) [pure virtual]**

Specifies the reliability of the horizontal position.

**Returns**

[LocationReliability](#) of the horizontal position if available else returns UNKNOWN.

**5.7.1.11.1.16 virtual LocationReliability telux::loc::ILocationInfo::getVerticalReliability ( ) [pure virtual]**

Specifies the reliability of the vertical position.

**Returns**

[LocationReliability](#) of the vertical position if available else returns UNKNOWN.

**5.7.1.11.1.17 virtual float telux::loc::ILocationInfo::getHorizontalUncertaintySemiMajor ( ) [pure virtual]**

Retrieves semi-major axis of horizontal elliptical uncertainty.

- Units: Meters

**Returns**

Semi-major horizontal elliptical uncertainty if available else returns NaN.

**5.7.1.11.1.18 virtual float telux::loc::ILocationInfo::getHorizontalUncertaintySemiMinor ( ) [pure virtual]**

Retrieves semi-minor axis of horizontal elliptical uncertainty.

- Units: Meters

#### Returns

Semi-minor horizontal elliptical uncertainty if available else returns NaN.

**5.7.1.11.1.19 virtual float telux::loc::ILocationInfo::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.

**5.7.1.11.1.20 virtual void telux::loc::ILocationInfo::getSVIds ( std::vector< uint16\_t > & idsOfUsedSVs ) [pure virtual]**

Retrieves GNSS Satellite Vehicles used in position data.

#### Parameters

out	<i>idsOfUsedSVs</i>	Vector of Satellite Vehicle identifiers.
-----	---------------------	--

**5.7.1.11.1.21 virtual SbasCorrection telux::loc::ILocationInfo::getSbasCorrection ( ) [pure virtual]**

Retrieves navigation solution mask used to indicate SBAS corrections.

#### Returns

- SBAS (Satellite Based Augmentation System) Correction mask used.

**5.7.1.11.1.22 virtual SessionStatus telux::loc::ILocationInfo::getSessionStatus ( ) [pure virtual]**

Retrieves status of the session that is requested by user application.

#### Returns

[SessionStatus](#)



**5.7.1.11.1.23 virtual telux::common::Status telux::loc::ILocationInfo::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.

**5.7.1.11.1.24 virtual std::shared\_ptr<IGpsTime> telux::loc::ILocationInfo::getGpsTime ( ) [pure virtual]**

Retrieves GPS time structure.

**Returns**

Pointer of [IGpsTime](#) object if available else returns null pointer.

**5.7.1.11.1.25 virtual telux::common::Status telux::loc::ILocationInfo::getCircularHorizontalUncertainty ( float & circularHorizontalUncertainty ) [pure virtual]**

Retrieves horizontal position uncertainty (circular) if available.

**Parameters**

out	<i>circularHorizontalUncertainty</i>	- circular horizontal uncertainty • Units: Meters
-----	--------------------------------------	--

**Returns**

Status of circular horizontal uncertainty.

**5.7.1.11.1.26 virtual telux::common::Status telux::loc::ILocationInfo::getHorizontalConfidence ( uint8\_t & horizontalConfidence ) [pure virtual]**

Retrieves horizontal uncertainty confidence if available.

**Parameters**

out	<i>horizontalConfidence</i>	- horizontal uncertainty confidence <ul style="list-style-type: none"> <li>• Units: Percent</li> <li>• Range: 0 to 99</li> </ul>
-----	-----------------------------	---

**Returns**

Status of horizontal uncertainty confidence.

**5.7.1.11.1.27 virtual float telux::loc::ILocationInfo::getHorizontalSpeed ( ) [pure virtual]**

Retrieves horizontal speed.

- Units: Meters/second

**Returns**

horizontal speed if available else returns NaN.

**5.7.1.11.1.28 virtual telux::common::Status telux::loc::ILocationInfo::getVerticalConfidence ( uint8\_t & verticalConfidence ) [pure virtual]**

Retrieves vertical uncertainty confidence if available.

**Parameters**

out	<i>verticalConfidence</i>	- vertical uncertainty confidence <ul style="list-style-type: none"> <li>• Units: Percent</li> <li>• Range: 0 to 99</li> </ul>
-----	---------------------------	---

**Returns**

Status of vertical uncertainty confidence.

**5.7.1.11.1.29 virtual float telux::loc::ILocationInfo::getVerticalSpeed ( ) [pure virtual]**

Retrieves vertical speed.

- Units: Meters/second

**Returns**

Float containing vertical speed if available else returns NaN.

**5.7.1.11.1.30** `virtual telux::common::Status telux::loc::LocationInfo::getSensorDataUsage ( std::shared_ptr< ISensorDataUsage > & sensorDataUsage ) [pure virtual]`

Retrieves sensor data was used in computing the position if available.

#### Parameters

out	<i>sensorDataUsage</i>	- which sensors were used in calculating the position
-----	------------------------	---

#### Returns

Status of availability of sensorDataUsage.

**5.7.1.11.1.31** `virtual telux::common::Status telux::loc::LocationInfo::getFixId ( uint32_t & fixId ) [pure virtual]`

Retrieves fix count if available. Fix count of a session starts with 0 and increments by one for each successive position report for a particular session.

#### Parameters

out	<i>fixId</i>	- identifier of fix for session
-----	--------------	---------------------------------

#### Returns

Status of availability of fix identifier.

**5.7.1.11.1.32** `virtual telux::common::Status telux::loc::LocationInfo::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.

**5.7.1.11.1.33** `virtual telux::common::Status telux::loc::LocationInfo::getVelocityUncertaintyEastNorthUp ( std::vector< float > & velocityUncertaintyEastNorthUp ) [pure virtual]`

Retrieves east, North, Up velocity uncertainty if available.

**Parameters**

out	<i>velocity</i> ↔ <i>UncertaintyEast</i> ↔ <i>NorthUp</i>	- east, North, Up velocity uncertainty Units: Meters/second
-----	---	---

**Returns**

Status of availability of east, North, Up velocity uncertainty.

**5.7.1.12 class telux::loc::ILocationInfoBase**

[ILocationInfoBase](#) provides interface to get basic position related information like latitude, longitude, altitude, timestamp.

**Public member functions**

- 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

**5.7.1.12.1 Member Function Documentation****5.7.1.12.1.1 virtual LocationTechnology telux::loc::ILocationInfoBase::getTechMask ( ) [pure virtual]**

Retrieves technology used in computing this fix.

**Returns**

Location technology mask.

**5.7.1.12.1.2 virtual float telux::loc::ILocationInfoBase::getSpeed ( ) [pure virtual]**

Retrieves Speed.

**Returns**

speed in meters per second.

**5.7.1.12.1.3 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.

**5.7.1.12.1.4 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.

**5.7.1.12.1.5 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.

**5.7.1.12.1.6 virtual float telux::loc::ILocationInfoBase::getHeading ( ) [pure virtual]**

Retrieves heading.

- Units: Degrees
- Range: 0 to 359.999

**Returns**

Heading if available else returns NaN.

**5.7.1.12.1.7 virtual float telux::loc::ILocationInfoBase::getHorizontalUncertainty ( ) [pure virtual]**

Retrieves the horizontal uncertainty.

**Returns**

Horizontal uncertainty.

**5.7.1.12.1.8 virtual float telux::loc::ILocationInfoBase::getVerticalUncertainty ( ) [pure virtual]**

Retrieves the vertical uncertainty.

- Units: Meters

**Returns**

Vertical uncertainty if available else returns NaN.

**5.7.1.12.1.9 virtual uint64\_t telux::loc::ILocationInfoBase::getTimeStamp ( ) [pure virtual]**

Retrieves UTC timeStamp for the location fix.

- Units: Milliseconds since Jan 1, 1970

**Returns**

TimeStamp in seconds if available else returns 0 (as UTC timeStamp has elapsed since January 1, 1970, it cannot be 0)

**5.7.1.12.1.10 virtual float telux::loc::ILocationInfoBase::getSpeedUncertainty ( ) [pure virtual]**

Retrieves 3-D speed uncertainty.

- Units: Meters per Second

**Returns**

Speed uncertainty if available else returns NaN.

**5.7.1.12.1.11 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.

**5.7.1.13 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 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 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`

### 5.7.1.13.1 Member Function Documentation

#### 5.7.1.13.1.1 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.

#### 5.7.1.13.1.2 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)

#### 5.7.1.13.1.3 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)



**5.7.1.13.1.4 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)

**5.7.1.13.1.5 virtual float telux::loc::ILocationInfoEx::getGeometricDop( ) [pure virtual]**

Retrieves geometric dilution of precision.

**Returns**

geometric dilution of precision.

**5.7.1.13.1.6 virtual float telux::loc::ILocationInfoEx::getTimeDop( ) [pure virtual]**

Retrieves time dilution of precision.

**Returns**

Time dilution of precision.

**5.7.1.13.1.7 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

**5.7.1.13.1.8 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.

**5.7.1.13.1.9 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.

**5.7.1.13.1.10 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.

**5.7.1.13.1.11 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.

**5.7.1.13.1.12 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.

**5.7.1.13.1.13 virtual float telux::loc::ILocationInfoEx::getEastStandardDeviation ( ) [pure virtual]**

Retrieves east standard deviation.

- Units: Meters

**Returns**

East Standard Deviation.

**5.7.1.13.1.14** `virtual float telux::loc::ILocationInfoEx::getNorthStandardDeviation ( ) [pure virtual]`

Retrieves north standard deviation.

- Units: Meters

#### Returns

North Standard Deviation.

**5.7.1.13.1.15** `virtual void telux::loc::ILocationInfoEx::getSVIds ( std::vector< uint16_t > & idsOfUsedSVs ) [pure virtual]`

Retrieves GNSS Satellite Vehicles used in position data.

#### Parameters

out	<i>idsOfUsedSVs</i>	Vector of Satellite Vehicle identifiers.
-----	---------------------	--

**5.7.1.13.1.16** `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.

**5.7.1.13.1.17** `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.

**5.7.1.13.1.18** `virtual GnssKinematicsData telux::loc::ILocationInfoEx::getBodyFrameData ( ) [pure virtual]`

Retrieves position related information.

**5.7.1.13.1.19** `virtual std::vector<GnssMeasurementInfo> telux::loc::ILocationInfoEx::getmeasUsageInfo ( ) [pure virtual]`

Retrieves gnss measurement usage info.

**5.7.1.13.1.20 virtual SystemTime telux::loc::ILocationInfoEx::getGnssSystemTime ( ) [pure virtual]**

Retrieves type of gnss system.

**Returns**

- Type of Gnss System.

**5.7.1.13.1.21 virtual float telux::loc::ILocationInfoEx::getTimeUncMs ( ) [pure virtual]**

Retrieves time uncertainty.

**Returns**

- Time uncertainty in milliseconds.

**5.7.1.13.1.22 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.

**5.7.1.13.1.23 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.

**5.7.1.13.1.24** `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.

**5.7.1.13.1.25** `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.

**5.7.1.13.1.26** `virtual DrCalibrationStatus telux::loc::ILocationInfoEx::getCalibrationStatus ( ) [pure virtual]`

Sensor calibration status.

#### Returns

mask indicating the calibration status with respect to different parameters.

**5.7.1.13.1.27** `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.

**5.7.1.13.1.28** `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.

### 5.7.1.14 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`

#### 5.7.1.14.1 Member Function Documentation

##### 5.7.1.14.1.1 virtual [GnssConstellationType](#) `telux::loc::ISVInfo::getConstellation ( )` [pure virtual]

Indicates to which constellation this satellite vehicle belongs.

#### Returns

[GnssConstellationType](#) if available else returns UNKNOWN.

##### 5.7.1.14.1.2 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)

**5.7.1.14.1.3 virtual SVHealthStatus telux::loc::ISVInfo::getSVHealthStatus ( ) [pure virtual]**

Health status of satellite vehicle.

**Returns**

HealthStatus of Satellite Vehicle if available else returns UNKNOWN.

- [SVHealthStatus](#)

**5.7.1.14.1.4 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](#)

**5.7.1.14.1.5 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.

**5.7.1.14.1.6 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.

**5.7.1.14.1.7 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.

**5.7.1.14.1.8 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.

**5.7.1.14.1.9 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.

**5.7.1.14.1.10 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.

**5.7.1.14.1.11 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.

**5.7.1.14.1.12 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.



### 5.7.1.15 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`

#### 5.7.1.15.1 Member Function Documentation

##### 5.7.1.15.1.1 virtual [AltitudeType](#) `telux::loc::IGnssSVInfo::getAltitudeType ( ) [pure virtual]`

Indicates whether altitude is assumed or calculated.

#### Returns

[AltitudeType](#) if available else returns UNKNOWN.

##### 5.7.1.15.1.2 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.

### 5.7.1.16 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`

#### 5.7.1.16.1 Member Function Documentation

##### 5.7.1.16.1.1 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.

### 5.7.1.17 class telux::loc::LocationFactory

[LocationFactory](#) allows creation of location manager.

#### Public member functions

- `std::shared_ptr< ILocationManager > getLocationManager ()`
- `std::shared_ptr< ILocationConfigurator > getLocationConfigurator ()`
- `~LocationFactory ()`

#### Static Public Member Functions

- `static LocationFactory & getInstance ()`

#### 5.7.1.17.1 Constructors and Destructors

5.7.1.17.1.1 `telux::loc::LocationFactory::~~LocationFactory ( )`

#### 5.7.1.17.2 Member Function Documentation

5.7.1.17.2.1 `static LocationFactory& telux::loc::LocationFactory::getInstance ( ) [static]`

Get Location Factory instance.

5.7.1.17.2.2 `std::shared_ptr<ILocationManager> telux::loc::LocationFactory::getLocationManager ( )`

Get instance of Location Manager

#### Returns

Pointer of [ILocationManager](#) object.

5.7.1.17.2.3 `std::shared_ptr<ILocationConfigurator> telux::loc::LocationFactory::getLocationConfigurator ( )`

Get instance of Location Configurator.

#### Returns

Pointer of [ILocationConfigurator](#) object.

### 5.7.1.18 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 `onLocationUpdate` (const std::shared\_ptr< [ILocationInfo](#) > &locationInfo)
- virtual void `onBasicLocationUpdate` (const std::shared\_ptr< [ILocationInfoBase](#) > &locationInfo)
- virtual void `onDetailedLocationUpdate` (const std::shared\_ptr< [ILocationInfoEx](#) > &locationInfo)
- 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 `~ILocationListener` ()

### 5.7.1.18.1 Constructors and Destructors

**5.7.1.18.1.1** virtual `telux::loc::ILocationListener::~ILocationListener` ( ) [`virtual`]

Destructor of [ILocationListener](#)

### 5.7.1.18.2 Member Function Documentation

**5.7.1.18.2.1** virtual void `telux::loc::ILocationListener::onLocationUpdate` ( const std::shared\_ptr< [ILocationInfo](#) > & *locationInfo* ) [`virtual`]

This function is called when device receives location update.

#### Parameters

in	<i>locationInfo</i>	- Location information like latitude, longitude, timeStamp other information such as heading, altitude and velocity etc.
----	---------------------	---

**5.7.1.18.2.2** 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, timeStamp other information such as heading, altitude and velocity etc.
----	---------------------	---

**5.7.1.18.2.3** 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, timeStamp, heading, altitude, velocity and other information such as deviations, elliptical accuracies etc.
----	---------------------	---

#### 5.7.1.18.2.4 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
----	-------------------	------------------------------

#### 5.7.1.18.2.5 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
----	-------------	--------------------

#### 5.7.1.18.2.6 virtual void telux::loc::ILocationListener::onGnssNmeaInfo ( uint64\_t timestamp, const std::string & nmea ) [virtual]

This function is called when device receives GNSS NMEA sentences.

**Parameters**

in	<i>timestamp</i>	- Timestamp
in	<i>nmea</i>	- Nmea sentence

**Note**

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

### 5.7.1.19 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 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)`=0
- virtual `telux::common::Status startDetailedEngineReports (uint32_t interval, LocReqEngine engineType, telux::common::ResponseCallback callback)`=0
- virtual `telux::common::Status startBasicReports (uint32_t distanceInMeters, uint32_t intervalInMs, telux::common::ResponseCallback callback)`=0
- virtual `telux::common::Status stopReports (telux::common::ResponseCallback callback)`=0
- virtual `telux::common::Status registerListener (std::weak_ptr< ILocationListener > listener)`=0
- virtual `telux::common::Status removeListener (std::weak_ptr< ILocationListener > listener)`=0
- virtual `telux::common::Status setPositionReportTimeout (uint32_t timeout, std::shared_ptr< telux::common::ICommandResponseCallback > callback=nullptr)`=0
- virtual `telux::common::Status setHorizontalAccuracyLevel (HorizontalAccuracyLevel accuracy=HorizontalAccuracyLevel::LOW, std::shared_ptr< telux::common::ICommandResponseCallback > callback=nullptr)`=0
- virtual `telux::common::Status setMinIntervalForReports (uint32_t minInterval, std::shared_ptr< telux::common::ICommandResponseCallback > callback=nullptr)`=0
- virtual `uint32_t getPositionReportTimeout ()`=0
- virtual `HorizontalAccuracyLevel getHorizontalAccuracyLevel ()`=0
- virtual `uint32_t getMinIntervalForFinalReports ()`=0
- virtual `~ILocationManager ()`

**5.7.1.19.1 Constructors and Destructors**

**5.7.1.19.1.1** virtual `telux::loc::ILocationManager::~~ILocationManager ( ) [virtual]`

Destructor of `ILocationManager`

**5.7.1.19.2 Member Function Documentation**

**5.7.1.19.2.1 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.

**5.7.1.19.2.2 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.

**5.7.1.19.2.3 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.

**5.7.1.19.2.4 virtual telux::common::Status telux::loc::ILocationManager::deRegisterListenerEx ( std::weak\_ptr< ILocationListener > *listener* ) [pure virtual]**

Remove a previously registered listener.

**Parameters**

in	<i>listener</i>	- Previously registered <a href="#">ILocationListener</a> that needs to be removed.
----	-----------------	---

**Returns**

Status of removeListener success or suitable status code

#### 5.7.1.19.2.5 `virtual telux::common::Status telux::loc::ILocationManager::startDetailedReports ( uint32_t interval, telux::common::ResponseCallback callback ) [pure virtual]`

Starts the richer location reports by configuring the time between them as the interval.

This Api enables the `onDetailedLocationUpdate`, `onGnssSVInfo` and `onGnssSignalInfo` Apis on the listener.

##### 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.
----	-----------------	--

##### Returns

Status of `startDetailedReports` i.e. success or suitable status code.

#### 5.7.1.19.2.6 `virtual telux::common::Status telux::loc::ILocationManager::startDetailedEngineReports ( uint32_t interval, LocReqEngine engineType, telux::common::ResponseCallback callback ) [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.

This Api enables the `onDetailedLocationUpdate`, `onGnssSVInfo` and `onGnssSignalInfo` Apis on the listener.

##### 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.

**Returns**

Status of startDetailedEngineReports i.e. success or suitable status code.

**5.7.1.19.2.7** `virtual telux::common::Status telux::loc::ILocationManager::startBasicReports ( uint32_t distanceInMeters, uint32_t intervalInMs, telux::common::ResponseCallback callback ) [pure virtual]`

Starts the Location report by configuring the time and distance between the consecutive reports.

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**

in	<i>callback</i>	- Optional callback to get the response of set minimum distance for reports.
----	-----------------	--

**Returns**

Status of startBasicReports i.e. success or suitable status code.

**5.7.1.19.2.8** `virtual telux::common::Status telux::loc::ILocationManager::stopReports ( telux::common::ResponseCallback callback ) [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.



### 5.7.1.19.2.9 virtual telux::common::Status telux::loc::ILocationManager::registerListener ( std::weak\_ptr< ILocationListener > *listener* ) [pure virtual]

Register a listener for specific updates from location manager like location and satellite vehicle info. As soon as the first listener is registered, position fixes will start being reported.

#### 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.

**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.

### 5.7.1.19.2.10 virtual telux::common::Status telux::loc::ILocationManager::removeListener ( std::weak\_ptr< ILocationListener > *listener* ) [pure virtual]

Remove a previously registered listener.

#### Parameters

in	<i>listener</i>	- Previously registered <a href="#">ILocationListener</a> that needs to be removed
----	-----------------	--

#### Returns

Status of removeListener success or suitable status code

**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.

### 5.7.1.19.2.11 virtual telux::common::Status telux::loc::ILocationManager::setPositionReportTimeout ( uint32\_t *timeout*, std::shared\_ptr< telux::common::ICommandResponseCallback > *callback = nullptr* ) [pure virtual]

Configures position report timeout. LocationManager tries to determine the position until the position report timeout has elapsed. If the final position cannot be determined before the timeout period, it returns a position report with session status marked as [SessionStatus::TIMEOUT](#) instead of [SessionStatus::SUCCESS](#). Position report timeout is an app specific setting. E.g. If this API is called with timeout parameter value 5000, the LocationManager tries to determine the position before 5 seconds. If position report is determined, a position report will be returned with sessionStatus=[SessionStatus::SUCCESS](#). Otherwise, a position report will be sent with sessionStatus=[SessionStatus::TIMEOUT](#) after 5 seconds.

**Parameters**

in	<i>timeout</i>	- Maximum time to get a position report in milliseconds.
in	<i>callback</i>	- Optional callback to get the response of set position report time out

**Returns**

Status as SUCCESS for setPositionReportTimeout.

**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.

**5.7.1.19.2.12** `virtual telux::common::Status telux::loc::ILocationManager::setHorizontalAccuracyLevel ( HorizontalAccuracyLevel accuracy = HorizontalAccuracyLevel::LOW, std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]`

Configuring horizontal accuracy level. If the final position cannot be determined with the required horizontal accuracy level within the timeout period specified in the setPositionReportTimeout API, a timeout fix will be provided. Refer to setPositionReportTimeout API for more details.

**Parameters**

in	<i>accuracy</i>	- <a href="#">HorizontalAccuracyLevel</a>
in	<i>callback</i>	- Optional callback to get the response of set horizontal accuracy level

**Returns**

Status as SUCCESS of setHorizontalAccuracyLevel.

**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.

**5.7.1.19.2.13** `virtual telux::common::Status telux::loc::ILocationManager::setMinIntervalForReports ( uint32_t minInterval, std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]`

Configuring minimum time interval between two consecutive position reports.

**Parameters**

in	<i>minInterval</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.
----	--------------------	---

in	<i>callback</i>	- Optional callback to get the response of set minimum interval for reports.
----	-----------------	--

**Returns**

Status of setMinIntervalForReports i.e. success or suitable status code.

**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.

**5.7.1.19.2.14** `virtual uint32_t telux::loc::ILocationManager::getPositionReportTimeout ( ) [pure virtual]`

Get timeout of a position report.

**Returns**

Maximum time to get a position report.

**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.

**5.7.1.19.2.15** `virtual HorizontalAccuracyLevel telux::loc::ILocationManager::getHorizontalAccuracyLevel ( ) [pure virtual]`

Get horizontal accuracy level of a location fix.

**Returns**

[HorizontalAccuracyLevel](#).

**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.

**5.7.1.19.2.16** `virtual uint32_t telux::loc::ILocationManager::getMinIntervalForFinalReports ( ) [pure virtual]`

Get the time interval between final reports.

**Returns**

Minimum time interval between final position reports.

**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.

## 5.7.2 Enumeration Type Documentation

### 5.7.2.1 enum telux::loc::FixRecurrence [strong]

Defines recurrence type of the fix. Obsolete

#### Enumerator

**PERIODIC** Request periodic fixes, minimum interval between final reports will be the periodicity.

Client can configure it using LocationService API i.e. setMinIntervalForFinalReports

**SINGLE** Request a single fix

### 5.7.2.2 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

### 5.7.2.3 enum telux::loc::PositionTechType

Defines technology used in computing the location fix.

#### Enumerator

**SATELLITE** Satellites used to generate the fix

**CELLID** Cell towers used to generate the fix

**WIFI** Wi-Fi access points used to generate the fix

**SENSORS** Sensors used to generate the fix

**REFERENCE\_LOCATION** Reference location used to generate the fix

**INJECTED\_COARSE\_POSITION** Coarse position injected into the location engine used to generate the fix

**AFLT** Advanced Forward Link Trilateration(AFLT), the phone takes measurements of signals from nearby towers and reports the time/distance readings back to the network to generate the fix

**HYBRID** GNSS and network-provided measurements used to generate the fix

**TECH\_COUNT** Bitset

### 5.7.2.4 enum telux::loc::LocationReliability [strong]

Specifies the reliability of the position.

#### Enumerator

**UNKNOWN**

**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\_COUNT** Bitset

### 5.7.2.6 enum telux::loc::SessionStatus [strong]

Defines status of the session that is requested by user application.

#### Enumerator

**UNKNOWN**  
**SUCCESS** Session successful  
**IN\_PROGRESS** Session is still in progress, further position reports will be generated until either the fix criteria specified by the client are met or the client response time out occurs  
**GENERAL\_FAILURE** Session failed  
**TIMEOUT** Fix request failed because the session timed out  
**USER\_END** Fix request failed because the session was ended by the user  
**BAD\_PARAMETER** Fix request failed due to bad parameters in the request  
**PHONE\_OFFLINE** Fix request failed because the phone is offline  
**ENGINE\_LOCKED** Fix request failed because the engine is locked

### 5.7.2.7 enum telux::loc::AltitudeType [strong]

Indicates whether altitude is assumed or calculated.

#### Enumerator

**UNKNOWN**  
**CALCULATED** Altitude is calculated  
**ASSUMED** Altitude is assumed, there may not be enough satellites to determine the precise altitude

### 5.7.2.8 enum telux::loc::GnssConstellationType [strong]

Defines constellation type of GNSS.

**Enumerator**

**UNKNOWN**  
**GPS** GPS satellite  
**GALILEO** GALILEO satellite  
**SBAS** SBAS satellite  
**COMPASS** COMPASS satellite  
**GLONASS** GLONASS satellite  
**BDS** BDS satellite  
**QZSS** QZSS satellite

**5.7.2.9 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**  
**UNHEALTHY** satellite is not operational and cannot be used in position calculations  
**HEALTHY** satellite is fully operational

**5.7.2.10 enum telux::loc::SVStatus [strong]**

Satellite vehicle processing status.

**Enumerator**

**UNKNOWN**  
**IDLE** SV is not being actively processed  
**SEARCH** The system is searching for this SV  
**TRACK** SV is being tracked

**5.7.2.11 enum telux::loc::SVInfoAvailability [strong]**

Indicates whether Satellite Vehicle info like ephemeris and almanac are present or not

**Enumerator**

**UNKNOWN**  
**YES** Ephemeris or Almanac exists  
**NO** Ephemeris or Almanac doesn't exist

**5.7.2.12 enum telux::loc::SensorType [strong]**

Defines which sensors were used in calculating the position in the position report

**Enumerator**

**UNKNOWN**  
**ACCELEROMETER** Bitmask to specify whether an accelerometer was used  
**GYROSCOPE** Bitmask to specify whether a gyroscope was used

### 5.7.2.13 enum telux::loc::MeasurementType

Specifies which measurements were aided by sensors.

#### Enumerator

**UNKNOWN**

**UNKNOWN**

**UNKNOWN**

**UNKNOWN**

**UNKNOWN**

**UNKNOWN**

**UNKNOWN**

**UNKNOWN**

**UNKNOWN**

**HEADING** Bitmask to specify whether a sensor was used to calculate heading

**SPEED** Bitmask to specify whether a sensor was used to calculate speed

**POSITION** Bitmask to specify whether a sensor was used to calculate position

**VELOCITY** Bitmask to specify whether a sensor was used to calculate velocity

**MEASUREMENT\_COUNT** Bitset

### 5.7.2.14 enum telux::loc::GnssPositionTechType

Specifies which position technology was used.

#### Enumerator

**GNSS\_DEFAULT**

**GNSS\_SATELLITE**

**GNSS\_CELLID**

**GNSS\_WIFI**

**GNSS\_SENSORS**

**GNSS\_REFERENCE\_LOCATION**

**GNSS\_INJECTED\_COARSE\_POSITION**

**GNSS\_AFLT**

**GNSS\_HYBRID**

**GNSS\_PPE**

### 5.7.2.15 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

### 5.7.2.16 enum telux::loc::GnssSystem [strong]

Specifies type of system.

#### Enumerator

**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.

**GNSS\_LOC\_SV\_SYSTEM\_GLONASS** GLONASS satellite.

**GNSS\_LOC\_SV\_SYSTEM\_BDS** BDS satellite.

**GNSS\_LOC\_SV\_SYSTEM\_QZSS** QZSS satellite.

### 5.7.2.17 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**

### 5.7.2.18 enum telux::loc::GlonassTimeValidity

Validity field for GLONASS time.

#### Enumerator

**GNSS\_GLO\_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**

### 5.7.2.19 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

**5.7.2.20 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

**5.7.2.21 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 elipsode semi major  
**HAS\_HOR\_ACCURACY\_ELIP\_SEMI\_MINOR** valid elipsode semi minor  
**HAS\_HOR\_ACCURACY\_ELIP\_AZIMUTH** valid accuracy elipsode 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\_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

### 5.7.2.22 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

### 5.7.2.23 enum telux::loc::GnssDataValidityType

#### Enumerator

**HAS\_JAMMER** Jammer Indicator is available  
**HAS\_AGC** AGC is available

### 5.7.2.24 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

### 5.7.2.25 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.

### 5.7.2.26 enum telux::loc::LocationAggregationType

Specifies the type of engine for the reported fixes

#### Enumerator

- LOC\_OUTPUT\_ENGINE\_FUSED** This is the propagated/aggregated reports from all engines running on the system (e.g.: DR/SPE/PPE) according to QTI algorithm.
- 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/RTK correction engine

### 5.7.2.27 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**

## 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 float telux::loc::DEFAULT\_TUNC\_THRESHOLD = 0.0**

Default value for threshold of time uncertainty. Units: milli-seconds.

**5.7.3.4 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.

## 5.8 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 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.

##### Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status startDataCall](#) (int profileId, IpFamilyType ipFamilyType=[IpFamilyType::IPV4V6](#), [DataCallResponseCb](#) callback=nullptr, [OperationType](#) operationType=[OperationType::DATA\\_LOCAL](#), std::string apn="")=0
- virtual [telux::common::Status stopDataCall](#) (int profileId, IpFamilyType ipFamilyType=[IpFamilyType::IPV4V6](#), [DataCallResponseCb](#) callback=nullptr, [OperationType](#) operationType=[OperationType::DATA\\_LOCAL](#), std::string apn="")=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::startDataCall ( int profileId, IpFamilyType ipFamilyType = IpFamilyType::IPV4V6, DataCallResponseCb callback = nullptr, OperationType operationType = OperationType::DATA_LOCAL, std::string apn = "" ) [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. This is an asynchronous API, client receives notification indicating the data call establishment or failure in callback.

#### 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 <a href="#">IDataProfileManager::requestProfileList</a> 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 <a href="#">telux::data::OperationType</a>
in	<i>apn</i>	Optional access point name

#### 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.

**5.8.2.1.2.4** `virtual telux::common::Status telux::data::IDataConnectionManager::stopDataCall ( int profileId, IpFamilyType ipFamilyType = IpFamilyType::IPV4V6, DataCallResponseCb callback = nullptr, OperationType operationType = OperationType::DATA_LOCAL, std::string apn = "" ) [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).

**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>
in	<i>apn</i>	Optional access point name

**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.

**5.8.2.1.2.5** `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

**5.8.2.1.2.6** `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

**5.8.2.1.2.7** `virtual int telux::data::IDataConnectionManager::getSlotId ( ) [pure virtual]`

Get associated slot id for the Data Connection Manager.

**Returns**

SlotId

**5.8.2.1.2.8** `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 [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](#) `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 TechPreference telux::data::IDataCall::getTechPreference ( ) [pure virtual]**

Get the technology on which the call was brought up.

**Returns**

[TechPreference](#).

**5.8.2.2.2.6 virtual std::list<IpAddrInfo> telux::data::IDataCall::getIpAddressInfo ( ) [pure virtual]**

Get list of IP address information.

**Returns**

List of IP address details.

**5.8.2.2.2.7 virtual IpFamilyType telux::data::IDataCall::getIpFamilyType ( ) [pure virtual]**

Get IP Family Type i.e. IPv4, IPv6 or Both

**Returns**

[IpFamilyType](#).

**5.8.2.2.2.8 virtual int telux::data::IDataCall::getProfileId ( ) [pure virtual]**

Get Profile Id

**Returns**

Profile Identifier.

**5.8.2.2.2.9 virtual OperationType telux::data::IDataCall::getOperationType ( ) [pure virtual]**

Get data operation used for the DataCall.

**Returns**

[OperationType](#)

**5.8.2.2.2.10 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.

**5.8.2.2.2.11 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.

**Parameters**

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 call 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 [~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>status</i>	Data Call Status
in	<i>dataCall</i>	Pointer to <a href="#">IDataCall</a>

### 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="#">DataRestrict↔ ModeType</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="#">DataRestrict↔ ModeType</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)
<a href="#">TechPreference</a>	techPref	Technology preference, default is <a href="#">TechPreference::UNKNOWN</a>
<a href="#">AuthProtocolType</a>	authType	Authentication protocol type, default is <a href="#">AuthProtocolType::AUTH_NONE</a>
<a href="#">IpFamilyType</a>	ipFamilyType	Preferred IP family for the call, default is <a href="#">IpFamilyType::UNKNOWN</a>

### 5.8.2.7 struct telux::data::DataCallStats

Data transfer statistics structure.

#### Data fields

Type	Field	Description
unsigned long	packetsTx	Number of packets transmitted
unsigned long	packetsRx	Number of packets received
long long	bytesTx	Number of bytes transmitted
long long	bytesRx	Number of bytes received
unsigned long	packets↔ DroppedTx	Number of transmit packets dropped
unsigned long	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::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

### 5.8.2.11 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< 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)`

## Static Public Member Functions

- static [DataFactory](#) & [getInstance](#) ()

### 5.8.2.11.1 Member Function Documentation

#### 5.8.2.11.1.1 static [DataFactory](#)& [telux::data::DataFactory::getInstance](#) ( ) [static]

Get Data Factory instance.

#### 5.8.2.11.1.2 [std::shared\\_ptr<IDataConnectionManager>](#) [telux::data::DataFactory::getDataConnectionManager](#) ( int *slotId* = *DEFAULT\_SLOT\_ID* )

Get Data Connection Manager

#### Parameters

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

#### Returns

instance of [IDataConnectionManager](#)

#### 5.8.2.11.1.3 [std::shared\\_ptr<IDataProfileManager>](#) [telux::data::DataFactory::getDataProfileManager](#) ( int *slotId* = *DEFAULT\_SLOT\_ID* )

Get Data Profile Manager

#### Parameters

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

#### Returns

instance of [IDataProfileManager](#)

#### 5.8.2.11.1.4 [std::shared\\_ptr<IDataFilterManager>](#) [telux::data::DataFactory::getDataFilterManager](#) ( int *slotId* = *DEFAULT\_SLOT\_ID* )

Get Data Filter Manager instance

#### Parameters

in	<i>slotId</i>	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.

#### 5.8.2.11.1.5 `std::shared_ptr<telux::data::net::INatManager> telux::data::DataFactory::getNatManager ( telux::data::OperationType oprType )`

Get Network Address Translation(NAT) Manager

**Parameters**

<code>in</code>	<code><i>oprType</i></code>	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.

#### 5.8.2.11.1.6 `std::shared_ptr<telux::data::net::IFirewallManager> telux::data::DataFactory::get<↔> FirewallManager ( telux::data::OperationType oprType )`

Get Firewall Manager

**Parameters**

<code>in</code>	<code><i>oprType</i></code>	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.

#### 5.8.2.11.1.7 `std::shared_ptr<telux::data::net::IFirewallEntry> telux::data::DataFactory::getNew<↔> FirewallEntry ( 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.

**5.8.2.11.1.8 std::shared\_ptr<IIPFilter> telux::data::DataFactory::getNewIpFilter ( 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.

**5.8.2.11.1.9 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.

**5.8.2.12 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](#) ()

**5.8.2.12.1 Constructors and Destructors****5.8.2.12.1.1 virtual telux::data::IDataFilterListener::~~IDataFilterListener ( ) [virtual]**

Destructor of [IDataFilterListener](#)

**5.8.2.12.2 Member Function Documentation****5.8.2.12.2.1 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.

**5.8.2.13 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 ()`

## 5.8.2.13.1 Constructors and Destructors

**5.8.2.13.1.1** virtual `telux::data::IDataFilterManager::~IDataFilterManager ( )` [virtual]

Destructor of [IDataFilterManager](#)

## 5.8.2.13.2 Member Function Documentation

**5.8.2.13.2.1** 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.

**5.8.2.13.2.2** `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.

**5.8.2.13.2.3** `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.

**5.8.2.13.2.4** `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.

**5.8.2.13.2.5** `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..

**Parameters**

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.
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.

**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.

**5.8.2.13.2.6** `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.
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.

```
5.8.2.13.2.7 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.
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.

**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.

**5.8.2.13.2.8** `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.
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.

#### 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.

**5.8.2.13.2.9** `virtual int telux::data::IDataFilterManager::getSlotId ( ) [pure virtual]`

Get associated slot id for the Data Filter Manager.

#### Returns

SlotId

### 5.8.2.14 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](#) ()

#### 5.8.2.14.1 Constructors and Destructors

**5.8.2.14.1.1** `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 )`

#### 5.8.2.14.2 Member Function Documentation

**5.8.2.14.2.1** `int telux::data::DataProfile::getId ( )`

Get profile identifier.

##### Returns

profile id

**5.8.2.14.2.2** `std::string telux::data::DataProfile::getName ( )`

Get profile name.

##### Returns

profile name

**5.8.2.14.2.3** `std::string telux::data::DataProfile::getApn ( )`

Get Access Point Name (APN) name.

##### Returns

APN name



**5.8.2.14.2.4 std::string telux::data::DataProfile::getUserName ( )**

Get profile user name.

**Returns**

user name

**5.8.2.14.2.5 std::string telux::data::DataProfile::getPassword ( )**

Get profile password.

**Returns**

profile password

**5.8.2.14.2.6 TechPreference telux::data::DataProfile::getTechPreference ( )**

Get technology preference.

**Returns**

TechPreference [TechPreference](#)

**5.8.2.14.2.7 AuthProtocolType telux::data::DataProfile::getAuthProtocolType ( )**

Get authentication preference.

**Returns**

AuthProtocolType [AuthProtocolType](#)

**5.8.2.14.2.8 IpFamilyType telux::data::DataProfile::getIpFamilyType ( )**

Get IP Family type.

**Returns**

IpFamilyType [IpFamilyType](#)

**5.8.2.14.2.9 std::string telux::data::DataProfile::toString ( )**

Get the text related informative representation of this object.

**Returns**

String containing informative string.

### 5.8.2.15 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.

#### 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 `onProfileUpdate` (int profileId, [TechPreference](#) techPreference, [ProfileChangeEvent](#) event)
- virtual `~IDataProfileListener` ()

#### 5.8.2.15.1 Constructors and Destructors

5.8.2.15.1.1 virtual `telux::data::IDataProfileListener::~~IDataProfileListener` ( ) [`virtual`]

Destructor of [IDataProfileListener](#)

#### 5.8.2.15.2 Member Function Documentation

5.8.2.15.2.1 virtual void `telux::data::IDataProfileListener::onProfileUpdate` ( int *profileId*, [TechPreference](#) *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.

#### Note

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

### 5.8.2.16 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` ()

### 5.8.2.16.1 Constructors and Destructors

#### 5.8.2.16.1.1 virtual `telux::data::IDataProfileManager::~~IDataProfileManager` ( ) [`virtual`]

Destructor for [IDataProfileManager](#)

### 5.8.2.16.2 Member Function Documentation

#### 5.8.2.16.2.1 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.

#### Note

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

#### 5.8.2.16.2.2 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.

**Note**

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

**5.8.2.16.2.3** `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**

<i>in, out</i>	<i>callback</i>	Callback pointer to get the response.
----------------	-----------------	---------------------------------------

**Returns**

Status of request profile i.e. success or suitable error code.

**5.8.2.16.2.4** `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**

<i>in</i>	<i>profileParams</i>	profileParams configuration to be passed for creating profile either for 3GPP or 3GPP2
<i>in, out</i>	<i>callback</i>	Callback pointer to get the result of create profile

**Returns**

Status of create profile i.e. success or suitable error code.

**5.8.2.16.2.5** `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**

<i>in</i>	<i>profileId</i>	Profile identifier
<i>in</i>	<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.

**5.8.2.16.2.6** `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.

**5.8.2.16.2.7** `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.

**5.8.2.16.2.8** `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 • <a href="#">TechPreference</a>
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.

**5.8.2.16.2.9 virtual int telux::data::IDataProfileManager::getSlotId ( ) [pure virtual]**

Get associated slot id for the Data Profile Manager.

**Returns**

SlotId

**5.8.2.16.2.10 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.

**Parameters**

in	<i>listener</i>	- Listener that processes the notification.
----	-----------------	---

**Returns**

Status.

**Note**

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

**5.8.2.16.2.11 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.

**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::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)

**5.8.2.17.1 Member Function Documentation****5.8.2.17.1.1 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>

**5.8.2.18 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)

**5.8.2.18.1 Member Function Documentation**

**5.8.2.18.1.1** 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>	telux::common::ErrorCode

### 5.8.2.19 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)

#### 5.8.2.19.1 Member Function Documentation

**5.8.2.19.1.1** 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>	telux::common::ErrorCode

### 5.8.2.20 union telux::data::DataCallEndReason.\_\_unnamed\_\_

#### Data fields

Type	Field	Description
MobileIp↵ ReasonCode	IpCode	
Internal↵ ReasonCode	internalCode	
CallManager↵ ReasonCode	cmCode	
SpecReason↵ Code	specCode	



Type	Field	Description
<a href="#">PPPReasonCode</a>	pppCode	
<a href="#">EHRPDReasonCode</a>	ehrpCode	
<a href="#">Ipv6ReasonCode</a>	ipv6Code	
<a href="#">HandoffReasonCode</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

### 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\_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\_HANOVER**  
**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\_LRRRC\_UL\_DATA\_CNF\_FAILURE\_TXN**  
**CE\_LRRRC\_UL\_DATA\_CNF\_FAILURE\_HO**  
**CE\_LRRRC\_UL\_DATA\_CNF\_FAILURE\_CONN\_REL**  
**CE\_LRRRC\_UL\_DATA\_CNF\_FAILURE\_RLF**  
**CE\_LRRRC\_UL\_DATA\_CNF\_FAILURE\_CTRL\_NOT\_CONN**  
**CE\_LRRRC\_CONN\_EST\_FAILURE**  
**CE\_LRRRC\_CONN\_EST\_FAILURE\_ABORTED**  
**CE\_LRRRC\_CONN\_EST\_FAILURE\_ACCESS\_BARRED**  
**CE\_LRRRC\_CONN\_EST\_FAILURE\_CELL\_RESEL**  
**CE\_LRRRC\_CONN\_EST\_FAILURE\_CONFIG\_FAILURE**  
**CE\_LRRRC\_CONN\_EST\_FAILURE\_TIMER\_EXPIRED**  
**CE\_LRRRC\_CONN\_EST\_FAILURE\_LINK\_FAILURE**  
**CE\_LRRRC\_CONN\_EST\_FAILURE\_NOT\_CAMPED**  
**CE\_LRRRC\_CONN\_EST\_FAILURE\_SI\_FAILURE**  
**CE\_LRRRC\_CONN\_EST\_FAILURE\_CONN\_REJECT**

**CE\_LRRR\_CONN\_REL\_NORMAL**  
**CE\_LRRR\_CONN\_REL\_RLF**  
**CE\_LRRR\_CONN\_REL\_CRE\_FAILURE**  
**CE\_LRRR\_CONN\_REL\_OOS\_DURING\_CRE**  
**CE\_LRRR\_CONN\_REL\_ABORTED**  
**CE\_LRRR\_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\_SNDP\_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)

## 5.9 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 `getPhoneNumber ()=0`
- virtual int `getSlotId ()=0`
- virtual std::string `getImsi ()=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.

**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.

**5.9.1.1.2.5 virtual std::string telux::tel::ISubscription::getPhoneNumber ( ) [pure virtual]**

Retrieves the phone number for the SIM subscription.

**Returns**

PhoneNumber.

**5.9.1.1.2.6 virtual int telux::tel::ISubscription::getSlotId ( ) [pure virtual]**

Retrieves SIM Slot index for the SIM pertaining to this subscription object.

**Returns**

SIM slotId.

**5.9.1.1.2.7 virtual std::string telux::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.

**5.9.1.2 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.

**Parameters**

in	<i>count</i>	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 `telx::tel::ISubscriptionManager::~ISubscriptionManager ( )` [virtual]

### 5.9.1.3.2 Member Function Documentation

5.9.1.3.2.1 virtual `bool telx::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> telx::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> telx::tel::ISubscriptionManager::getSubscription ( int slotId = DEFAULT_SLOT_ID, telx::common::Status * status = nullptr )` [pure virtual]

Get Subscription details of the SIM in the given SIM slot.

#### Parameters

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> > telx::tel::ISubscriptionManager::getAllSubscriptions ( telx::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

<i>in</i>	<i>callback</i>	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.

**Parameters**

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**

in	<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 telux::tel::OperatorInfo::getMnc ( )**

Get mnc from operator numeric.

**Returns**

MNC.

**5.10.1.4.2.4 OperatorStatus telux::tel::OperatorInfo::getStatus ( )**

Get status of operator.

**Returns**

status of the operator [OperatorStatus](#).

**5.10.1.5 class telux::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 telux::tel::INetworkSelectionListener::~~INetworkSelectionListener ( ) [virtual]**

Destructor of [INetworkSelectionListener](#)

**5.10.1.5.2 Member Function Documentation****5.10.1.5.2.1 virtual void telux::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**

### 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 telux::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 class telux::tel::IServingSystemManager

Serving System Manager class provides the API to request and set service domain preference and RAT preference.

##### 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::common::Status registerListener (std::weak_ptr< IServingSystemListener > listener)=0`
- virtual `telux::common::Status deregisterListener (std::weak_ptr< IServingSystemListener > listener)=0`
- virtual `~IServingSystemManager ()`

##### 5.11.1.1.1 Constructors and Destructors

5.11.1.1.1 `virtual telux::tel::IServingSystemManager::~~IServingSystemManager ( ) [virtual]`

Destructor of `IServingSystemManager`

##### 5.11.1.1.2 Member Function Documentation

5.11.1.1.2 `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.

**5.11.1.1.2.2** `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.

**5.11.1.1.2.3** `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.

**5.11.1.1.2.4** `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.

**5.11.1.1.2.5** `virtual telux::common::Status telux::tel::IServingSystemManager::setServiceDomain↔`  
`Preference ( 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.

#### 5.11.1.1.2.6 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.

#### 5.11.1.1.2.7 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.

#### 5.11.1.1.2.8 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

**5.11.1.2 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 [~IServingSystemListener](#) ()

**5.11.1.2.1 Constructors and Destructors**

**5.11.1.2.1.1** virtual telux::tel::IServingSystemListener::~~IServingSystemListener ( ) [virtual]

Destructor of [IServingSystemListener](#)

**5.11.1.2.2 Member Function Documentation**

**5.11.1.2.2.1** 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>
----	-------------------	-------------------------------

**5.11.1.2.2.2** 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>
----	-------------------	---

## 5.11.2 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

## 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 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 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.

### 5.12.1.4 struct telux::common::PlatformVersion

Structure contains the version of the platform software

#### Data fields

Type	Field	Description
string	meta	Meta <a href="#">Version</a> , for example: SA2150P_SA515M.LE_LE.1-3_2-1-00297-STD.INT-1
string	modem	Modem <a href="#">Version</a> , for example: MPSS.HI.3.1.c3-00114-SDX55_GENAUTO_TEST-1
string	externalApp	External App <a href="#">Version</a> , for example: LE.UM.3.2.3-72102-SA2150p.Int-1
string	integratedApp	Integrated App MDM <a href="#">Version</a> , for example: LE.UM.4.1.1-71802-sa515m.Int-1

### 5.12.1.5 class telux::common::Version

Provides version of SDK.

#### Static Public Member Functions

- static std::string [getReleaseName](#) ()
- static [SdkVersion](#) [getSdkVersion](#) ()
- static [Status](#) [getPlatformVersion](#) ([PlatformVersion](#) &pv)

#### 5.12.1.5.1 Member Function Documentation



**5.12.1.5.1.1 static std::string telux::common::Version::getReleaseName ( ) [static]**

Get the release name.

**Returns**

String contains release name

**5.12.1.5.1.2 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

**5.12.1.5.1.3 static Status telux::common::Version::getPlatformVersion ( PlatformVersion & pv ) [static]**

Get the platform version. Need obtain required permissions from telux\_allow\_version.

**Parameters**

in		
----	--	--

**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

### 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

**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.

**PIN\_BLOCKED** PIN is blocked. Unblock operation must be issued.

**PIN\_PERM\_BLOCKED** PIN is permanently blocked. The SIM is unusable.

**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

## 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::Cv2xFactory`

`Cv2xFactory` is the factory that creates the Cv2x Radio.

##### Public member functions

- `std::shared_ptr< ICv2xRadioManager > getCv2xRadioManager ()`
- `std::shared_ptr< ICv2xThrottleManager > getCv2xThrottleManager ()`

##### Static Public Member Functions

- static `Cv2xFactory & getInstance ()`

#### 5.13.1.1.1 Member Function Documentation

##### 5.13.1.1.1.1 `static Cv2xFactory& telux::cv2x::Cv2xFactory::getInstance ( ) [static]`

Get `Cv2xFactory` instance

##### Returns

Reference to `Cv2xFactory` singleton.

##### 5.13.1.1.1.2 `std::shared_ptr<ICv2xRadioManager> telux::cv2x::Cv2xFactory::getCv2xRadioManager ( )`

Get `Cv2xRadioManager` instance.

##### Returns

shared pointer to Radio upon success. `nullptr` otherwise.

##### 5.13.1.1.1.3 `std::shared_ptr<ICv2xThrottleManager> telux::cv2x::Cv2xFactory::getCv2xThrottleManager ( )`

Get `Cv2xThrottleManager` instance.

##### Returns

shared pointer to Cv2x ThrottleManager upon success. `nullptr` otherwise.

### 5.13.1.2 class telux::cv2x::ICv2xRadio

This class encapsulates a Cv2xRadio interface.

Returned from getCv2xRadio in Cv2xFactory

#### Public member functions

- virtual Cv2xRadioCapabilities getCapabilities () const =0
- virtual bool isReady () 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 getInterfaceNameFromIpType (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 setGlobalIPInfo](#) (const [IPv6AddrType](#) &ipv6Addr, [common::ResponseCallback](#) cb)=0
- virtual [telux::common::Status setGlobalIPUnicastRoutingInfo](#) (const [GlobalIPUnicastRoutingInfo](#) &destL2Addr, [common::ResponseCallback](#) cb)=0

### 5.13.1.2.1 Constructors and Destructors

5.13.1.2.1.1 virtual [telux::cv2x::ICv2xRadio::~~ICv2xRadio](#) ( ) [[virtual](#)]

Destructor for [ICv2xRadio](#)

### 5.13.1.2.2 Member Function Documentation

5.13.1.2.2.1 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

5.13.1.2.2.2 virtual bool [telux::cv2x::ICv2xRadio::isReady](#) ( ) const [[pure virtual](#)]

Returns true if the radio interface has completed initialization.

#### Returns

True if ready. False otherwise.

5.13.1.2.2.3 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.

**5.13.1.2.2.4** `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.
----	-----------------	---

**5.13.1.2.2.5** `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.
----	-----------------	---

**5.13.1.2.2.6** `virtual telux::common::Status telux::cv2x::ICv2xRadio::createRxSubscription ( Traffic< IpType 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()`.

**5.13.1.2.2.7 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.

**5.13.1.2.2.8 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.

**5.13.1.2.2.9** `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.

**Note**

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

**5.13.1.2.2.10** `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.

**5.13.1.2.2.11** `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.

**5.13.1.2.2.12** `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.

**5.13.1.2.2.13** `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.

**5.13.1.2.2.14** `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.

**5.13.1.2.2.15** `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.

**Note**

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

#### 5.13.1.2.2.16 **virtual telux::common::Status telux::cv2x::ICv2xRadio::requestDataSessionSettings ( RequestDataSessionSettingsCallback *cb* ) [pure virtual]**

Request data session settings currently in use.

**Parameters**

<i>in</i>	<i>cb</i>	- Callback that will be invoked and returns the data session settings. Must not be null.
-----------	-----------	--

**Returns**

SUCCESS if no error occurred.

**Note**

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

#### 5.13.1.2.2.17 **virtual telux::common::Status telux::cv2x::ICv2xRadio::updateSrcL2Info ( UpdateSrcL2InfoCallback *cb* ) [pure virtual]**

Requests modem to change L2 info.

**Parameters**

<i>in</i>	<i>cb</i>	- Callback that will be invoked and returns status. Must not be null.
-----------	-----------	---

**Returns**

SUCCESS if no error occurred.

#### 5.13.1.2.2.18 **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**

<i>in</i>	<i>infoList</i>	- Trusted and malicious UE information list
<i>in</i>	<i>cb</i>	- Callback that will be invoked and returns status. Must not be null.

**Returns**

SUCCESS if no error occurred.

**Note**

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

**5.13.1.2.2.19 virtual std::string telux::cv2x::ICv2xRadio::getInterfaceNameFromIpType ( TrafficIpType ipType ) [pure virtual]**

Get interface name based on ipType.

**Parameters**

<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.

**5.13.1.2.2.20 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.

**5.13.1.2.2.21** `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.

**5.13.1.2.2.22** `virtual telux::common::Status telux::cv2x::ICv2xRadio::setGlobalIPInfo ( const IPv6AddrType & ipv6Addr, common::ResponseCallback cb ) [pure virtual]`

Set CV2X global IP address for the IP interface.

Use case and Precondition: OBU: Registers a TX/RX *NON IP* flow for receiving the signed WSA/WRA for IP session initiation; Once receives the IP prefix in the WDS/WRA from RSU, call this method.

RSU: Specifies its own global prefix via this method, and creates/composes WSA/WRA advertising the IP configs.

**Parameters**

in	<i>ipv6Addr</i>	- CV2X global IP address prefix.
in	<i>cb</i>	- Callback that is invoked when set the global IP address complete. This may be null.

**Returns**

SUCCESS if no error occurred.

**5.13.1.2.2.23** `virtual telux::common::Status telux::cv2x::ICv2xRadio::setGlobalIPUnicastRoutingInfo ( const GlobalIPUnicastRoutingInfo & destL2Addr, common::ResponseCallback cb ) [pure virtual]`

Set CV2X IP interface global IP unicast routing information.

Use case and Precondition: OBU: Registers a TX/RX *NON IP* flow for receiving the signed WSA/WRA for IP session initiation; Once receives the IP prefix in the WSA/WRA from RSU, call the [setGlobalIPInfo](#) method to update the ip interface with global IP; Now call this method to set the routing information with dest L2 addr negotiated in WSA/WRA.

RSU: Specifies its own global prefix via [setGlobalIPInfo](#), and creates/composes WSA/WRA advertising the IP configs; Now set routing information of its own via this method.

#### Parameters

in	<i>destL2Addr</i>	- CV2X destination L2 address for unicast routing purpose.
in	<i>cb</i>	- Callback that is invoked when set global IP unicast routing information complete. This may be null.

#### Returns

SUCCESS if no error occurred.

### 5.13.1.3 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](#) ()

#### 5.13.1.3.1 Constructors and Destructors

**5.13.1.3.1.1** `virtual telux::cv2x::ICv2xRadioListener::~~ICv2xRadioListener ( ) [virtual]`

Destructor for [ICv2xRadioListener](#)

### 5.13.1.3.2 Member Function Documentation

#### 5.13.1.3.2.1 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

#### 5.13.1.3.2.2 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.
----	---------------	----------------------

##### Note

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

**Deprecated** use onStatusChanged in Cv2xListener

#### 5.13.1.3.2.3 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.
----	---------------------	-----------------------

#### 5.13.1.3.2.4 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

**5.13.1.3.2.5** `virtual void telux::cv2x::ICv2xRadioListener::onSpsSchedulingChanged ( const SpsSchedulingInfo & schedulingInfo ) [virtual]`

Called when SPS scheduling has changed.

#### Parameters

in	<i>schedulingInfo</i>	- SPS scheduling information .
----	-----------------------	--------------------------------

#### Note

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

**5.13.1.3.2.6** `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 .
----	---------------------	------------------------------------

#### 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 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` ()

#### 5.13.1.4.1 Constructors and Destructors

5.13.1.4.1.1 `virtual telux::cv2x::ICv2xRadioManager::~~ICv2xRadioManager ( ) [virtual]`

#### 5.13.1.4.2 Member Function Documentation

5.13.1.4.2.1 `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.

5.13.1.4.2.2 `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.

5.13.1.4.2.3 `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.

5.13.1.4.2.4 `virtual telux::common::Status telux::cv2x::ICv2xRadioManager::startCv2x ( StartCv2x↔ Callback cb ) [pure virtual]`

Put modem into CV2X mode.

**Parameters**

in	<i>cb</i>	- Callback that is invoked when Cv2x mode is started
----	-----------	--

**Returns**

SUCCESS on success. Error status otherwise.

#### 5.13.1.4.2.5 virtual telux::common::Status telux::cv2x::ICv2xRadioManager::stopCv2x ( StopCv2x↔ Callback *cb* ) [pure virtual]

Take modem out of CV2X mode

**Parameters**

in	<i>cb</i>	- Callback that is invoked when Cv2x mode is stopped
----	-----------	--

**Returns**

SUCCESS on success. Error status otherwise.

#### 5.13.1.4.2.6 virtual telux::common::Status telux::cv2x::ICv2xRadioManager::requestCv2xStatus ( RequestCv2xStatusCallback *cb* ) [pure virtual]

request CV2X status from modem

**Parameters**

in	<i>cb</i>	- Callback that is invoked when Cv2x status is retrieved
----	-----------	--

**Returns**

SUCCESS on success. Error status otherwise.

**Deprecated** use requestCv2xStatus(RequestCv2xCalbackEx)

#### 5.13.1.4.2.7 virtual telux::common::Status telux::cv2x::ICv2xRadioManager::requestCv2xStatus ( RequestCv2xStatusCallbackEx *cb* ) [pure virtual]

request CV2X status from modem

**Parameters**

in	<i>cb</i>	- Callback that is invoked when Cv2x status is retrieved
----	-----------	--

**Returns**

SUCCESS on success. Error status otherwise.

#### 5.13.1.4.2.8 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.
----	-----------------	--

##### Note

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

#### 5.13.1.4.2.9 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.
----	-----------------	--

##### Note

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

#### 5.13.1.4.2.10 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.

### 5.13.1.5 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.

Type	Field	Description
uint16_t	localPort	Local port number of the socket used for binding.

### 5.13.1.6 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

### 5.13.1.7 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

### 5.13.1.8 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



Type	Field	Description
bool	time↔ Uncertainty↔ Valid	Time uncertainty value is valid
float	time↔ Uncertainty	Time uncertainty value in milliseconds

### 5.13.1.9 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.

### 5.13.1.10 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.
bool	isUnicast	Set to true if isUnicast flow. If false, Non-Unicast flow will be created. Note: Unicast flows ignore subscribed Service Ids

### 5.13.1.11 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
Priority	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
Periodicity	periodicity	
uint64_t	periodicityMs	This is the new interface to specify periodicity in milliseconds for SpsFlowInfo. 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.

### 5.13.1.12 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.
Radio↔ Concurrency↔ Mode	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.
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.
uint8_t	isUnicast↔ Supported	Non zero value if unicast is supported.

**5.13.1.13 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.

**5.13.1.14 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.

**5.13.1.15 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.

### 5.13.1.16 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.
bool	trustedUEs↔ Valid	Trusted remote UE sources are valid.
vector< <a href="#">TrustedUE↔ Info</a> >	trustedUEs	Trusted remote UE sources.

### 5.13.1.17 struct telux::cv2x::IPv6Address

Contains IPv6 address.

Used in [DataSessionSettings](#)

#### Data fields

Type	Field	Description
uint8_t	addr[16]	

### 5.13.1.18 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.

### 5.13.1.19 struct telux::cv2x::IPv6AddrType

Encapsulates ipv6 prefix length in bits and ipv6 prefix.

Used in setGlobalIPInfo.

**Data fields**

Type	Field	Description
uint8_t	prefixLen	< ipv6 address prefix length in bits, range [1, 128]
uint8_t	ipv6Addr[C↔ V2X_IPV6_↔ ADDR_ARR↔ AY_LEN]	

**5.13.1.20 struct telux::cv2x::GlobalIPUnicastRoutingInfo**

Encapsulates destination L2 address.

Used in setGlobalIPUnicastRoutingInfo.

**Data fields**

Type	Field	Description
uint8_t	destMac↔ Addr[CV2X_↔ MAC_ADD↔ R_LEN]	< Array that stores CV2X L2 MAC address at the last 3 bytes in big endian order

**5.13.1.21 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](#) ()

**5.13.1.21.1 Constructors and Destructors**

**5.13.1.21.1.1** virtual telux::cv2x::ICv2xRxSubscription::~~ICv2xRxSubscription ( ) [virtual]

### 5.13.1.21.2 Member Function Documentation

**5.13.1.21.2.1** `virtual uint32_t telux::cv2x::ICv2xRxSubscription::getSubscriptionId ( ) const [pure virtual]`

Accessor for Rx subscription ID

#### Returns

subscription ID

**5.13.1.21.2.2** `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)

**5.13.1.21.2.3** `virtual int telux::cv2x::ICv2xRxSubscription::getSock ( ) const [pure virtual]`

Accessor for the socket file descriptor

#### Returns

The Rx subscriptions's socket fd.

**5.13.1.21.2.4** `virtual struct sockaddr_in6 telux::cv2x::ICv2xRxSubscription::getSockAddr ( ) const [pure virtual]`

Accessor for the socket address description

#### Returns

The Rx subscriptions's socket address

**5.13.1.21.2.5** `virtual uint16_t telux::cv2x::ICv2xRxSubscription::getPortNum ( ) const [pure virtual]`

Accessor for the subscriptions's port number

#### Returns

The Rx subscriptions's port num

**5.13.1.21.2.6** `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

**5.13.1.21.2.7** `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
----	---------------	---------------------------------------

### 5.13.1.22 class telux::cv2x::ICv2xTxFlow

This is 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](#) ()

#### 5.13.1.22.1 Constructors and Destructors

**5.13.1.22.1.1** `virtual telux::cv2x::ICv2xTxFlow::~~ICv2xTxFlow ( ) [virtual]`

#### 5.13.1.22.2 Member Function Documentation



**5.13.1.22.2.1** `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

**5.13.1.22.2.2** `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)

**5.13.1.22.2.3** `virtual uint32_t telux::cv2x::ICv2xTxFlow::getServiceId ( ) const [pure virtual]`

Accessor for service ID

**Returns**

The flow's Service ID.

**5.13.1.22.2.4** `virtual int telux::cv2x::ICv2xTxFlow::getSock ( ) const [pure virtual]`

Accessor for the socket file descriptor

**Returns**

The flow's socket fd.

**5.13.1.22.2.5** `virtual struct sockaddr_in6 telux::cv2x::ICv2xTxFlow::getSockAddr ( ) const [pure virtual]`

Accessor for the socket address description

**Returns**

The flow's socket address

**5.13.1.22.2.6** `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

### 5.13.1.23 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](#) ()

#### 5.13.1.23.1 Constructors and Destructors

5.13.1.23.1.1 virtual telux::cv2x::ICv2xTxRxSocket::~ICv2xTxRxSocket ( ) [virtual]

#### 5.13.1.23.2 Member Function Documentation

5.13.1.23.2.1 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

5.13.1.23.2.2 virtual uint32\_t telux::cv2x::ICv2xTxRxSocket::getServiceId ( ) const [pure virtual]

Accessor for service ID

#### Returns

The Service ID bound to the socket.

5.13.1.23.2.3 virtual int telux::cv2x::ICv2xTxRxSocket::getSocket ( ) const [pure virtual]

Accessor for the socket file descriptor

#### Returns

The socket fd.

**5.13.1.23.2.4** `virtual struct sockaddr_in6 telux::cv2x::ICv2xTxRxSocket::getSocketAddr ( ) const`  
`[pure virtual]`

Accessor for the socket address description

#### Returns

The socket address

**5.13.1.23.2.5** `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*** Timing is invalid

**CONFIG** Config is invalid  
**UE\_MODE** UE Mode is invalid  
**GEOPOLYGON** Left current geopolygon  
**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**

## 5.14 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.

#### Note

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

#### 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.

#### Note

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

#### 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.

#### 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 [onDtmfToneDetection](#) ([DtmfTone](#) dtmfTone)
- virtual [~IVoiceListener](#) ()

#### 5.14.1.11.1 Constructors and Destructors

##### 5.14.1.11.1.1 `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
----	-----------------	----------------------

#### 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 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

#### Note

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

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

#### Note

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

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'ed when the [IStreamBuffer](#) is destroyed.

#### Returns

raw buffer

#### Note

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

**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

**Note**

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

**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
-------------	----------------------------------

**Note**

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

**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

**Note**

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

**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`

### 5.14.1.17.1 Member Function Documentation

5.14.1.17.1.1 `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.

#### Note

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

**5.14.1.17.1.2** `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.

#### Note

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

**5.14.1.17.1.3** `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

<i>in</i>	<i>callback</i>	callback pointer to get the response of getDevices.
-----------	-----------------	---

#### 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.

**5.14.1.17.1.4** `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

<i>in</i>	<i>callback</i>	callback pointer to get the response of getStreamTypes.
-----------	-----------------	---

#### 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.

#### 5.14.1.17.1.5 `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.

##### Note

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

#### 5.14.1.17.1.6 `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.

#### 5.14.1.17.1.7 `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.

**Note**

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

**5.14.1.17.1.8 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.

**5.14.1.17.1.9 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`

**5.14.1.18.1 Member Function Documentation****5.14.1.18.1.1 virtual DeviceType telux::audio::IAudioDevice::getType ( ) [pure virtual]**

Get the type of Device (i.e SPEAKER, MIC etc)

**Returns**

DeviceType

**Note**

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

**5.14.1.18.1.2 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

**Note**

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

**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

**5.14.1.19.1 Member Function Documentation****5.14.1.19.1.1 virtual `StreamType` `telux::audio::IAudioStream::getType` ( ) [pure virtual]**

Get the stream type like VOICE, PLAY, CAPTURE

**Returns**

`StreamType`

**Note**

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

**5.14.1.19.1.2 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.

**Note**

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

**5.14.1.19.1.3 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.

**Note**

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

**5.14.1.19.1.4 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.

**Note**

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

**5.14.1.19.1.5 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.

**Note**

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

#### 5.14.1.19.1.6 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.

**Note**

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

#### 5.14.1.19.1.7 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.

**Note**

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

**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

**5.14.1.20.1 Member Function Documentation****5.14.1.20.1.1 virtual telux::common::Status telux::audio::IAudioVoiceStream::startAudio ( telux::common::ResponseCallback *callback* = nullptr ) [pure virtual]**

Starts audio stream

**Parameters**

in	<i>callback</i>	callback to get the response of startAudio.
----	-----------------	---

**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.

**5.14.1.20.1.2 virtual telux::common::Status telux::audio::IAudioVoiceStream::stopAudio ( telux::common::ResponseCallback *callback* = nullptr ) [pure virtual]**

Stops audio stream

**Parameters**

in	<i>callback</i>	callback to get the response of stopAudio.
----	-----------------	--

**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.

**5.14.1.20.1.3** `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.

**Note**

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

**5.14.1.20.1.4** `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.

**Note**

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

**5.14.1.20.1.5** `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.

**Note**

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

**5.14.1.20.1.6** `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

**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 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`

#### 5.14.1.21.1 Member Function Documentation

##### 5.14.1.21.1.1 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

#### Note

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

##### 5.14.1.21.1.2 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.

**Note**

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

#### 5.14.1.21.1.3 **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.

#### 5.14.1.21.1.4 **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.

#### 5.14.1.21.1.5 **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`

**5.14.1.22.1 Member Function Documentation**

**5.14.1.22.1.1 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

**Note**

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

**5.14.1.22.1.2 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.

<i>in</i>	<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.

**5.14.1.23 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

**5.14.1.23.1 Member Function Documentation****5.14.1.23.1.1 virtual telux::common::Status telux::audio::IAudioLoopbackStream::startLoopback ( telux::common::ResponseCallback *callback* = nullptr ) [pure virtual]**

Start loopback between source and sink devices

**Parameters**

<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.

**Note**

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

**5.14.1.23.1.2 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.

**Note**

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

**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

**5.14.1.24.1 Member Function Documentation**

**5.14.1.24.1.1 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] duration Duration (in milliseconds) for which the tone needs to be played. The constant infiniteToneDuration(=0xFFFF) represents infinite duration. [in] gain Tone Gain. [in] callback callback to get the response of play tone.
-----------	-------------	---

**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.

#### 5.14.1.24.1.2 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.

##### 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 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 <code>tearDown</code> .
----	-----------------	---

##### 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.

##### Parameters

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*

### 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\_AFTER\_PLAY* Stop Playing Immediately and clear buffer pipeline

## 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.

##### Note

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

##### 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.

##### Note

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

### 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.

#### Note

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

### 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.

#### 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 `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.

#### Note

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

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.

#### Note

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

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.

**Note**

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

#### 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.

**Note**

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

#### 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.

**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 [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](#)

#### Note

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

##### 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

#### Note

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

##### 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

#### Note

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



### 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.

#### 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 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

#### Note

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

**5.15.1.5.2.2 virtual std::string telux::therm::IThermalZone::getDescription ( ) [pure virtual]**

Retrieves the type of sensor.

**Returns**

Sensor type

**Note**

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

**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

**Note**

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

**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

**Note**

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

**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

**Note**

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

#### 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.

**Note**

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

#### 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.

**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 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

**Note**

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

**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

**Note**

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

**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

**Note**

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

**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

**Note**

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

**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.

**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 [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

**Note**

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

**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

**Note**

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

#### 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 within the *imminentDuration* time. If disabled successfully within *imminentDuration* time, the system timer for auto-enablement will be reset.

**Parameters**

in	<i>imminentDuration</i>	Time elapsed(in seconds) for the shutdown mode to be enabled
----	-------------------------	--

**Note**

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

#### 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.

**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< [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.

#### Note

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

**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.

#### Note

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

**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

in	<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.

**Note**

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

#### 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

**Note**

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

#### 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.

**Note**

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



**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.

#### Note

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

## 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.

##### Note

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

##### Public member functions

- `std::shared_ptr< ITcuActivityManager > getTcuActivityManager ()`
- `~PowerFactory ()`

##### Static Public Member Functions

- `static PowerFactory & getInstance ()`

#### 5.16.1.1.1 Constructors and Destructors

5.16.1.1.1.1 `telux::power::PowerFactory::~~PowerFactory ( )`

#### 5.16.1.1.2 Member Function Documentation

5.16.1.1.2.1 `static PowerFactory& telux::power::PowerFactory::getInstance ( ) [static]`

API to get the factory instance for TCU-activity management

5.16.1.1.2.2 `std::shared_ptr<ITcuActivityManager> telux::power::PowerFactory::getTcuActivity↔  
Manager ( )`

API to get the TCU-activity Manager instance

##### Returns

Pointer of `ITcuActivityManager` object.

##### Note

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

### 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.

#### 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 [onTcuActivityStateUpdate](#) ([TcuActivityState](#) state)
- 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](#) ( [TcuActivityState](#) *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
----	--------------	--

#### Note

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

### 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.

#### 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< 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.

**Note**

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

**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.

#### Note

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

**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 <a href="#">ITcuActivityListener</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.

**5.16.1.3.2.4** `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

**Note**

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

**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.

**Note**

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

**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

**Note**

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

**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.

#### 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.

#### Note

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

**5.16.1.3.2.8 virtual TcuActivityState telux::power::ITcuActivityManager::getActivityState ( ) [pure virtual]**

Get the current TCU-activity state.

#### Returns

TcuActivityState

#### Note

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

**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.

## Note

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

## 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

## 5.17 Remote SIM Provisioning

This section contains APIs related to Remote SIM provisioning.

### 5.17.1 Data Structure Documentation

#### 5.17.1.1 struct telux::rsp::CustomHeader

Header information to be sent along with HTTP post request.

##### Data fields

Type	Field	Description
string	name	Header name
string	value	Header value

#### 5.17.1.2 class telux::rsp::IHttpTransactionListener

The interface listens for indication to perform HTTP request and send back the response for HTTP request to modem.

The methods in the listener can be invoked from multiple threads. It is client's responsibility to make sure 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 [onNewHttpRequest](#) (int slotId, const std::string &url, int tokenId, const std::vector< [CustomHeader](#) > &headers, const std::string &reqPayload)
- virtual [~IHttpTransactionListener](#) ()

#### 5.17.1.2.1 Constructors and Destructors

5.17.1.2.1.1 virtual telux::rsp::IHttpTransactionListener::~IHttpTransactionListener ( ) [virtual]

Destructor of [IHttpTransactionListener](#)

#### 5.17.1.2.2 Member Function Documentation

5.17.1.2.2.1 virtual void telux::rsp::IHttpTransactionListener::onNewHttpRequest ( int *slotId*, const std::string & *url*, int *tokenId*, const std::vector< [CustomHeader](#) > & *headers*, const std::string & *reqPayload* ) [virtual]

An application handling this indication should perform the HTTP request and call the [IHttpTransactionManager::sendHttpTransactionReq](#) to provide the result of the HTTP transaction.

**Parameters**

in	<i>slotId</i>	Slot identifier corresponding to the card.
in	<i>url</i>	URL to sent HTTP post request.
in	<i>tokenId</i>	Token identifier.
in	<i>headers</i>	Header information to be sent along with HTTP post request.
in	<i>reqPayload</i>	Request payload.

**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 class telux::rsp::IHttpTransactionManager**

[IHttpTransactionManager](#) is the interface to service HTTP related requests from the modem, for Sim profile update related operations.

**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 [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status](#) [sendHttpTransactionResult](#) (uint32\_t token, [HttpResult](#) result, const std::vector< [CustomHeader](#) > &headers, const std::vector< uint8\_t > &response, [common::ResponseCallback](#) callback=nullptr, int slotId=DEFAULT\_SLOT\_ID)=0
- virtual [telux::common::Status](#) [registerListener](#) (std::weak\_ptr< [IHttpTransactionListener](#) > listener)=0
- virtual [telux::common::Status](#) [deregisterListener](#) (std::weak\_ptr< [IHttpTransactionListener](#) > listener)=0
- virtual [~IHttpTransactionManager](#) ()

**5.17.1.3.1 Constructors and Destructors**

**5.17.1.3.1.1** virtual [telux::rsp::IHttpTransactionManager::~IHttpTransactionManager](#) ( ) [[virtual](#)]

Destructor for [IHttpTransactionManager](#)

**5.17.1.3.2 Member Function Documentation**

**5.17.1.3.2.1** virtual bool [telux::rsp::IHttpTransactionManager::isSubsystemReady](#) ( ) [[pure virtual](#)]

Checks if the eUICC subsystem is ready.

**Returns**

True if EuiccManager 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::rsp::IHttpTransactionManager::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::rsp::IHttpTransactionManager::sendHttpRequest(` `Result ( uint32_t token, HttpRequest result, const std::vector< CustomHeader > &` `headers, const std::vector< uint8_t > & response, common::ResponseCallback callback` `= nullptr, int slotId = DEFAULT_SLOT_ID ) [pure virtual]`

Send the result of HTTP Post request transaction to modem.

**Parameters**

in	<i>token</i>	Token identifier for request and response pair.
in	<i>result</i>	HTTP transaction request result.
in	<i>headers</i>	Custom Headers in HTTP Response.
in	<i>response</i>	HTTP response payload.
in	<i>callback</i>	Callback function to get the result of send HTTP transaction request.
in	<i>slotId</i>	Slot identifier corresponding to the card.

**Returns**

Status of send HTTP transaction 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.

#### 5.17.1.3.2.4 virtual telux::common::Status telux::rsp::IHttpTransactionManager::registerListener ( std::weak\_ptr< IHttpTransactionListener > *listener* ) [pure virtual]

Register a listener for specific events like perform HTTP Post request.

##### Parameters

in	<i>listener</i>	Pointer of <a href="#">IHttpTransactionListener</a> object that processes the notification.
----	-----------------	---

##### Returns

Status of registerHttpListener 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.

#### 5.17.1.3.2.5 virtual telux::common::Status telux::rsp::IHttpTransactionManager::deregisterListener ( std::weak\_ptr< IHttpTransactionListener > *listener* ) [pure virtual]

De-register the listener.

##### Parameters

in	<i>listener</i>	Pointer of <a href="#">IHttpTransactionListener</a> object that needs to be removed.
----	-----------------	--

##### Returns

Status of deregisterHttpListener 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.

### 5.17.1.4 class telux::rsp::SimProfile

[SimProfile](#) class represents single eUICC profile on the card.

##### Note

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

## Public member functions

- **SimProfile** (int profileId, 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** ()
- 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** ()

### 5.17.1.4.1 Constructors and Destructors

**5.17.1.4.1.1** **telux::rsp::SimProfile::SimProfile** ( int *profileId*, 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* )

### 5.17.1.4.2 Member Function Documentation

**5.17.1.4.2.1** int **telux::rsp::SimProfile::getSlotId** ( )

Get slot id associated for this profile

#### Returns

SlotId

#### Note

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

**5.17.1.4.2.2 int telux::rsp::SimProfile::getProfileId ( )**

Get profile identifier.

**Returns**

unique identifier for the profile

**Note**

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

**5.17.1.4.2.3 const std::string& telux::rsp::SimProfile::getIccid ( )**

Get profile ICCID.

**Returns**

profile ICCID coded as in EF-ICCID

**Note**

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

**5.17.1.4.2.4 bool telux::rsp::SimProfile::isActive ( )**

Indicates the profile state whether active or not.

**Returns**

true if profile is Active

**Note**

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

**5.17.1.4.2.5 const std::string& telux::rsp::SimProfile::getNickName ( )**

Get profile nick name.

**Returns**

profile nick name

**Note**

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

**5.17.1.4.2.6 const std::string& telux::rsp::SimProfile::getSPN ( )**

Get profile service provider name.

**Returns**

profile service provider name.

**Note**

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

**5.17.1.4.2.7 const std::string& telux::rsp::SimProfile::getName ( )**

Get profile name.

**Returns**

profile name

**Note**

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

**5.17.1.4.2.8 IconType telux::rsp::SimProfile::getIconType ( )**

Get profile icon type.

**Returns**

profile icon type

**Note**

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



**5.17.1.4.2.9** `std::vector<uint8_t> telux::rsp::SimProfile::getIcon ( )`

Get profile icon content.

**Returns**

profile icon content

**Note**

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

**5.17.1.4.2.10** `ProfileClass telux::rsp::SimProfile::getClass ( )`

Get profile class.

**Returns**

profile class

**Note**

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

**5.17.1.4.2.11** `PolicyRuleMask telux::rsp::SimProfile::getPolicyRule ( )`

Get profile policy rules.

**Returns**

mask of profile policy rules

**Note**

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

**5.17.1.4.2.12** `std::string telux::rsp::SimProfile::toString ( )`

Get the text related informative representation of this object.

**Returns**

String containing informative string.

### 5.17.1.5 class telux::rsp::SimProfileFactory

[SimProfileFactory](#) is the central factory to create all eUICC manager class instances.

#### Public member functions

- `std::shared_ptr< ISimProfileManager > getSimProfileManager ()`
- `std::shared_ptr< IHttpTransactionManager > getHttpTransactionManager ()`

#### Static Public Member Functions

- static `SimProfileFactory & getInstance ()`

#### 5.17.1.5.1 Member Function Documentation

##### 5.17.1.5.1.1 static SimProfileFactory& telux::rsp::SimProfileFactory::getInstance ( ) [static]

Get SIM Profile Factory instance.

##### 5.17.1.5.1.2 std::shared\_ptr<ISimProfileManager> telux::rsp::SimProfileFactory::getSimProfileManager ( )

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

#### Returns

instance of [ISimProfileManager](#)

#### Note

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

##### 5.17.1.5.1.3 std::shared\_ptr<IHttpTransactionManager> telux::rsp::SimProfileFactory::getHttpTransactionManager ( )

Get HttpTransactionManager. HttpTransactionManager is a primary interface for sending the response for HTTP post request to modem and listen for indication for HTTP post request to download the profile.

#### Returns

instance of [IHttpTransactionManager](#)

#### Note

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

### 5.17.1.6 class telux::rsp::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.

#### 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 [onAddProfileUpdate](#) (int slotId, bool userConsentRequired, [DownloadStatus](#) status, uint8\_t percentage, [DownloadErrorCause](#) cause, [PolicyRuleMask](#) mask)
- virtual [~ISimProfileListener](#) ()

#### 5.17.1.6.1 Constructors and Destructors

5.17.1.6.1.1 virtual telux::rsp::ISimProfileListener::~~ISimProfileListener ( ) [virtual]

Destructor of [ISimProfileListener](#)

#### 5.17.1.6.2 Member Function Documentation

5.17.1.6.2.1 virtual void telux::rsp::ISimProfileListener::onAddProfileUpdate ( int *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>userConsentRequired</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)

#### Note

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

### 5.17.1.7 class telux::rsp::ISimProfileManager

[ISimProfileManager](#) is a primary interface for remote eUICCs (eSIMs or embedded SIMs) provisioning. This interface provides APIs to add, delete, set profile on the eUICC.

#### 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 [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status addProfile](#) (const std::string &activationCode, [common::ResponseCallback](#) callback=nullptr, const std::string &confirmationCode="", bool userConsentSupported=false, int slotId=DEFAULT\_SLOT\_ID)=0
- virtual [telux::common::Status deleteProfile](#) (int profileId, [common::ResponseCallback](#) callback=nullptr, int slotId=DEFAULT\_SLOT\_ID)=0
- virtual [telux::common::Status setProfile](#) (int profileId, bool enable, [common::ResponseCallback](#) callback=nullptr, int slotId=DEFAULT\_SLOT\_ID)=0
- virtual [telux::common::Status updateNickName](#) (int profileId, const std::string &nickName, [common::ResponseCallback](#) callback=nullptr, int slotId=DEFAULT\_SLOT\_ID)=0
- virtual [telux::common::Status requestProfileList](#) ([ProfileListResponseCb](#)=nullptr, int slotId=DEFAULT\_SLOT\_ID)=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](#) ()

#### 5.17.1.7.1 Constructors and Destructors

5.17.1.7.1.1 virtual [telux::rsp::ISimProfileManager::~ISimProfileManager](#) ( ) [[virtual](#)]

Destructor for [ISimProfileManager](#)

#### 5.17.1.7.2 Member Function Documentation

5.17.1.7.2.1 virtual bool [telux::rsp::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.

### 5.17.1.7.2.2 `virtual std::future<bool> telux::rsp::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.

### 5.17.1.7.2.3 `virtual telux::common::Status telux::rsp::ISimProfileManager::addProfile ( const std::string & activationCode, common::ResponseCallback callback = nullptr, const std::string & confirmationCode = "", bool userConsentSupported = false, int slotId = DEFAULT_SLOT_ID ) [pure virtual]`

Add new profile to eUICC card and download and install the profile on eUICC.

**Parameters**

in	<i>activationCode</i>	Activation code.
in	<i>callback</i>	Callback function to get the result of add profile.
in	<i>confirmationCode</i>	Optional confirmation code required for downloading the profile.
in	<i>userConsentSupported</i>	Optional User consent supported or not.
in	<i>slotId</i>	Slot identifier corresponding to the card.

**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.

**5.17.1.7.2.4** `virtual telux::common::Status telux::rsp::ISimProfileManager::deleteProfile ( int profileId, common::ResponseCallback callback = nullptr, int slotId = DEFAULT_SLOT_ID ) [pure virtual]`

Delete profile from eUICC card.

#### Parameters

in	<i>profileId</i>	Profile identifier
in	<i>callback</i>	Callback function to get the result of delete profile.
in	<i>slotId</i>	Slot identifier corresponding to the card.

#### 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.

**5.17.1.7.2.5** `virtual telux::common::Status telux::rsp::ISimProfileManager::setProfile ( int profileId, bool enable, common::ResponseCallback callback = nullptr, int slotId = DEFAULT_SLOT_ID ) [pure virtual]`

Enable or disable profile which allows to switch to other profile on eUICC card.

#### Parameters

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.
in	<i>slotId</i>	Slot identifier corresponding to the card.

#### 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.

**5.17.1.7.2.6** `virtual telux::common::Status telux::rsp::ISimProfileManager::updateNickName ( int profileId, const std::string & nickName, common::ResponseCallback callback = nullptr, int slotId = DEFAULT_SLOT_ID ) [pure virtual]`

Update nick name of the profile

**Parameters**

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.
in	<i>slotId</i>	Slot identifier corresponding to the card.

**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.

**5.17.1.7.2.7** `virtual telux::common::Status telux::rsp::ISimProfileManager::requestProfileList ( ProfileListResponseCb = nullptr, int slotId = DEFAULT_SLOT_ID ) [pure virtual]`

Request list of profiles supported by the eUICC card.

**Parameters**

in	<i>callback</i>	Callback function to get the result of request profile list.
in	<i>slotId</i>	Slot identifier corresponding to the card.

**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.

**5.17.1.7.2.8** `virtual telux::common::Status telux::rsp::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.

### 5.17.1.7.2.9 virtual telux::common::Status telux::rsp::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::rsp::HttpResult [strong]

Defines the HTTP request result.

**Enumerator**

**TRANSACTION\_SUCCESSFUL** HTTP request successful

**UNKNOWN\_ERROR** Unknown error

**HTTP\_SERVER\_ERROR** Server error

**HTTP\_TLS\_ERROR** TLS error

**HTTP\_NETWORK\_ERROR** Network error

### 5.17.2.2 enum telux::rsp::ProfileType [strong]

Indicates profile type of card

**Enumerator**

**UNKNOWN**

**REGULAR** Regular profile

**EMERGENCY** Emergency profile



### 5.17.2.3 enum telux::rsp::IconType [strong]

Indicates profile icon type.

#### Enumerator

**NONE** No icon information  
**JPEG** JPEG icon  
**PNG** PNG icon

### 5.17.2.4 enum telux::rsp::ProfileClass [strong]

Indicates profile class.

#### Enumerator

**UNKNOWN** No info about profile class  
**TEST** Test profile  
**PROVISIONING** Provisioning profile  
**OPERATIONAL** Operational profile

### 5.17.2.5 enum telux::rsp::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

### 5.17.2.6 enum telux::rsp::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

### 5.17.2.7 enum telux::rsp::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 [onAduTransfer](#) (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 void [onServiceStatusChange](#) (const [telux::common::ServiceStatus](#) status)
- virtual [~IRemoteSimListener](#) ()

#### 5.18.1.1.1 Constructors and Destructors

5.18.1.1.1 virtual [telux::tel::IRemoteSimListener::~~IRemoteSimListener](#) ( ) [[virtual](#)]

Destructor of [IRemoteSimListener](#)

#### Note

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

#### 5.18.1.1.2 Member Function Documentation

5.18.1.1.2.1 virtual void [telux::tel::IRemoteSimListener::onAduTransfer](#) ( 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)

**Note**

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

**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.

**Note**

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

**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.

**Note**

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

**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.

**Note**

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

**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.

**Note**

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

**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.

**Note**

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

### 5.18.1.1.2.7 virtual void telux::tel::IRemoteSimListener::onServiceStatusChange ( const telux↔ ::common::ServiceStatus *status* ) [virtual]

This function is called when the modem service goes down or comes up.

#### Parameters

in	<i>status</i>	Service status
----	---------------	----------------

#### Note

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

### 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`

#### Note

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

### 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.

#### Note

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

#### 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.

#### Note

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

**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.

#### Note

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

**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.

#### Note

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

**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.

**Note**

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

**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.

**Note**

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

**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.

**Parameters**

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.

**Note**

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



**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.

#### Note

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

**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.

#### Note

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

**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.

**Note**

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

**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.

**Note**

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

**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.

**Note**

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

#### 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

**Note**

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

#### 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](#)

**Note**

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

## 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.

##### Note

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

##### Public member functions

- `std::shared_ptr< IModemConfigManager > getModemConfigManager ()`
- `~ConfigFactory ()`

##### Static Public Member Functions

- `static ConfigFactory & getInstance ()`

#### 5.19.1.1.1 Constructors and Destructors

##### 5.19.1.1.1.1 `telux::config::ConfigFactory::~~ConfigFactory ( )`

#### 5.19.1.1.2 Member Function Documentation

##### 5.19.1.1.2.1 `static ConfigFactory& telux::config::ConfigFactory::getInstance ( ) [static]`

Get instance of Config Factory

##### 5.19.1.1.2.2 `std::shared_ptr<IModemConfigManager> telux::config::ConfigFactory::getModemConfigManager ( )`

Get instance of ModemConfig manager

##### Returns

pointer of `IModemConfigManager` object.

##### Note

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

### 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.

#### 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 [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.

**Note**

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

**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.

**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 [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.

**Note**

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

**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.

**Note**

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

**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.

**Note**

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

**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.

#### Note

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

**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.

#### Note

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

**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.

**Note**

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

**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

**Note**

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

**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

**Note**

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

**5.19.1.4.1.9 virtual telux::common::Status telux::config::IModemConfigManager::getAutoSelectionMode ( 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

**Note**

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

**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.

**Note**

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

**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.

#### Note

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

**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.

#### Note

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

## 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 Telematics\_cv2x\_cpp

•

### 5.20.1 Data Structure Documentation

#### 5.20.1.1 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](#) ()

##### 5.20.1.1.1 Constructors and Destructors

5.20.1.1.1.1 virtual telux::cv2x::ICv2xThrottleManagerListener::~~ICv2xThrottleManagerListener ( )  
[virtual]

Destructor of [ICv2xThrottleManagerListener](#)

##### 5.20.1.1.2 Member Function Documentation

5.20.1.1.2.1 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.

5.20.1.1.2.2 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

**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.

**5.20.1.2 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](#) ()

**5.20.1.2.1 Constructors and Destructors**

**5.20.1.2.1.1** virtual [telux::cv2x::ICv2xThrottleManager::~ICv2xThrottleManager](#) ( ) [**virtual**]

**5.20.1.2.2 Member Function Documentation**

**5.20.1.2.2.1** 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.

**5.20.1.2.2.2** `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.

**5.20.1.2.2.3** `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.
----	-----------------	---

**5.20.1.2.2.4** `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.
----	-----------------	---

**5.20.1.2.2.5** `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.

## 5.21 V2x\_api\_common

The following common typedefs and macros are used across all parts of this platform.

### 5.21.1 Data Structure Documentation

#### 5.21.1.1 struct v2x\_api\_ver\_t

Contains retrieved information about the SDK API library that is called. Each SDK component (Kinematics, Radio, Vehicle Data) implements a method to return this structure.

##### Data fields

Type	Field	Description
uint32_t	version_num	Version number of the interface.
char	build_date_↔ str[128]	Date of the build (part of the data string).
char	build_time_↔ str[128]	Time of the build (part of the data string).
char	build_details↔ _str[128]	Build details (part of the data string).

### 5.21.2 Enumeration Type Documentation

#### 5.21.2.1 enum v2x\_status\_enum\_type

Valid types for subsystem status, and return status codes for API function calls and callbacks.

##### Enumerator

**V2X\_STATUS\_SUCCESS** Operation is successful.

**V2X\_STATUS\_FAIL** Operation is unsuccessful. This is a generic error failure status that can be due to radio hardware resource limitations, geofencing, and so on.

**V2X\_STATUS\_ENO\_MEMORY** Failure due to a memory allocation issue.

**V2X\_STATUS\_EBADPARAM** One of the supplied parameters is bad.

**V2X\_STATUS\_EALREADY** Attempted step was already issued, and this call is not required.

**V2X\_STATUS\_KINETICS\_PLACEHOLDER** Begin the return codes associated with the Kinematics interface.

**V2X\_STATUS\_RADIO\_PLACEHOLDER** Begin the return codes associated with the Radio interface.

**V2X\_STATUS\_ECHANNEL\_UNAVAILABLE** Requested radio frequency cannot be used at this time.

**V2X\_STATUS\_VEHICLE\_PLACEHOLDER** Begin the return codes associated with the Vehicle Data interface.



## 5.22 V2x\_api\_kinematics

Abstraction of the system GNSS + DR solution for returning precision fixes with low latency via callbacks. This solution is used each time a fix is available, and it supports multiple callbacks to a short list of clients.

Common types are used for the motion and location reporting system of the platform. These types include the structures that are used to both configure the Kinematics subsystem and to report periodic fixes. The fixes are combinations of inertial/motion data and GNSS solutions determined (possibly directly) from satellite processing or dead-reckoning in degraded SV reception.

### 5.22.1 Define Documentation

#### 5.22.1.1 #define V2X\_KINEMATICS\_HANDLE\_BAD (-1)

Invalid handle returned by [v2x\\_kinematics\\_init\(\)](#) upon an error.

### 5.22.2 Data Structure Documentation

#### 5.22.2.1 struct v2x\_GNSSstatus\_t

Contains status information for the GNSS satellite.

This structure is used for each reported fix to indicate the quality of the available constellation (or whether the constellation is not available).

##### Data fields

Type	Field	Description
bool	unavailable	Specifies whether a constellation is not equipped or is unavailable. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – GNSS is available</li> <li>• 1 – GNSS is unavailable</li> </ul>
bool	aPDOPof↔ Under5	Specifies whether dilution of precision is greater than 5. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not greater than 5</li> <li>• 1 – Greater than 5</li> </ul>
bool	inViewOf↔ Under5	Specifies whether fewer than five satellites are in view. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Five or more satellites are in view</li> <li>• 1 – Fewer than five satellites are in view</li> </ul>
bool	local↔ Corrections↔ Present	Specifies whether DGPS type corrections are used. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not used</li> <li>• 1 – Used</li> </ul>
bool	network↔ Corrections↔ Present	Specifies whether RTK type corrections are used. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not used</li> <li>• 1 – Used</li> </ul>

### 5.22.2.2 struct v2x\_gnss\_fix\_rates\_supported\_list\_t

Defines supported GNSS fix generation rates (such as 1 Hz, 5 Hz, 10 Hz).

#### Data fields

Type	Field	Description
uint32_t	qty_rates_↔ supported	Specify whether the listing or discovery of the supported rates is supported.  <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not supported</li> <li>• 1 – Supported</li> </ul>
pb_size_t	rates_↔ supported_hz↔ _array_count	Number of supported rates.
uint32_t	rates_↔ supported_hz↔ _array[32]	Array of data rates supported by the API.

### 5.22.2.3 struct v2x\_init\_t

Defines client initialization.

#### Data fields

Type	Field	Description
uint32_t	log_level_mask	Log levels as defined in syslog.h.
char	server_ip_↔ addr[32]	IP address of the server.

### 5.22.2.4 struct v2x\_kinematics\_capabilities\_t feature\_flags\_t

Defines Kinematics features supported by the hardware.

#### Data fields

Type	Field	Description
bool	has_3_axis_↔ gyro	Specifies whether the hardware supports 3-axis gyro. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not supported</li> <li>• 1 – Supported</li> </ul>
bool	has_3_axis_↔ accelerometer	Specifies whether the hardware supports the 3-axis accelerometer. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not supported</li> <li>• 1 – Supported</li> </ul>
bool	has_imu_↔ supplemented_↔ _dead_↔ reckoning	Specifies whether a dead reckoning (DR) solution is available and enabled or only GNSS is the result. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – GNSS is available</li> <li>• 1 – DR is available</li> </ul>
bool	has_yaw_rate_↔ _sensor	Specifies whether the IMU includes a yaw rate sensor. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Does not include sensor</li> <li>• 1 – Includes sensor</li> </ul>
bool	used_vehicle_↔ _speed	Specifies whether the DR algorithm uses the vehicle speed sensor. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Does not use sensor</li> <li>• 1 – Uses sensor</li> </ul>
bool	used_single_↔ wheel_ticks	Specifies whether the DR algorithm uses the single wheel ticks. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Does not use ticks</li> <li>• 1 – Uses ticks</li> </ul>
bool	used_front_↔ differential_↔ wheel_ticks	Specifies whether the DR algorithm uses two front differential wheel ticks. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Does not use ticks</li> <li>• 1 – Uses ticks</li> </ul>
bool	used_rear_↔ differential_↔ wheel_ticks	Specifies whether the DR algorithm uses two rear differential wheel ticks. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Does not use ticks</li> <li>• 1 – Uses ticks</li> </ul>

Type	Field	Description
bool	used_vehicle_↔ _dynamic_↔ model	Specifies whether the DR algorithm uses vehicle dynamic model factoring in differential ticks, steering, and so on. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Does not use factoring</li> <li>• 1 – Uses factoring</li> </ul>

### 5.22.2.5 struct v2x\_rates\_t

Defines the rate type.

#### Data fields

Type	Field	Description
uint32_t	rate_report_hz	Requested or reported current number of fixes per second.
uint32_t	offset_↔ nanoseconds	Currently unsupported.

### 5.22.2.6 struct v2x\_kinematics\_capabilities\_t

Returned via v2x\_kinematics\_get\_capabilities() for the client to discover the lower level function that this system supports.

#### Data fields

Type	Field	Description
v2x_↔ kinematics_↔ _capabilities_↔ _t_feature_↔ flags_t	feature_flags	Features supported by the API.
uint32_t	max_fix_rate_↔ _supported_hz	Highest rate platform that supports GNSS fix generation (in Hz).
v2x_gnss_↔ fix_rates_↔ supported_↔ list_t	rates_list	Supported fix rates.

### 5.22.2.7 struct v2x\_location\_fix\_t

Contains a standardized set of parameters that are used for ITS applications.

The contents of this structure do not include every possible GNSS element (for example, raw range data is not included). These fields are via low latency for safety applications, both to use locally and to load into a J2945/1 or ETSI G5 EN302.637-2 CAM. For example, the fields are used for CAM-type and BSM-type safety beacons.

This structure is populated for each location or dead-reckoning fix, and it is supplied on the fix available callback.

Predefined J2735s can be used to communicate raw SV observations and RTK correction data. Currently, however, they are not supplied from this structure.

#### Data fields

Type	Field	Description
double	utc_fix_time	UTC time in seconds.
<a href="#">v2x_fix_mode_t</a>	fix_mode	Location engine used to produce this record. <b>Supported values:</b> No fix, 2D, 3D, RTCM
double	latitude	Latitude in degrees.
double	longitude	Longitude in degrees.
double	altitude	Altitude in meters above the geoid (mean sea level).
uint32_t	qty_SV_in_view	Number of usable space vehicles (SV) that should be in view.
uint32_t	qty_SV_used	Actual number of SVs used in this fix calculation.
<a href="#">v2x_GNSS_status_t</a>	gnss_status	Status of the GNSS data.
bool	has_SemiMajorAxisAccuracy	Specifies whether the value of the SemiMajorAxisAccuracy field is valid. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not valid</li> <li>• 1 – Valid</li> </ul>
double	SemiMajorAxisAccuracy	Accuracy of the major axis, in meters.
bool	has_SemiMinorAxisAccuracy	Specifies whether the value of the SemiMinorAxisAccuracy field is valid. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not valid</li> <li>• 1 – Valid</li> </ul>
double	SemiMinorAxisAccuracy	Accuracy of the minor axis, in meters.
bool	has_SemiMajorAxisOrientation	Specifies whether the value of the SemiMajorAxisOrientation field is valid. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not valid</li> <li>• 1 – Valid</li> </ul>

Type	Field	Description
double	SemiMajor↔ AxisOrientation	Orientation of the major axis, in meters.
bool	has_heading	Specifies whether the value of the heading field is valid. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not valid</li> <li>• 1 – Valid</li> </ul>
double	heading	Track degrees relative to true north.
bool	has_velocity	Specifies whether the value of the velocity field is valid. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not valid</li> <li>• 1 – Valid</li> </ul>
double	velocity	Speed over ground in meters/second.
bool	has_climb	Specifies whether the value of the climb field is valid. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not valid</li> <li>• 1 – Valid</li> </ul>
double	climb	Vertical speed in meters/second.
bool	has_lateral_↔ acceleration	Specifies whether the value of the lateral_acceleration field is valid. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not valid</li> <li>• 1 – Valid</li> </ul>
double	lateral_↔ acceleration	Acceleration in a latitudinal direction, in meters/second <sup>^2</sup> .
bool	has_↔ longitudinal_↔ acceleration	Specifies whether the value of the longitudinal acceleration field is valid. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not valid</li> <li>• 1 – Valid</li> </ul>
double	longitudinal_↔ acceleration	Acceleration in a longitudinal direction, in meters/second <sup>^2</sup> .
bool	has_vehicle_↔ vertical_↔ acceleration	Specifies whether the value of the vehicle_vertical_acceleration field is valid. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not valid</li> <li>• 1 – Valid</li> </ul>
double	vehicle_↔ vertical_↔ acceleration	Vertical acceleration of the vehicle in G force.
bool	has_yaw_rate↔ _degrees_per↔ _second	Specifies whether the value of the yaw_rate_degrees_per_second field is valid. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not valid</li> <li>• 1 – Valid</li> </ul>

Type	Field	Description
double	yaw_rate_↔ degrees_per_↔ second	Yaw rate in degrees/second, per SAE J2735.
bool	has_yaw_rate_↔ _95pct_↔ confidence	Specifies whether the value of the yaw_rate_95pct_confidence field is valid. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not valid</li> <li>• 1 – Valid</li> </ul>
double	yaw_rate_↔ 95pct_↔ confidence	95% confidence (2 sigma) on the yaw rate in degrees/second.
bool	has_lane_↔ position_↔ number	Specifies whether the value of the lane_position_number field is valid. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not valid</li> <li>• 1 – Valid</li> </ul>
double	lane_position_↔ _number	Current lane number, where 0 is either the outer-most edge of the hard shoulder or off-road.
bool	has_lane_↔ position_↔ 95pct_↔ confidence	Specifies whether the value of the lane_position_95pct_confidence field is valid. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not valid</li> <li>• 1 – Valid</li> </ul>
double	lane_position_↔ _95pct_↔ confidence	95% confidence range on the lane position.
bool	has_time_↔ confidence	Specifies whether the value of the time_confidence field is valid. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not valid</li> <li>• 1 – Valid</li> </ul>
float	time_↔ confidence	95% (2 sigma) confidence in number of seconds.
bool	has_heading_↔ confidence	Specifies whether the value of the heading_confidence field is valid. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not valid</li> <li>• 1 – Valid</li> </ul>
float	heading_↔ confidence	95% heading confidence in degrees.
bool	has_velocity_↔ confidence	Specifies whether the value of the velocity_confidence field is valid. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not valid</li> <li>• 1 – Valid</li> </ul>
float	velocity_↔ confidence	95% velocity confidence in meters/second.

Type	Field	Description
bool	has_elevation↔ _confidence	Specifies whether the value of the elevation_confidence field is valid. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not valid</li> <li>• 1 – Valid</li> </ul>
float	elevation_↔ confidence	95% uncertainty range (2 sigma) confidence in meters.
uint32_t	leap_seconds	Indicates that both UTC and GPS are required because IEEE 1609.2 security requires operations to be performed in raw GPS time.

## 5.22.3 Enumeration Type Documentation

### 5.22.3.1 enum v2x\_fix\_mode\_t

Valid GNSS fix modes.

#### Enumerator

**V2X\_GNSS\_MODE\_NOT\_SEEN** SV is unavailable or not in view.

**V2X\_GNSS\_MODE\_NO\_FIX** No SV fix.

**V2X\_GNSS\_MODE\_2D** 2D fix with latitude and longitude information.

**V2X\_GNSS\_MODE\_3D** 3D fix with latitude, longitude, and altitude information.

## 5.22.4 Function Documentation

### 5.22.4.1 v2x\_api\_ver\_t v2x\_kinematics\_api\_version ( void )

Gets the compiled API version interface (as an integer number).

#### Returns

[v2x\\_api\\_ver\\_t](#) – Filled with the version number, build date, and detailed build information.



### 5.22.4.2 `v2x_kinematics_handle_t v2x_kinematics_init ( v2x_init_t * param, v2x_kinematics_init_callback_t cb, void * context )`

Initializes the Kinematics library.

#### Associated data types

[v2x\\_init\\_t](#)  
[v2x\\_kinematics\\_init\\_callback\\_t](#)

#### Parameters

in	<i>param</i>	Pointer to the structure that contains parameters for the IP address of the server, logging level, and so on.
in	<i>cb</i>	Callback function called when initialization is complete.
in	<i>context</i>	Pointer to the application context for use with the callbacks, which can help the caller code.

#### Returns

Handle number to use with subsequent calls.

[V2X\\_KINEMATICS\\_HANDLE\\_BAD](#) – Upon an error.

### 5.22.4.3 `v2x_status_enum_type v2x_kinematics_start_rate_notification ( v2x_kinematics_handle_t handle, v2x_kinematics_rate_notification_listener_t cb, void * context )`

Gets the current rate and offset from the Kinematics library.

#### Associated data types

[v2x\\_kinematics\\_handle\\_t](#)

[v2x\\_kinematics\\_rate\\_notification\\_listener\\_t](#)

#### Parameters

in	<i>handle</i>	Handle number to use with subsequent calls. If there is an error in initialization, the value is -1.
in	<i>cb</i>	Callback function used to report rate notification changes.
in	<i>context</i>	Pointer to the application context for use with the callbacks, which can help the caller code.

#### Returns

Indication of success or failure from [v2x\\_status\\_enum\\_type](#).

#### 5.22.4.4 `v2x_status_enum_type v2x_kinematics_set_rate ( v2x_kinematics_handle_t handle, v2x_rates_t * rate, v2x_kinematics_set_rate_callback_t cb, void * context )`

Sets the current rate and offset from the Kinematics library.

##### Associated data types

[v2x\\_kinematics\\_handle\\_t](#)

[v2x\\_rates\\_t](#)

[v2x\\_kinematics\\_set\\_rate\\_callback\\_t](#)

##### Parameters

in	<i>handle</i>	Handle number to use with subsequent calls. If there is an error in initialization, the value is -1.
in	<i>rate</i>	Pointer to the rate structure filled with the fix timing parameters.
in	<i>cb</i>	Callback function called when rates and offsets are successfully set. This parameter can be NULL.
in	<i>context</i>	Pointer to the application context for use with the callbacks, which can help the caller code.

##### Returns

Indication of success or failure from [v2x\\_status\\_enum\\_type](#).

**5.22.4.5** `v2x_status_enum_type v2x_kinematics_register_listener ( v2x_kinematics_↔  
_handle_t handle, v2x_kinematics_newfix_listener_t listener, void * context  
)`

Registers for a Kinematics result listener callback at the requested rate.

#### Associated data types

[v2x\\_kinematics\\_handle\\_t](#)

[v2x\\_kinematics\\_newfix\\_listener\\_t](#)

#### Parameters

in	<i>handle</i>	Handle number to use with subsequent calls. If there is an error in initialization, the value is -1.
in	<i>listener</i>	Callback function to use for this listener.
in	<i>context</i>	Pointer to the application context for use with the callbacks, which can help the caller code.

#### Detailed description

This function requests GNSS fix/motion callbacks at a specified rate (Hz) with a specified offset.

Only certain rates are supported (such as 1 Hz, 2 Hz, 5 Hz, 10 Hz), which are obtained from `v2x_kinematics_get_capabilities()`.

Currently, a request cannot be made for a rate slower than 1 Hz.

#### Returns

Indication of success or failure from [v2x\\_status\\_enum\\_type](#).

#### 5.22.4.6 `v2x_status_enum_type v2x_kinematics_deregister_listener ( v2x_kinematics_handle_t handle, v2x_kinematics_deregister_callback_t cb, void * context )`

Deregisters a previously registered GNSS fix that the listener established earlier via [v2x\\_kinematics\\_register\\_listener\(\)](#).

##### Associated data types

[v2x\\_kinematics\\_handle\\_t](#)  
[v2x\\_kinematics\\_deregister\\_callback\\_t](#)

##### Parameters

in	<i>handle</i>	Handle number of the registered fix.
in	<i>cb</i>	Callback function to use for this listener. This parameter can be NULL.
in	<i>context</i>	Pointer to the application context for use with the callbacks, which can help the caller code.

##### Returns

Indication of success or failure from [v2x\\_status\\_enum\\_type](#).

#### 5.22.4.7 `v2x_status_enum_type v2x_kinematics_final ( v2x_kinematics_handle_t handle, v2x_kinematics_final_callback_t cb, void * context )`

Terminates the Kinematics library.

##### Associated data types

[v2x\\_kinematics\\_handle\\_t](#)

[v2x\\_kinematics\\_final\\_callback\\_t](#)

##### Parameters

in	<i>handle</i>	Handle number of the library.
in	<i>cb</i>	Callback function called when termination is complete. This parameter can be NULL.
in	<i>context</i>	Pointer to the application context for use with the callbacks, which can help the caller code.

##### Returns

Indication of success or failure from [v2x\\_status\\_enum\\_type](#).

#### 5.22.4.8 void v2x\_kinematics\_enable\_fixes ( v2x\_kinematics\_handle\_t *handle* )

Enables the Kinematics fixes from GNSS.

##### Associated data types

[v2x\\_kinematics\\_handle\\_t](#)

##### Parameters

in	<i>handle</i>	Unique identifier for the library.
----	---------------	------------------------------------

##### Returns

None.

#### 5.22.4.9 void v2x\_kinematics\_disable\_fixes ( v2x\_kinematics\_handle\_t *handle* )

Disables the Kinematics fixes from GNSS.

##### Associated data types

[v2x\\_kinematics\\_handle\\_t](#)

##### Parameters

in	<i>handle</i>	Unique identifier for the library.
----	---------------	------------------------------------

##### Returns

None.

## 5.23 V2x\_api\_radio

Abstraction of the radio driver parameters for a V2X broadcast socket interface, including 3GPP CV2X QoS bandwidth contracts.

### 5.23.1 Define Documentation

#### 5.23.1.1 #define V2X\_RADIO\_HANDLE\_BAD (-1)

Invalid handle returned by `v2x_radio_init()` and `v2x_radio_init_v2()` upon an error.

#### 5.23.1.2 #define V2X\_MAX\_RADIO\_SESSIONS (10)

Limit on the number of simultaneous RmNet Radio interfaces this library can have open at once.

Typically, there are only a few actual radios. On the same radio however, one interface can be for IP traffic, and another interface can be for non-IP traffic.

#### 5.23.1.3 #define V2X\_RX\_WILDCARD\_PORTNUM (9000)

Wildcard value for a port number. When the wildcard is used, all V2X received traffic is routed.

#### 5.23.1.4 #define MAX\_POOL\_IDS\_LIST\_LEN (20)

Maximum length of the pool ID list that is returned in `v2x_iface_capabilities_t`.

#### 5.23.1.5 #define MAX\_MALICIOUS\_IDS\_LIST\_LEN (50)

Maximum length of the malicious ID list that can be passed in `v2x_radio_update_trusted_ue_list()`.

#### 5.23.1.6 #define MAX\_TRUSTED\_IDS\_LIST\_LEN (50)

Maximum length of the trusted ID list that can be passed in `v2x_radio_update_trusted_ue_list()`.

#### 5.23.1.7 #define MAX\_SUBSCRIBE\_SIDS\_LIST\_LEN (10)

Maximum length for the subscribed service ID list that can be passed in `v2x_radio_rx_sock_create_and_bind_v2()`.

### 5.23.2 Data Structure Documentation

#### 5.23.2.1 struct trusted\_ue\_info\_t

Contains time confidence, position confidence, and propagation delay for a trusted UE.

##### Data fields

Type	Field	Description
uint32_t	source_l2_id	L2 ID of the trusted source
float	time_↔ uncertainty	Time uncertainty in milliseconds.



Type	Field	Description
uint16_t	time_↔ confidence_↔ level	Deprecated. Use time_uncertainty instead. Confidence level of the time period. <b>Supported values:</b> 0 through 127, where 0 is invalid or unavailable and 127 is the most confident
uint16_t	position_↔ confidence_↔ level	Confidence level of the position. <b>Supported values:</b> 0 through 127, where 0 is invalid or unavailable and 127 is the most confident
uint32_t	propagation_↔ delay	Propagation delay in microseconds.

### 5.23.2.2 struct tx\_pool\_id\_info\_t

Contains minimum and maximum frequencies for a Tx pool ID. This struct is used in [v2x\\_iface\\_capabilities\\_t](#).

#### Data fields

Type	Field	Description
uint8_t	pool_id	ID of the Tx pool.
uint16_t	min_freq	Minimum frequency in MHz.
uint16_t	max_freq	Maximum frequency in MHz.

### 5.23.2.3 struct v2x\_iface\_capabilities\_t

Contains information on the capabilities of a Radio interface.

#### Data fields

Type	Field	Description
int	link_ip_MTU↔ _bytes	Maximum data payload length (in bytes) of a packet supported by the IP Radio interface.
int	link_non_ip↔ MTU_bytes	Maximum data payload length (in bytes) of a packet supported by the non-IP Radio interface.
v2x↔ concurrency↔ sel_t	max↔ supported↔ concurrency	Indicates whether this interface supports concurrent WWAN with V2X (PC5).
uint16_t	non_ip_tx↔ payload↔ offset_bytes	<p>Byte offset in a non-IP Tx packet before the actual payload begins. In 3GPP CV2X, the first byte after the offset is the 1-byte V2X Family ID.</p> <p>This offset is to the left for a per-packet Tx header that includes Tx information that might be inserted in front of the packet payload (in subsequent releases).</p> <p>An example of Tx information is MAC/Phy parameters (power, rate, retransmissions policy, and so on).</p> <p>Currently, this value is expected to be 0. But it is reserved to support possible per-packet Tx/Rx headers that might be added in future releases of this API.</p>
uint16_t	non_ip_rx↔ payload↔ offset_bytes	<p>Byte offset in a non-IP Rx packet before the actual payload begins. Initially, this value is zero. But it allows for later insertion of per-packet Rx information (sometimes called metadata) to be added to the front of the data payload. An example of Rx information is MAC/Phy measurements (receive signal strength, timestamps, and so on).</p> <p>The V2X Family ID is considered as part of the payload in the 3GPP CV2X. Higher layers (applications that are clients to this API) must remove or advance past that 1 byte to get to the more familiar actual WSMP/Geonetworking payload.</p>
uint16_t	int_min↔ periodicity↔ multiplier_ms	<p>Lowest number of milliseconds requested for a bandwidth.</p> <p>This value is also the basis for all possible bandwidth reservation periods. For example, if this multiplier=100 ms, applications can only reserve bandwidths of 100 ms, 200 ms, up to 1000 ms.</p>
uint16_t	int↔ maximum↔ periodicity_ms	Least frequent bandwidth periodicity that is supported. Above this value, use event-driven periodic messages of a period larger than this value.
unsigned	supports↔ 10ms↔ periodicity: 1	<p>Indicates whether n*10 ms periodicities are supported.</p> <p><b>Supported values:</b></p> <ul style="list-style-type: none"> <li>• 0 – Not supported</li> <li>• 1 – Supported</li> </ul>

Type	Field	Description
unsigned	supports_↔ 20ms_↔ periodicity: 1	Indicates whether an n*20 ms bandwidth reservation is supported. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not supported</li> <li>• 1 – Supported</li> </ul>
unsigned	supports_↔ 50ms_↔ periodicity: 1	Indicates whether 50 ms periodicity is supported. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not supported</li> <li>• 1 – Supported</li> </ul>
unsigned	supports_↔ 100ms_↔ periodicity: 1	Indicates whether the basic minimum periodicity of 100 ms is supported. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not supported</li> <li>• 1 – Supported</li> </ul>
unsigned	max_quantity↔ _of_auto_↔ retrans: 4	Maximum number automatic retransmissions. <b>Supported values:</b> 0 through 15
unsigned	size_of_↔ layer2_mac_↔ address: 4	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.
uint16_t	v2x_number_↔ of_priority_↔ levels	Number of different priority levels supported. For example, 8 is the current 3GPP standard (where a lower number means a higher priority).
uint16_t	highest_↔ priority_value	Least urgent priority number supported by this radio. Higher numbers are lower priority, so if the full range is supported, this value is <a href="#">V2X_PRIO_BACKGROUND</a> .
uint16_t	lowest_↔ priority_value	Highest priority value (most urgent traffic). Lower numbers are higher priority, so if the highest level supported this value is <a href="#">V2X_PRIO_MOST_URGENT</a> .
uint16_t	max_qty_SP↔ S_flows	Maximum number of supported SPS reservations.
uint16_t	max_qty_non↔ _SPS_flows	Maximum number of supported event flows (non-SPS ports).
int32_t	max_tx_pwr	Maximum supported transmission power in dBm.
int32_t	min_tx_pwr	Minimum supported transmission power in dBm.
uint32_t	tx_pool_ids_↔ supported_len	Length of the tx_pool_ids_supported array.
<a href="#">tx_pool_id_↔ info_t</a>	tx_pool_ids_↔ supported[M↔ <a href="#">AX_POOL_I↔ DS_LIST_L↔ EN]</a>	Array of Tx pool IDs and their associated minimum and maximum frequencies.

### 5.23.2.4 struct v2x\_tx\_bandwidth\_reservation\_t

Used when requesting a QoS bandwidth contract, which is implemented in PC5 3GPP V2-X radio as a *Semi Persistent Flow* (SPS).

The underlying radio providing the interface might support periodicities of various granularity in 100 ms integer multiples (such as 200 ms, 300 ms, and 400 ms).

The reservation is also used internally as a handle.

#### Data fields

Type	Field	Description
int	v2xid	Variable length 4-byte PSID or ITS_AID, or another application ID.
<a href="#">v2x_priority_et</a>	priority	Specifies one of the 3GPP levels of priority for the traffic that is pre-reserved on the SPS flow. Use <a href="#">v2x_radio_query_parameters()</a> to get the exact number of supported priority levels.
int	period_↔ interval_ms	Bandwidth-reserved periodicity interval in milliseconds. There are limits on which intervals the underlying radio supports. Use the capabilities query method to discover the <code>int_min_periodicity_multiplier_ms</code> and <code>int_maximum_periodicity_ms</code> supported intervals.
int	tx_↔ reservation_↔ size_bytes	Number of Tx bandwidth bytes that are sent every periodicity interval.

### 5.23.2.5 struct v2x\_chan\_meas\_params\_t

Contains the measurement parameters for configuring the MAC/Phy radio channel measurements (such as CBR utilization).

The radio chip contains requests on radio measurement parameters that API clients can use to specify the following:

- How their higher-level application requires the CBR/CBP to be measured
- Over which time window
- When to send a report

#### Data fields

Type	Field	Description
int	channel_↔ measurement_↔ _interval_us	Duration in microseconds of the sliding window size.
int	rs_threshold_↔ decidbm	Parameter to the radio CBR measurement that is used for determining how busy the channel is. Signals weaker than the specified receive strength (RSRP, or RSSI) are not considered to be in use (busy).

### 5.23.2.6 struct v2x\_chan\_measurements\_t

Periodically returned by the radio with all measurements about the radio channel, such as the amount of noise and bandwidth saturation (channel\_busy\_percentage, or CBR).

#### Data fields

Type	Field	Description
float	channel_busy_↔ _percentage	No measurement parameters are supplied.
float	noise_floor	Measurement of the background noise for a quiet channel.
float	time_↔ uncertainty	V2X time uncertainty in milliseconds.

### 5.23.2.7 struct v2x\_radio\_calls\_t

Contains callback functions used in a v2x\_radio\_init() and v2x\_radio\_init\_v2 call.

The radio interface uses these callback functions for events such as completion of initialization, a Layer-02 MAC address change, or a status event (loss of sufficient GPS time precision to transmit).

These callbacks are related to a specific radio interface, and its MAC/Phy parameters, such as transmit power, bandwidth utilization, and changes in radio status.

#### Data Fields

- void(\* v2x\_radio\_init\_complete)(v2x\_status\_enum\_type status, void \*context)
- void(\* v2x\_radio\_status\_listener)(v2x\_event\_t event, void \*context)
- void(\* v2x\_radio\_chan\_meas\_listener)(v2x\_chan\_measurements\_t \*measurements, void \*context)
- void(\* v2x\_radio\_l2\_addr\_changed\_listener)(int new\_l2\_address, void \*context)
- void(\* v2x\_radio\_macphy\_change\_complete\_cb)(void \*context)
- void(\* v2x\_radio\_capabilities\_listener)(v2x\_iface\_capabilities\_t \*caps, void \*context)
- void(\* v2x\_service\_status\_listener)(v2x\_service\_status\_t status, void \*context)

#### 5.23.2.7.1 Field Documentation

##### 5.23.2.7.1.1 void(\* v2x\_radio\_calls\_t::v2x\_radio\_init\_complete)(v2x\_status\_enum\_type status, void \*context)

Callback that indicates initialization is complete.

#### Associated data types

[v2x\\_status\\_enum\\_type](#)

#### Parameters

in	<i>status</i>	Updated current radio status that indicates whether transmit and receive are ready.
in	<i>context</i>	Pointer to the context that was supplied during initial registration.

**5.23.2.7.1.2 void(\* v2x\_radio\_calls\_t::v2x\_radio\_status\_listener) (v2x\_event\_t event, void \*context)**

Callback made when the status in the radio changes. For example, in response to a fault when there is a loss of GPS timing accuracy.

**Associated data types**

[v2x\\_event\\_t](#)

**Parameters**

in	<i>event</i>	Delivery of the event that just occurred, such losing the ability to transmit.
in	<i>context</i>	Pointer to the context of the caller who originally registered for this callback.

**5.23.2.7.1.3 void(\* v2x\_radio\_calls\_t::v2x\_radio\_chan\_meas\_listener) (v2x\_chan\_measurements\_t \*measurements, void \*context)**

Callback made from lower layers when periodic radio measurements are prepared.

**Associated data types**

[v2x\\_chan\\_measurements\\_t](#)

**Parameters**

in	<i>measurements</i>	Pointer to the periodic measurements.
in	<i>context</i>	Pointer to the context of the caller who originally registered for this callback.

**5.23.2.7.1.4 void(\* v2x\_radio\_calls\_t::v2x\_radio\_l2\_addr\_changed\_listener) (int new\_l2\_address, void \*context)**

Callback made by the platform SDK when the MAC address (L2 SRC address) changes.

**Parameters**

in	<i>new_l2_address</i>	New L2 source address as an integer (because the L2 address is 3 bytes).
in	<i>context</i>	Pointer to the context of the caller who originally registered for this callback.

**5.23.2.7.1.5 void(\* v2x\_radio\_calls\_t::v2x\_radio\_macphy\_change\_complete\_cb) (void \*context)**

Callback made to indicate that the requested radio MAC/Phy change (such as channel/frequency and power) has completed.

**Parameters**

in	<i>context</i>	Pointer to the context of the caller who originally registered for this callback.
----	----------------	---

**5.23.2.7.1.6 void(\* v2x\_radio\_calls\_t::v2x\_radio\_capabilities\_listener) (v2x\_iface\_capabilities\_t \*caps, void \*context)**

Callback made when V2X capabilities change.

**Associated data types**

[v2x\\_iface\\_capabilities\\_t](#)

**Parameters**

in	<i>caps</i>	Pointer to the capabilities of this interface.
in	<i>context</i>	Pointer to the context of the caller who originally registered for this callback.

**5.23.2.7.1.7 void(\* v2x\_radio\_calls\_t::v2x\_service\_status\_listener) (v2x\_service\_status\_t status, void \*context)**

Callback made when the service status changes.

**Associated data types**

[v2x\\_service\\_status\\_t](#)

**Parameters**

in	<i>status</i>	Service status.
in	<i>context</i>	Pointer to the context of the caller who originally registered for this callback.



### 5.23.2.8 struct v2x\_sps\_mac\_details\_t

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	periodicity_↔ in_use_ns	Actual transmission interval period (in nanoseconds) scheduled relative to 1PP 0:00.00 time.
uint16_t	currently_↔ reserved_↔ periodic_bytes	Actual number of bytes currently reserved at the MAC layer. This number can be slightly larger than original request.
uint32_t	tx_↔ reservation_↔ offset_ns	Actual offset, from a 1PPS pulse and Tx flow periodicity, that the MAC selected and is using for the transmit reservation.  If data goes to the radio with enough time, it can be transmitted on the medium in the next immediately scheduled slot.
uint64_t	utc_time_ns	Absolute UTC start time of next selected grant, in nanoseconds.

### 5.23.2.9 struct v2x\_per\_sps\_reservation\_calls\_t

Callback functions used in [v2x\\_radio\\_tx\\_sps\\_sock\\_create\\_and\\_bind\(\)](#) calls.

#### Data Fields

- void(\* [v2x\\_radio\\_l2\\_reservation\\_change\\_complete\\_cb](#) )(void \*context, [v2x\\_sps\\_mac\\_details\\_t](#) \*details)
- void(\* [v2x\\_radio\\_sps\\_offset\\_changed](#) )(void \*context, [v2x\\_sps\\_mac\\_details\\_t](#) \*details)

#### 5.23.2.9.1 Field Documentation

##### 5.23.2.9.1.1 void(\* [v2x\\_per\\_sps\\_reservation\\_calls\\_t::v2x\\_radio\\_l2\\_reservation\\_change\\_complete\\_cb](#) )(void \*context, [v2x\\_sps\\_mac\\_details\\_t](#) \*details)

Callback made upon completion of a reservation change that a [v2x\\_radio\\_tx\\_reservation\\_change\(\)](#) call initiated for a MAC/Phy contention.

The current SPS offset and reservation parameter are passed in the details structure returned by the pointer details.

#### Associated data types

[v2x\\_sps\\_mac\\_details\\_t](#)

#### Parameters

in	<i>context</i>	Pointer to the application context.
in	<i>details</i>	Pointer to the MAC information.

### 5.23.2.9.1.2 void(\* v2x\_per\_sps\_reservation\_calls\_t::v2x\_radio\_sps\_offset\_changed)(void \*context, v2x\_sps\_mac\_details\_t \*details)

Callback periodically made when the MAC SPS timeslot changes. The new reservation offset is in the details structure returned by pointer details.

#### Associated data types

[v2x\\_sps\\_mac\\_details\\_t](#)

#### Parameters

in	<i>measurements</i>	Pointer to the channel measurements.
in	<i>context</i>	Pointer to the context.

#### Detailed description

This callback can occur when a MAC contention triggers a new reservation time slot to be selected. It is relevant only to connections opened with [v2x\\_radio\\_tx\\_sps\\_sock\\_create\\_and\\_bind\(\)](#).

### 5.23.2.10 struct v2x\_tx\_flow\_info\_t

Advanced parameters that can be specified for Tx SPS and event-driven flows.

#### Data fields

Type	Field	Description
<a href="#">v2x_auto_retransmit_policy_t</a>	retransmit_policy	V2X retransmit policy.
uint8_t	default_tx_power_valid	Indicates whether the default Tx power is specified. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Default power is not specified</li> <li>• 1 – Default power is specified and is valid</li> </ul>
int32_t	default_tx_power	Default power used for transmission.
uint8_t	mcs_index_valid	Indicates whether the MCS index is specified. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Index is not specified</li> <li>• 1 – Index is specified and is valid</li> </ul>
uint8_t	mcs_index	MCS index number
uint8_t	tx_pool_id_valid	Indicates whether the Tx pool ID is valid. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – ID is not specified</li> <li>• 1 – ID is specified and is valid</li> </ul>
uint8_t	tx_pool_id	ID of the Tx pool.
int8_t	is_unicast_valid	Indicates whether is_unicast is specified. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Is unicast is not specified</li> <li>• 1 – Is unicast is specified and is valid</li> </ul>
uint8_t	is_unicast	Non zero if requested flow is unicast. Note: Unicast flows ignore subscribed Service Ids

### 5.23.2.11 struct v2x\_sock\_info\_t

Parameters to identify a Tx or Rx socket.

#### Data fields

Type	Field	Description
int	sock	Pointer to the file descriptor for the socket.
struct sockaddr_in6	sockaddr	IPv6 socket address. The sockaddr_in6 buffer is initialized with the IPv6 source address and source port that are used for the bind() function.

### 5.23.2.12 struct v2x\_sid\_list\_t

Parameters to identify a service ID list.

#### Data fields

Type	Field	Description
int	length	number of services IDs included in the array of sid.
uint32_t	sid[ <a href="#">MAX_S↔ UBSCRIBE_↔ SIDS_LIST_↔ LEN</a> ]	array of service IDs.

### 5.23.2.13 struct v2x\_tx\_sps\_flow\_info\_t

Advanced parameters that can be specified for Tx SPS flows.

#### Data fields

Type	Field	Description
<a href="#">v2x_tx_↔ bandwidth_↔ _reservation_t</a>	reservation	Transmit reservation information.
<a href="#">v2x_tx_flow_↔ info_t</a>	flow_info	Transmit resource information about the SPS Tx flow.

### 5.23.2.14 struct socket\_info\_t

Parameters that can be specified for the creation of CV2X socket.

#### Data fields

Type	Field	Description
uint32_t	service_id	V2X service ID bound to the CV2X socket.
uint16_t	local_port	Local port number of the CV2X socket used for binding.

## 5.23.3 Enumeration Type Documentation

### 5.23.3.1 enum v2x\_concurrency\_sel\_t

Describes whether the radio chip modem should attempt or support concurrent 3GPP CV2X operation with a WWAN 4G/5G data call.

Some chips are only capable of operating on CV2X. In this case:

- The callers can only request V2X\_WWAN\_NONCONCURRENT.
- The interface parameters that are returned list v2x\_concurrency\_sel\_t as the value for V2X\_WAN\_NONCONCURRENT.

**Enumerator**

**V2X\_WWAN\_NONCONCURRENT** No simultaneous WWAN + CV2X on this interface.

**V2X\_WWAN\_CONCURRENT** Interface allows requests for concurrent support of WWAN + CV2X connections.

**5.23.3.2 enum v2x\_event\_t**

Event indications sent asynchronously from the radio via callbacks that indicate the state of the radio. The state can change in response to the loss of timing precision or a geofencing change.

**Enumerator**

**V2X\_INACTIVE** V2X communication is disabled.

**V2X\_ACTIVE** V2X communication is enabled. Transmit and receive are possible.

**V2X\_TX\_SUSPENDED** Small loss of timing precision occurred. Transmit is no longer supported.

**V2X\_RX\_SUSPENDED** Radio can no longer receive any messages.

**V2X\_TXRX\_SUSPENDED** Radio can no longer transmit or receive for some reason.

### 5.23.3.3 enum v2x\_priority\_et

Range of supported priority levels, where a lower number means a higher priority. For example, 8 is the current 3GPP standard.

#### Enumerator

**V2X\_PRIO\_MOST\_URGENT** Highest priority.  
**V2X\_PRIO\_1**  
**V2X\_PRIO\_2**  
**V2X\_PRIO\_3**  
**V2X\_PRIO\_4**  
**V2X\_PRIO\_5**  
**V2X\_PRIO\_6**  
**V2X\_PRIO\_BACKGROUND** Lowest priority.

### 5.23.3.4 enum v2x\_service\_status\_t

Valid service availability states.

#### Enumerator

**SERVICE\_UNAVAILABLE**  
**SERVICE\_AVAILABLE**

### 5.23.3.5 enum v2x\_auto\_retransmit\_policy\_t

V2X Tx retransmission policies supported by the modem.

#### Enumerator

**V2X\_AUTO\_RETRANSMIT\_DISABLED** Retransmit mode is disabled.  
**V2X\_AUTO\_RETRANSMIT\_ENABLED** Retransmit mode is enabled.  
**V2X\_AUTO\_RETRANSMIT\_DONT\_CARE** Modem falls back to its default behavior.

### 5.23.3.6 enum traffic\_ip\_type\_t

V2X Ip Types

#### Enumerator

**TRAFFIC\_IP** Use Ip type traffic.  
**TRAFFIC\_NON\_IP** Use Non-Ip type traffic.

## 5.23.4 Function Documentation

### 5.23.4.1 `uint16_t v2x_convert_priority_to_traffic_class ( v2x_priority_et priority )`

Converts a traffic priority to one of the 255 IPv6 traffic class bytes that are used in the data plane to indicate per-packet priority on non-SPS (event driven) data ports.

#### Associated data types

[v2x\\_priority\\_et](#)

#### Parameters

in	<i>priority</i>	Packet priority that is to be converted to an IPv6 traffic class. This priority is between the lowest and highest priority values returned in <a href="#">v2x_iface_capabilities_t</a> .
----	-----------------	--

#### Detailed description

This function is symmetric and is a reverse operation.

The traffic priority is one of the values between `min_priority_value` and `max_priority_value` returned in the [v2x\\_iface\\_capabilities\\_t](#) query.

#### Returns

IPv6 traffic class for achieving the calling input parameter priority level.



### 5.23.4.2 `v2x_priority_et v2x_convert_traffic_class_to_priority ( uint16_t traffic_class )`

Maps an IPv6 traffic class to a V2X priority value.

#### Parameters

in	<i>traffic_class</i>	IPv6 traffic classification that came in a packet from the radio.
----	----------------------	---

#### Detailed description

This function is the inverse of the [v2x\\_convert\\_priority\\_to\\_traffic\\_class\(\)](#) function. It is symmetric and is a reverse operation.

#### Returns

Priority level (between highest and lowest priority values) equivalent to the input IPv6 traffic class parameter.

### 5.23.4.3 `v2x_api_ver_t v2x_radio_api_version ( )`

Method used to query the platform SDK for its version number, build information, and build date.

#### Returns

[v2x\\_api\\_ver\\_t](#) – Contains the build date and API version number.

#### 5.23.4.4 `v2x_status_enum_type v2x_radio_query_parameters ( const char * iface_name, v2x_iface_capabilities_t * caps )`

Gets the capabilities of a specific Radio interface attached to the system.

##### Associated data types

[v2x\\_iface\\_capabilities\\_t](#)

##### Parameters

in	<i>iface_name</i>	Pointer to the Radio interface name. The Radio interface is one of the following: <ul style="list-style-type: none"> <li>• An RmNet interface (HLOS)</li> <li>• The interface supplied for IP communication</li> <li>• The interface for non-IP communication (such as WSMP and Geonetworking).</li> </ul>
out	<i>caps</i>	Pointer to the <a href="#">v2x_iface_capabilities_t</a> structure, which contains the capabilities of this specific interface.

##### Returns

[V2X\\_STATUS\\_SUCCESS](#) – The radio is ready for data-plane sockets to be created and bound.

Error code – If there is a problem (see [v2x\\_status\\_enum\\_type](#)).

##### Dependencies

An SPS flow must have been successfully initialized.

### 5.23.4.5 v2x\_status\_enum\_type v2x\_radio\_deinit ( v2x\_radio\_handle\_t *handle* )

De-initializes a specific Radio interface.

#### Associated data types

[v2x\\_radio\\_handle\\_t](#)

#### Parameters

in	<i>handle</i>	Handle to the Radio that was initialized.
----	---------------	---

#### Returns

Indication of success or failure (see [v2x\\_status\\_enum\\_type](#)).

#### Dependencies

The interface must be pre-initialized with [v2x\\_radio\\_init\(\)](#) or [v2x\\_radio\\_init\\_v2\(\)](#). The handle from that function must be used as the parameter in this function.

#### 5.23.4.6 `int v2x_radio_rx_sock_create_and_bind ( v2x_radio_handle_t handle, int * sock, struct sockaddr_in6 * rx_sockaddr )`

Opens a new V2X radio receive socket, and initializes the given sockaddr buffer. The socket is also bound as an AF\_INET6 UDP type socket.

#### Associated data types

[v2x\\_radio\\_handle\\_t](#)

#### Parameters

in	<i>handle</i>	Identifies the initialized Radio interface.
out	<i>sock</i>	Pointer to the socket that, on success, returns the socket descriptor. The caller must release this socket with <a href="#">v2x_radio_sock_close()</a> .
out	<i>rx_sockaddr</i>	Pointer to the IPv6 UDP socket. The sockaddr_in6 buffer is initialized with the IPv6 source address and source port that are used for the bind.

#### Detailed description

You can execute any sockopts that are appropriate for this type of socket (AF\_INET6).

The port number for the receive path is not exposed, but it is in the sockaddr\_ll structure (if the caller is interested).

#### Returns

0 – On success.

Otherwise:

- EPERM – Socket creation failed; for more details, check errno.h.
- EAFNOSUPPORT – On failure to find the interface.
- EACCES – On failure to get the MAC address of the device.

#### Dependencies

The interface must be pre-initialized with `v2x_radio_init()` or `v2x_radio_init_v2()`. The handle from that function must be used as the parameter in this function.

### 5.23.4.7 `int v2x_radio_rx_sock_create_and_bind_v2 ( v2x_radio_handle_t handle, int id_ist_len, uint32_t * id_list, int * sock, struct sockaddr_in6 * rx_sockaddr )`

Opens a new V2X radio receive socket with specific service IDs for subscription, and initializes the given sockaddr buffer. The socket is also bound as an AF\_INET6 UDP type socket.

#### Associated data types

[v2x\\_radio\\_handle\\_t](#)

#### Parameters

in	<i>handle</i>	Identifies the initialized Radio interface.
in	<i>id_ist_len</i>	Identifies the length of service ID list.
in	<i>id_list</i>	Pointer to the service ID list for subscription, subscribe wildcard if input nullptr.
out	<i>sock</i>	Pointer to the socket that, on success, returns the socket descriptor. The caller must release this socket with <a href="#">v2x_radio_sock_close()</a> .
out	<i>rx_sockaddr</i>	Pointer to the IPv6 UDP socket. The sockaddr_in6 buffer is initialized with the IPv6 source address and source port that are used for the bind.

#### Detailed description

You can execute any sockopts that are appropriate for this type of socket (AF\_INET6).

This API can be used to subscribe wildcard, catchall port, or specific service IDs. The Rx port should be set with `v2x_set_rx_port()` before any subscription via this API, otherwise a default port number will be used.

Wildcard is used to receive all traffic. Only one port can be registered as wildcard port. Once wildcard is registered successfully, all received packets will be directed to wildcard port, and any subscription for specific service IDs or catchall port at other ports will be invalid. The parameter `id_list` of this API should be set to a null list for wildcard subscription.

Catchall port is used to receive packets with non-registered service IDs (via specific service IDs subscription). Only one port can be registered as catchall port. If catchall port is registered successfully, received packets with non-registered service ID will be directed to catchall port. All specific service IDs subscription (if any) should be performed before catchall port subscription. The parameter `id_list` of this API should include all non-registered service IDs for catchall port subscription.

Any port different from catchall port can be used to receive packets with specific service IDs. Only one port can be registered for a single service ID, a list of service IDs can be registered at a single port. To subscribe specific service IDs at a given Rx port, a Tx flow must be pre-setup with the Tx service ID set to any service ID included in the list of specific service IDs and the Tx source port set to the same port number as Rx port. The parameter `id_list` of this API should include all interested service IDs for the given Rx port.

## Returns

0 – On success.

Otherwise:

- EPERM – Socket creation failed; for more details, check `errno.h`.
- EAFNOSUPPORT – On failure to find the interface.
- EACCES – On failure to get the MAC address of the device.

## Dependencies

The interface must be pre-initialized with `v2x_radio_init()`. The handle from that function must be used as the parameter in this function. The Rx port must be pre-set with `v2x_set_rx_port()`, otherwise a default port number will be used. For any specific service ID subscription, a Tx flow must be pre-setup using one of the following methods:

- [v2x\\_radio\\_tx\\_sps\\_sock\\_create\\_and\\_bind\(\)](#)
- [v2x\\_radio\\_tx\\_sps\\_sock\\_create\\_and\\_bind\\_v2\(\)](#)
- [v2x\\_radio\\_tx\\_sps\\_only\\_create\(\)](#)
- [v2x\\_radio\\_tx\\_sps\\_only\\_create\\_v2\(\)](#)
- [v2x\\_radio\\_tx\\_event\\_sock\\_create\\_and\\_bind\(\)](#)
- [v2x\\_radio\\_tx\\_event\\_sock\\_create\\_and\\_bind\\_v2\(\)](#)
- [v2x\\_radio\\_tx\\_event\\_sock\\_create\\_and\\_bind\\_v3\(\)](#)

#### 5.23.4.8 `int v2x_radio_rx_sock_create_and_bind_v3 ( v2x_radio_handle_t handle, uint16_t port_num, int id_ist_len, uint32_t * id_list, int * sock, struct sockaddr_in6 * rx_sockaddr )`

Opens a new V2X radio receive socket with specific service IDs for subscription and specific port number for the receive path, and initializes the given sockaddr buffer. The socket is also bound as an AF\_INET6 UDP type socket.

This `v2x_radio_rx_sock_create_and_bind_v3()` method differs from `v2x_radio_rx_sock_create_and_bind_v2()` in that you can use the `port_num` parameter to specify the port number for the receive path.

#### Associated data types

[v2x\\_radio\\_handle\\_t](#)

#### Parameters

in	<i>handle</i>	Identifies the initialized Radio interface.
in	<i>port_num</i>	Identifies the port number for the receive path.
in	<i>id_ist_len</i>	Identifies the length of service ID list.
in	<i>id_list</i>	Pointer to the service ID list for subscription, subscribe wildcard if input nullptr.
out	<i>sock</i>	Pointer to the socket that, on success, returns the socket descriptor. The caller must release this socket with <a href="#">v2x_radio_sock_close()</a> .
out	<i>rx_sockaddr</i>	Pointer to the IPv6 UDP socket. The <code>sockaddr_in6</code> buffer is initialized with the IPv6 source address and source port that are used for the bind.

#### Detailed description

You can execute any sockets that are appropriate for this type of socket (AF\_INET6).

This API can be used to subscribe wildcard, catchall port, or specific service IDs.

Wildcard is used to receive all traffic. Only one port can be registered as wildcard port. Once wildcard is registered successfully, all received packets will be directed to wildcard port, and any subscription for specific service IDs or catchall port at other ports will be invalid. The parameter `id_list` of this API should be set to a null list for wildcard subscription.

Catchall port is used to receive packets with non-registered service IDs (via specific service IDs subscription). Only one port can be registered as catchall port. If catchall port is registered successfully, received packets with non-registered service ID will be directed to catchall port. All specific service IDs subscription (if any) should be performed before catchall port subscription. The parameter `id_list` of this API should include all non-registered service IDs for catchall port subscription.

Any port different from catchall port can be used to receive packets with specific service IDs. Only one port can be registered for a single service ID, a list of service IDs can be registered at a single port. To subscribe specific service IDs at a given Rx port, a Tx flow must be pre-setup with the Tx service ID set to any service ID included in the list of specific service IDs and the Tx source port set to the same

port number as Rx port. The parameter `id_list` of this API should include all interested service IDs for the given Rx port.

### Returns

0 – On success.

Otherwise:

- EPERM – Socket creation failed; for more details, check `errno.h`.
- EAFNOSUPPORT – On failure to find the interface.
- EACCES – On failure to get the MAC address of the device.

### Dependencies

The interface must be pre-initialized with `v2x_radio_init()`. The handle from that function must be used as the parameter in this function. For any specific service ID subscription, a Tx flow must be pre-setup using one of the following methods:

- [v2x\\_radio\\_tx\\_sps\\_sock\\_create\\_and\\_bind\(\)](#)
- [v2x\\_radio\\_tx\\_sps\\_sock\\_create\\_and\\_bind\\_v2\(\)](#)
- [v2x\\_radio\\_tx\\_sps\\_only\\_create\(\)](#)
- [v2x\\_radio\\_tx\\_sps\\_only\\_create\\_v2\(\)](#)
- [v2x\\_radio\\_tx\\_event\\_sock\\_create\\_and\\_bind\(\)](#)
- [v2x\\_radio\\_tx\\_event\\_sock\\_create\\_and\\_bind\\_v2\(\)](#)
- [v2x\\_radio\\_tx\\_event\\_sock\\_create\\_and\\_bind\\_v3\(\)](#)



**5.23.4.9** `int v2x_radio_sock_create_and_bind ( v2x_radio_handle_t handle, v2x_tx_sps_flow_info_t * tx_flow_info, v2x_per_sps_reservation_calls_t * calls, int tx_sps_portnum, int tx_event_portnum, int rx_portnum, v2x_sid_list_t * rx_id_list, v2x_sock_info_t * tx_sps_sock, v2x_sock_info_t * tx_event_sock, v2x_sock_info_t * rx_sock )`

Creates Tx SPS socket, Tx Event socket and Rx socket with specified parameters. The socket is also bound as an AF\_INET6 UDP type socket.

This `v2x_radio_sock_create_and_bind()` method is the combination of function `v2x_radio_tx_sps_sock_create_and_bind_v2()`/`v2x_radio_tx_event_sock_create_and_bind_v2` in the transmit direction and function `v2x_radio_rx_sock_create_and_bind_v3()` in the receiving direction.

#### Associated data types

[v2x\\_radio\\_handle\\_t](#)

#### Parameters

in	<i>handle</i>	Identifies the initialized Radio interface.
in	<i>tx_flow_info</i>	Pointer to the Tx SPS or event flow information. To create event flow, set reservation.v2xid and flow_info in this structure.
in	<i>calls</i>	Pointer to reservation callbacks or listeners. This parameter is called when underlying radio MAC parameters change related to the SPS bandwidth contract. For example, the callback after a reservation change, or if the timing offset of the SPS adjusts itself in response to traffic. This parameter passes NULL if no callbacks are required.
in	<i>tx_sps_portnum</i>	Requested Tx source port number for SPS transmissions, or -1 for no Tx sps flow.
in	<i>tx_event_portnum</i>	Requested Tx source port number for event transmissions, or -1 for no Tx event flow.
in	<i>rx_portnum</i>	Requested Rx destination port number, or -1 for no Rx subscription.
in	<i>rx_id_list</i>	Pointer to the Rx service ID list for subscription, subscribe wildcard if input nullptr.
out	<i>tx_sps_sock</i>	Pointer to the Tx sps socket that, on success, returns the socket descriptor and the IPv6 socket address. The caller must release this socket with <code>v2x_radio_sock_close()</code> .
out	<i>tx_event_sock</i>	Pointer to the Tx event socket that, on success, returns the socket descriptor and the IPv6 socket address. The caller must release this socket with <code>v2x_radio_sock_close()</code> .
out	<i>rx_sock</i>	Pointer to the Rx socket that, on success, returns the socket descriptor and the IPv6 socket address. The caller must release this socket with <code>v2x_radio_sock_close()</code> .

#### Detailed description

You can execute any sockopts that are appropriate for this type of socket (AF\_INET6).

This API can be used for the registration of both Tx and Rx. It sets up sockets on the requested port numbers. A negative port number corresponds to no actions for Tx or Rx.

Wildcard is used to receive all traffic. Only one port can be registered as wildcard port. Once wildcard is registered successfully, all received packets will be directed to wildcard port, and any subscription for specific service IDs or catchall port at other ports will be invalid. The parameter `rx_id_list` of this API should be set to a null list for wildcard subscription.

Catchall port is used to receive packets with non-registered service IDs (via specific service IDs subscription). Only one port can be registered as catchall port. If catchall port is registered successfully, received packets with non-registered service ID will be directed to catchall port. All specific service IDs subscription (if any) should be performed before catchall port subscription. The parameter `rx_id_list` of this API should include all non-registered service IDs for catchall port subscription.

Any port different from catchall port can be used to receive packets with specific service IDs. Only one port can be registered for a single service ID, a list of service IDs can be registered at a single port. To subscribe specific service IDs at a given Rx port, a Tx flow should also be set up using this API. The parameter `rx_id_list` should include all interested service IDs for the given Rx port, the parameter `tx_flow_info.reservation.v2xid` should be set to one of the service ID included in `rx_id_list`, the parameter `tx_sps_portnum` or `tx_event_portnum` should be set to the same port number as `rx_portnum`.

## Returns

0 – On success.

Otherwise:

- EPERM – Socket creation failed; for more details, check `errno.h`.
- EAFNOSUPPORT – On failure to find the interface.
- EACCES – On failure to get the MAC address of the device.

## Dependencies

The interface must be pre-initialized with `v2x_radio_init()`. The handle from that function must be used as the parameter in this function.

**5.23.4.10** `int v2x_radio_tx_sps_sock_create_and_bind ( v2x_radio_handle_t handle, v2x_tx_bandwidth_reservation_t * res, v2x_per_sps_reservation_calls_t * calls, int sps_portnum, int event_portnum, int * sps_sock, struct sockaddr_in6 * sps_sockaddr, int * event_sock, struct sockaddr_in6 * event_sockaddr )`

Creates and binds a socket with a bandwidth-reserved (SPS) Tx flow with the requested ID, priority, periodicity, and size on a specified source port number. The socket is created as an IPv6 UDP socket.

#### Associated data types

[v2x\\_radio\\_handle\\_t](#)  
[v2x\\_tx\\_bandwidth\\_reservation\\_t](#)  
[v2x\\_per\\_sps\\_reservation\\_calls\\_t](#)

#### Parameters

in	<i>handle</i>	Identifies the initialized Radio interface on which this data connection is made.
in	<i>res</i>	Pointer to the parameter structure (how often the structure is sent, how many bytes are reserved, and so on).
in	<i>calls</i>	Pointer to reservation callbacks or listeners. This parameter is called when underlying radio MAC parameters change related to the SPS bandwidth contract. For example, the callback after a reservation change, or if the timing offset of the SPS adjusts itself in response to traffic. This parameter passes NULL if no callbacks are required.
in	<i>sps_portnum</i>	Requested source port number for the bandwidth reserved SPS transmissions.
in	<i>event_portnum</i>	Requested source port number for the bandwidth reserved event transmissions, or -1 for no event port.
out	<i>sps_sock</i>	Pointer to the socket that is bound to the requested port for Tx with reserved bandwidth.
out	<i>sps_sockaddr</i>	Pointer to the IPv6 UDP socket. The sockaddr_in6 buffer is initialized with the IPv6 source address and source port that are used for the bind() function. The caller can then use the buffer for subsequent sendto() function calls.
out	<i>event_sock</i>	Pointer to the socket that is bound to the event-driven transmission port.
out	<i>event_sockaddr</i>	Pointer to the IPV6 UDP socket. The sockaddr_in6 buffer is initialized with the IPv6 source address and source port that are used for the bind() function. The caller can then use the buffer for subsequent sendto() function calls.

## Detailed description

The radio attempts to reserve the flow with the specified size and rate passed in the request parameters.

This function is used only for Tx. It sets up two UDP sockets on the requested two HLOS port numbers.

For only a single SPS flow, indicate the event port number by using a negative number or NULL for the event\_sockaddr. For a single event-driven port, use [v2x\\_radio\\_tx\\_event\\_sock\\_create\\_and\\_bind\(\)](#) instead.

Because the modem endpoint requires a specific global address, all data sent on these sockets must have a configurable IPv6 destination address for the non-IP traffic.

The Priority parameter of the SPS reservation is used only for the reserved Tx bandwidth (SPS) flow. The non-SPS/event-driven data sent to the event\_portnum parameter is prioritized on the air, based on the IPv6 Traffic Class of the packet.

The 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.

This call is a blocking call. When it returns, the sockets are ready to use, assuming there is no error.

## Returns

0 – On success.

Otherwise:

- EPERM – Socket creation failed; for more details, check errno.h.
- EAFNOSUPPORT – On failure to find the interface.
- EACCES – On failure to get the MAC address of the device.

## Dependencies

The interface must be pre-initialized with [v2x\\_radio\\_init\(\)](#) or [v2x\\_radio\\_init\\_v2\(\)](#). The handle from that function must be used as the parameter in this function.

### 5.23.4.11 `int v2x_radio_tx_sps_only_create ( v2x_radio_handle_t handle, v2x_tx_↔ bandwidth_reservation_t * res, v2x_per_sps_reservation_calls_t * calls, int sps_portnum, int * sps_sock, struct sockaddr_in6 * sps_sockaddr )`

Creates a socket with a bandwidth-reserved (SPS) Tx flow.

Only SPS transmissions are to be implemented for the socket, which is created as an IPv6 UDP socket.

#### Associated data types

[v2x\\_radio\\_handle\\_t](#)  
[v2x\\_tx\\_bandwidth\\_reservation\\_t](#)  
[v2x\\_per\\_sps\\_reservation\\_calls\\_t](#)

#### Parameters

in	<i>handle</i>	Identifies the initialized Radio interface on which this data connection is made.
in	<i>res</i>	Pointer to the parameter structure (how often the structure is sent, how many bytes are reserved, and so on).
in	<i>calls</i>	Pointer to reservation callbacks or listeners. This parameter is called when underlying radio MAC parameters change related to the SPS bandwidth contract. For example, the callback after a reservation change, or if the timing offset of the SPS adjusts itself in response to traffic. This parameter passes NULL if no callbacks are required.
in	<i>sps_portnum</i>	Requested source port number for the bandwidth reserved SPS transmissions.
out	<i>sps_sock</i>	Pointer to the socket that is bound to the requested port for Tx with reserved bandwidth.
out	<i>sps_sockaddr</i>	Pointer to the IPv6 UDP socket. The <code>sockaddr_in6</code> buffer is initialized with the IPv6 source address and source port that are used for the <code>bind()</code> function. The caller can then use the buffer for subsequent <code>sendto()</code> function calls.

#### Detailed description

The radio attempts to reserve the flow with the specified size and rate passed in the request parameters.

This function is used only for Tx. It sets up a UDP socket on the requested HLOS port number.

Because the modem endpoint requires a specific global address, all data sent on the socket must have a configurable IPv6 destination address for the non-IP traffic.

The Priority parameter of the SPS reservation is used only for the reserved Tx bandwidth (SPS) flow.

The caller is expected to identify an unused local port number for the SPS flow.

This call is a blocking call. When it returns, the socket is ready to use, assuming there is no error.

**Returns**

0 – On success.

Otherwise:

- EPERM – Socket creation failed; for more details, check `errno.h`.
- EINVAL – On failure to find the interface or get bad parameters.

**Dependencies**

The interface must be pre-initialized with `v2x_radio_init()` or `v2x_radio_init_v2()`. The handle from that function must be used as the parameter in this function.

### 5.23.4.12 `v2x_status_enum_type v2x_radio_tx_reservation_change ( int * sps_sock, v2x_tx_bandwidth_reservation_t * updated_reservation )`

Adjusts the reservation for transmit bandwidth.

#### Associated data types

[v2x\\_tx\\_bandwidth\\_reservation\\_t](#)

#### Parameters

out	<i>sps_sock</i>	Pointer to the socket bound to the requested port.
in	<i>updated_reservation</i>	Pointer to a bandwidth reservation with new reservation information.

#### Detailed description

This function will not update reservation priority. Can be used as follows:

- When the bandwidth requirement changes in periodicity (for example, due to an application layer DCC algorithm)
- Because the packet size is increasing (for example, due to a growing path history size in a BSM).

When the reservation change is complete, a callback to the structure is passed in a `v2x_radio_init()` or `v2x_radio_init_v2()` call.

#### Returns

[V2X\\_STATUS\\_SUCCESS](#).

Error code – If there is a problem (see [v2x\\_status\\_enum\\_type](#)).

#### Dependencies

An SPS flow must have been successfully initialized with the `v2x_radio_tx_sps_sock_create_and_bind()` method.

### 5.23.4.13 `int v2x_radio_tx_event_sock_create_and_bind ( const char * interface, int v2x_id, int event_portnum, struct sockaddr_in6 * event_sock_addr, int * sock )`

Opens and binds an event-driven socket (one with no bandwidth reservation). The socket is bound as an AF\_INET6 UDP type socket.

#### Parameters

in	<i>interface</i>	Pointer to the operating system name to use. This interface is an RmNet interface (HLOS).
in	<i>v2x_id</i>	Used for transmissions that are ultimately mapped to an L2 destination address.
in	<i>event_portnum</i>	Local port number to which the socket is bound. Used for transmissions of this ID.
out	<i>event_sock_addr</i>	Pointer to the sockaddr_ll structure buffer to be initialized.
out	<i>sock</i>	Pointer to the file descriptor. Loaded when the function is successful.

#### Detailed description

This function is used only for Tx when no periodicity is available for the application type. If you know your transmit data periodicity, use [v2x\\_radio\\_tx\\_sps\\_sock\\_create\\_and\\_bind\(\)](#) instead.

These event-driven sockets pay attention to QoS parameters in the IP socket.

#### Returns

0 – On success.

Otherwise:

- EPERM – Socket creation failed; for more details, check `errno.h`.
- EAFNOSUPPORT – On failure to find the interface.
- EACCES – On failure to get the MAC address of the device.



#### 5.23.4.14 `v2x_status_enum_type v2x_radio_start_measurements ( v2x_radio_↔ _handle_t handle, v2x_chan_meas_params_t * measure_this_way )`

Requests a channel utilization (CBP/CBR) measurement result on a channel.

##### Associated data types

[v2x\\_radio\\_handle\\_t](#)

[v2x\\_chan\\_meas\\_params\\_t](#)

##### Parameters

in	<i>handle</i>	Handle to the port.
in	<i>measure_this_way</i>	Indicates how and what to measure, and how often to send results. Some higher-level standards (like J2945/1 and ETSI TS102687 DCC) have specific time windows and items to measure.

##### Detailed description

This function uses the callbacks passed in during initialization to deliver the measurements. Measurement callbacks continue until the Radio interface is closed.

##### Returns

[V2X\\_STATUS\\_SUCCESS](#) – The radio is now ready for data-plane sockets to be created and bound.

[V2X\\_STATUS\\_FAIL](#) – CBR measurement is not supported yet.

##### Dependencies

The interface must be pre-initialized with `v2x_radio_init()` or `v2x_radio_init_v2()`. The handle from that function must be used as the parameter in this function.

#### 5.23.4.15 `v2x_status_enum_type v2x_radio_stop_measurements ( v2x_radio_handle_t handle )`

Discontinues any periodic MAC/Phy channel measurements and the reporting of them via listener calls.

##### Associated data types

[v2x\\_radio\\_handle\\_t](#)

##### Parameters

in	<i>handle</i>	Handle to the radio measurements to be stopped.
----	---------------	---

##### Returns

[V2X\\_STATUS\\_SUCCESS](#).

##### Dependencies

The measurements must have been started with [v2x\\_radio\\_start\\_measurements\(\)](#).

### 5.23.4.16 int v2x\_radio\_sock\_close ( int \* sock\_fd )

Closes a specified socket file descriptor and deregisters any modem resources associated with it (such as reserved SPS bandwidth contracts).

#### Parameters

in	<i>sock_fd</i>	Socket file descriptor.
----	----------------	-------------------------

#### Detailed description

This function works on receive, SPS, or event-driven sockets.

The socket file descriptor must be closed when the client exits. We recommend using a trap to catch controlled shutdowns.

#### Returns

Integer value of the close(sock) operation.

#### Dependencies

The socket must have been opened with one of the following methods:

- [v2x\\_radio\\_rx\\_sock\\_create\\_and\\_bind\(\)](#)
- [v2x\\_radio\\_tx\\_sps\\_sock\\_create\\_and\\_bind\(\)](#)
- [v2x\\_radio\\_tx\\_sps\\_sock\\_create\\_and\\_bind\\_v2\(\)](#)
- [v2x\\_radio\\_tx\\_sps\\_only\\_create\(\)](#)
- [v2x\\_radio\\_tx\\_sps\\_only\\_create\\_v2\(\)](#)
- [v2x\\_radio\\_tx\\_event\\_sock\\_create\\_and\\_bind\(\)](#)
- [v2x\\_radio\\_tx\\_event\\_sock\\_create\\_and\\_bind\\_v2\(\)](#)
- [v2x\\_radio\\_tx\\_event\\_sock\\_create\\_and\\_bind\\_v3\(\)](#)

**5.23.4.17 void v2x\_radio\_set\_log\_level ( int *new\_level*, int *use\_syslog* )**

Configures the V2X log level and destination for SDK and lower layers.

**Parameters**

in	<i>new_level</i>	Log level to set to one of the standard syslog levels (LOG_ERR, LOG_INFO, and so on).
in	<i>use_syslog</i>	Destination: send to stdout (0) or syslog (otherwise).

**Returns**

None.

**5.23.4.18 v2x\_event\_t cv2x\_status\_poll ( uint64\_t \* *status\_age\_useconds* )**

Polls for the recent V2X status.

**Parameters**

out	<i>status_age_useconds</i>	Pointer to the age in microseconds of the last event (radio status) that is being reported.
-----	----------------------------	---

**Detailed description**

This function does not generate any modem control traffic. For efficiency, it simply returns the most recently cached value that was reported from the modem (often reported at a high rate or frequent rate from the modem).

**Returns**

Indication of success or failure (see [v2x\\_status\\_enum\\_type](#)).

### 5.23.4.19 int v2x\_radio\_trigger\_l2\_update ( v2x\_radio\_handle\_t *handle* )

Triggers the modem to change its source L2 address by randomly generating a new address.

#### Associated data types

[v2x\\_radio\\_handle\\_t](#)

#### Parameters

in	<i>handle</i>	Initialized Radio interface on which this data connection is made.
----	---------------	--

#### Detailed description

When the change is complete, clients are notified of the new L2 address via the [v2x\\_radio\\_calls\\_t::v2x\\_radio\\_l2\\_addr\\_changed\\_listener\(\)](#) callback function.

#### Returns

0 – On success.

Otherwise:

- EPERM – Socket creation failed; for more details, check errno.h.
- EAFNOSUPPORT – On failure to find the interface.
- EACCES – On failure to get the MAC address of the device.

**5.23.4.20** `int v2x_radio_update_trusted_ue_list ( unsigned int malicious_list_len, unsigned int malicious_list[MAX_MALICIOUS_IDS_LIST_LEN], unsigned int trusted_list_len, trusted_ue_info_t trusted_list[MAX_TRUSTED_IDS_LIST_LEN] )`

Updates the list of malicious and trusted IDs tracked by the modem.

#### Associated data types

[trusted\\_ue\\_info\\_t](#)

#### Parameters

in	<i>malicious_list_len</i>	Number of malicious IDs in <i>malicious_list</i> .
in	<i>malicious_list</i>	List of malicious IDs.
in	<i>trusted_list_len</i>	Number of trusted IDs in <i>trusted_list</i> .
in	<i>trusted_list</i>	List of trusted IDs.

#### Returns

0 – On success.

Otherwise:

- EPERM – Socket creation failed; for more details, check `errno.h`.
- EAFNOSUPPORT – On failure to find the interface.
- EACCES – On failure to get the MAC address of the device.

**5.23.4.21** `int v2x_radio_tx_sps_sock_create_and_bind_v2 ( v2x_radio_handle_t handle, v2x_tx_sps_flow_info_t * sps_flow_info, v2x_per_sps_reservation_calls_t * calls, int sps_portnum, int event_portnum, int * sps_sock, struct sockaddr_in6 * sps_sockaddr, int * event_sock, struct sockaddr_in6 * event_sockaddr )`

Creates and binds a socket with a bandwidth-reserved (SPS) Tx flow with the requested ID, priority, periodicity, and size on a specified source port number. The socket is created as an IPv6 UDP socket.

This `v2x_radio_tx_sps_sock_create_and_bind_v2()` method differs from `v2x_radio_tx_sps_sock_create_and_bind()` in that you can use the `sps_flow_info` parameter to specify transmission resource information about the Tx flow.

#### Associated data types

[v2x\\_radio\\_handle\\_t](#)  
[v2x\\_tx\\_sps\\_flow\\_info\\_t](#)  
[v2x\\_per\\_sps\\_reservation\\_calls\\_t](#)

#### Parameters

in	<i>handle</i>	Identifies the initialized Radio interface on which this data connection is made.
in	<i>sps_flow_info</i>	Pointer to the flow information in the <a href="#">v2x_tx_sps_flow_info_t</a> structure.
in	<i>calls</i>	Pointer to reservation callbacks or listeners. This parameter is called when underlying radio MAC parameters change related to the SPS bandwidth contract. For example, the callback after a reservation change, or if the timing offset of the SPS adjusts itself in response to traffic. This parameter passes NULL if no callbacks are required.
in	<i>sps_portnum</i>	Requested source port number for the bandwidth reserved SPS transmissions.
in	<i>event_portnum</i>	Requested source port number for the bandwidth reserved event transmissions, or -1 for no event port.
out	<i>sps_sock</i>	Pointer to the socket that is bound to the requested port for Tx with reserved bandwidth.
out	<i>sps_sockaddr</i>	Pointer to the IPv6 UDP socket. The <code>sockaddr_in6</code> buffer is initialized with the IPv6 source address and source port that are used for the <code>bind()</code> function. The caller can then use the buffer for subsequent <code>sendto()</code> function calls.
out	<i>event_sock</i>	Pointer to the socket that is bound to the event-driven transmission port.
out	<i>event_sockaddr</i>	Pointer to the IPV6 UDP socket. The <code>sockaddr_in6</code> buffer is initialized with the IPv6 source address and source port that are used for the <code>bind()</code> function. The caller can then use the buffer for subsequent <code>sendto()</code> function calls.

## Detailed description

The radio attempts to reserve the flow with the specified size and rate passed in the request parameters.

This function is used only for Tx. It sets up two UDP sockets on the requested two HLOS port numbers.

For only a single SPS flow, indicate the event port number by using a negative number or NULL for the event\_sockaddr. For a single event-driven port, use [v2x\\_radio\\_tx\\_event\\_sock\\_create\\_and\\_bind\(\)](#) or [v2x\\_radio\\_tx\\_event\\_sock\\_create\\_and\\_bind\\_v2\(\)](#) or [v2x\\_radio\\_tx\\_event\\_sock\\_create\\_and\\_bind\\_v3\(\)](#) instead.

Because the modem endpoint requires a specific global address, all data sent on these sockets must have a configurable IPv6 destination address for the non-IP traffic.

The Priority parameter of the SPS reservation is used only for the reserved Tx bandwidth (SPS) flow. The non-SPS/event-driven data sent to the event\_portnum parameter is prioritized on the air, based on the IPv6 Traffic Class of the packet.

The 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.

This call is a blocking call. When it returns, the sockets are ready to use, assuming there is no error.

## Returns

0 – On success.

Otherwise:

- EPERM – Socket creation failed; for more details, check errno.h.
- EAFNOSUPPORT – On failure to find the interface.
- EACCES – On failure to get the MAC address of the device.

## Dependencies

The interface must be pre-initialized with [v2x\\_radio\\_init\(\)](#) or [v2x\\_radio\\_init\\_v2\(\)](#). The handle from that function must be used as the parameter in this function.



**5.23.4.22** `int v2x_radio_tx_sps_only_create_v2 ( v2x_radio_handle_t handle, v2x_tx_sps_flow_info_t * sps_flow_info, v2x_per_sps_reservation_calls_t * calls, int sps_portnum, int * sps_sock, struct sockaddr_in6 * sps_sockaddr )`

Creates a socket with a bandwidth-reserved (SPS) Tx flow.

Only SPS transmissions are to be implemented for the socket, which is created as an IPv6 UDP socket.

This `v2x_radio_tx_sps_only_create_v2()` method differs from `v2x_radio_tx_sps_only_create()` in that you can use the `sps_flow_info` parameter to specify transmission resource information about the Tx flow.

#### Associated data types

[v2x\\_radio\\_handle\\_t](#)  
[v2x\\_tx\\_sps\\_flow\\_info\\_t](#)  
[v2x\\_per\\_sps\\_reservation\\_calls\\_t](#)

#### Parameters

in	<i>handle</i>	Identifies the initialized Radio interface on which this data connection is made.
in	<i>sps_flow_info</i>	Pointer to the flow information in the <a href="#">v2x_tx_sps_flow_info_t</a> structure.
in	<i>calls</i>	Pointer to reservation callbacks or listeners. This parameter is called when underlying radio MAC parameters change related to the SPS bandwidth contract. For example, the callback after a reservation change, or if the timing offset of the SPS adjusts itself in response to traffic. This parameter passes NULL if no callbacks are required.
in	<i>sps_portnum</i>	Requested source port number for the bandwidth reserved SPS transmissions.
out	<i>sps_sock</i>	Pointer to the socket that is bound to the requested port for Tx with reserved bandwidth.
out	<i>sps_sockaddr</i>	Pointer to the IPv6 UDP socket. The <code>sockaddr_in6</code> buffer is initialized with the IPv6 source address and source port that are used for the <code>bind()</code> function. The caller can then use the buffer for subsequent <code>sendto()</code> function calls.

#### Detailed description

The radio attempts to reserve the flow with the specified size and rate passed in the request parameters.

This function is used only for Tx. It sets up a UDP socket on the requested HLOS port number. Because the modem endpoint requires a specific global address, all data sent on the socket must have a configurable IPv6 destination address for the non-IP traffic.

The caller is expected to identify an unused local port number to use for binding the SPS flow.

This call is a blocking call. When it returns, the socket is ready to use, assuming there is no error.

**Returns**

0 – On success.

Otherwise:

- EPERM – Socket creation failed; for more details, check `errno.h`.
- EINVAL – On failure to find the interface or get bad parameters.

**Dependencies**

The interface must be pre-initialized with `v2x_radio_init()` or `v2x_radio_init_v2()`. The handle from that function must be used as the parameter in this function.

### 5.23.4.23 `v2x_status_enum_type v2x_radio_tx_reservation_change_v2 ( int * sps_sock, v2x_tx_sps_flow_info_t * updated_flow_info )`

Adjusts the reservation for transmit bandwidth.

This `v2x_radio_tx_reservation_change_v2()` method differs from `v2x_radio_tx_reservation_change()` in that you can use the `updated_flow_info` parameter to specify transmission resource information about the Tx flow.

#### Associated data types

[v2x\\_tx\\_sps\\_flow\\_info\\_t](#)

#### Parameters

out	<code>sps_sock</code>	Pointer to the socket bound to the requested port.
in	<code>updated_flow_info</code>	Pointer to the new reservation information.

#### Detailed description

This function will not update reservation priority. Can be used as follows:

- When the bandwidth requirement changes in periodicity (for example, due to an application layer DCC algorithm)
- Because the packet size is increasing (for example, due to a growing path history size in a BSM).

When the reservation change is complete, a callback to the structure is passed in a `v2x_radio_init()` or `v2x_radio_init_v2()` call.

#### Returns

[V2X\\_STATUS\\_SUCCESS](#).

Error code – On failure (see [v2x\\_status\\_enum\\_type](#)).

#### Dependencies

An SPS flow must have been successfully initialized with `v2x_radio_tx_sps_sock_create_and_bind()` or `v2x_radio_tx_sps_sock_create_and_bind_v2()`.

#### 5.23.4.24 `v2x_status_enum_type v2x_radio_tx_event_flow_info_change ( int * sock, v2x_tx_flow_info_t * updated_flow_info )`

Adjusts the flow parameters for an existing Tx event socket.

##### Associated data types

[v2x\\_tx\\_flow\\_info\\_t](#)

##### Parameters

out	<i>sock</i>	Pointer to the socket bound to the requested port.
in	<i>updated_flow_info</i>	Pointer to the new flow parameters.

##### Detailed description

When the reservation change is complete, a callback to the structure is passed in a `v2x_radio_init()` or `v2x_radio_init_v2()` call.

This call is a blocking call. When it returns, the socket is ready to be use, assuming there is no error.

##### Returns

`V2X_STATUS_SUCCESS`.

Error code – On failure (see `v2x_status_enum_type`).

##### Dependencies

An event flow must have been successfully initialized with `v2x_radio_tx_event_sock_create_and_bind()` or `v2x_radio_tx_event_sock_create_and_bind_v2()` or `v2x_radio_tx_event_sock_create_and_bind_v3()`.

### 5.23.4.25 `v2x_status_enum_type v2x_update_configuration ( const char * config_file_path )`

Updates the V2X radio configuration file.

#### Parameters

in	<code>config_file_path</code>	Pointer to the path of the configuration file.
----	-------------------------------	--

#### Detailed description

The V2X radio status must be INACTIVE. If the V2X status is ACTIVE or SUSPENDED (see [v2x\\_event\\_t](#)), call [stop\\_v2x\\_mode\(\)](#) first.

This call is a blocking call. When it returns the configuration has been updated, assuming no error.

#### Returns

[V2X\\_STATUS\\_SUCCESS](#).

Otherwise:

- [V2X\\_STATUS\\_EALREADY](#) – Failure because V2X status is not [V2X\\_INACTIVE](#).
- [V2X\\_STATUS\\_FAIL](#) – Other failure.

#### Dependencies

V2X radio status must be [V2X\\_INACTIVE](#) ([v2x\\_event\\_t](#)).

#### 5.23.4.26 `v2x_status_enum_type start_v2x_mode ( )`

Starts V2X mode.

The V2X radio status must be `INACTIVE`. If the status is `ACTIVE` or `SUSPENDED` (see `v2x_event_t`), call `stop_v2x_mode()` first.

This call is a blocking call. When it returns, V2X mode has been started, assuming there is no error.

##### Returns

`V2X_STATUS_SUCCESS`.

Otherwise:

- `V2X_STATUS_EALREADY` – Failure because V2X mode is already started.
- `V2X_STATUS_FAIL` – Other failure.

##### Dependencies

V2X radio status must be `V2X_INACTIVE` (`v2x_event_t`).

#### 5.23.4.27 `v2x_status_enum_type stop_v2x_mode ( )`

Stops V2X mode.

The V2X radio status must be `ACTIVE` or `SUSPENDED` (see `v2x_event_t`). If the status is `INACTIVE`, call `start_v2x_mode()` first.

This call is a blocking call. When it returns, V2X mode has been stopped, assuming there is no error.

##### Returns

`V2X_STATUS_SUCCESS`.

Otherwise:

- `V2X_STATUS_EALREADY` – Failure because V2X mode is already stopped.
- `V2X_STATUS_FAIL` – Other failure.

##### Dependencies

V2X radio status must be `V2X_ACTIVE`, `V2X_TX_SUSPENDED`, `V2X_RX_SUSPENDED`, or `V2X_TXRX_SUSPENDED`.

**5.23.4.28** `v2x_radio_handle_t v2x_radio_init_v2 ( traffic_ip_type_t ip_type, v2x_concurrency_sel_t mode, v2x_radio_calls_t * callbacks_p, void * ctx_p )`

Initializes the Radio interface and sets the callback that will be used when events in the radio change (including when radio initialization is complete).

#### Associated data types

[traffic\\_ip\\_type\\_t](#)  
[v2x\\_concurrency\\_sel\\_t](#)  
[v2x\\_radio\\_calls\\_t](#)

#### Parameters

in	<i>ip_type</i>	The Ip or non-IP interface.
in	<i>mode</i>	WAN concurrency mode, although the radio might not support concurrency. Errors can be generated.
in	<i>callbacks</i>	Pointer to the <a href="#">v2x_radio_calls_t</a> structure that is prepopulated with function pointers used during radio events (such as loss of time synchronization or accuracy) for subscribers. This parameter also points to a callback for this initialization function.
in	<i>context</i>	Voluntary pointer to the first parameter on the callback.

#### Detailed description

This function call is a nonblocking, and it is a control plane action.

Use [v2x\\_radio\\_deinit\(\)](#) when radio operations are complete.

Currently, the channel and transmit power are not specified. They are specified with a subsequent call to [v2x\\_radio\\_calls\\_t::v2x\\_radio\\_init\\_complete\(\)](#) when initialization is complete.

#### Returns

Handle to the specified initialized radio. The handle is used for reconfiguring, opening or changing, and closing reservations.

[V2X\\_RADIO\\_HANDLE\\_BAD](#) – If there is an error. No initialization callback is made.

#### 5.23.4.29 `int v2x_radio_tx_event_sock_create_and_bind_v3 ( traffic_ip_type_t ip_type, int v2x_id, int event_portnum, v2x_tx_flow_info_t * event_flow_info, struct sockaddr_in6 * event_sockaddr, int * sock )`

Opens and binds an event-driven socket (one with no bandwidth reservation). The socket is bound as an AF\_INET6 UDP type socket.

This `v2x_radio_tx_event_sock_create_and_bind_v3()` method differs from `v2x_radio_tx_event_sock_create_and_bind_v2()` in that you can use the `traffic_ip_type_t` parameter to specify traffic ip type instead of requiring the interface name.

#### Associated data types

[v2x\\_tx\\_flow\\_info\\_t](#) [traffic\\_ip\\_type\\_t](#)

#### Parameters

in	<i>ip_type</i>	<code>traffic_ip_type</code> .
in	<i>v2x_id</i>	Used for transmissions that are ultimately mapped to an L2 destination address.
in	<i>event_portnum</i>	Local port number to which the socket is bound. Used for transmissions of this ID.
in	<i>event_flow_info</i>	Pointer to the event flow parameters.
out	<i>event_sock_addr</i>	Pointer to the <code>sockaddr_ll</code> structure buffer to be initialized.
out	<i>sock</i>	Pointer to the file descriptor. Loaded when the function is successful.

#### Detailed description

This function is used only for Tx when no periodicity is available for the application type. If you know your transmit data periodicity, use `v2x_radio_tx_sps_sock_create_and_bind()` or `v2x_radio_tx_sps_sock_create_and_bind_v2()` instead.

These event-driven sockets pay attention to QoS parameters in the IP socket.

#### Returns

0 – On success.

Otherwise:

- EPERM – Socket creation failed; for more details, check `errno.h`.
- EAFNOSUPPORT – On failure to find the interface.
- EACCES – On failure to get the MAC address of the device.



### 5.23.4.30 v2x\_status\_enum\_type get\_iface\_name ( traffic\_ip\_type\_t *ip\_type*, char \* *iface\_name*, size\_t *buffer\_len* )

Returns interface name set during radio initialization.

#### Associated data types

traffic\_ip\_type\_t

#### Parameters

in	<i>ip_type</i>	traffic_ip_type_t
out	<i>iface_name</i>	pointer to buffer for interface name
in	<i>buffer_len</i>	length of the buffer passed for interface name. Must be at least the max buffer size for an interface name (IFNAMSIZE).

#### Detailed description

This function should only be called after successfully initializing a radio.

#### Returns

[V2X\\_STATUS\\_SUCCESS](#).

[V2X\\_STATUS\\_FAIL](#) – If there is an error. Interface name will be an empty string.

### 5.23.4.31 `int v2x_radio_tcp_sock_create_and_bind ( v2x_radio_handle_t handle, const v2x_tx_flow_info_t * event_info, const socket_info_t * sock_info, int * sock_fd, struct sockaddr_in6 * sockaddr )`

Creates a TCP socket for event Tx and Rx. The socket is bound as an AF\_INET6 TCP type socket.

This `v2x_radio_tcp_sock_create_and_bind()` API creates a new TCP socket and binds the socket 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 to enable TCP control and data packets in both transmitting and receiving directions.

If the created socket is expected to work as TCP client mode, the caller must establish a connection to the address specified using function `connect()`, and then use the socket for `send()` and `recv()` on successful connection. The caller must release the created socket and associated resources with `v2x_radio_sock_close()`.

If the created socket is expected to work as TCP server mode, the caller must mark the created socket as a listening socket with function `listen()`, that is, as a socket that will be used to accept incoming connection requests using `accept()`. The caller can then use the connected socket returned by `accept()` for `send()` and `recv()`. The caller must close all connected sockets returned by `accept()` with function `close()` first, and then release the listening socket and associated resources with `v2x_radio_sock_close()`.

This call is a blocking call. When it returns, the created TCP socket is ready to use, assuming there is no error.

#### Associated data types

[v2x\\_radio\\_handle\\_t](#)

#### Parameters

in	<i>handle</i>	Identifies the initialized Radio interface. The caller must specify IP interface for radio initialization.
in	<i>event_info</i>	Pointer to the Tx event flow information.
in	<i>sock_info</i>	Pointer to the TCP socket information.
out	<i>sock_fd</i>	Pointer to the socket that, on success, returns the TCP socket descriptor. The caller must release this socket with <a href="#">v2x_radio_sock_close()</a> .
out	<i>sockaddr</i>	Pointer to the address of TCP socket. The <code>sockaddr_in6</code> buffer is initialized with the IPv6 source address and source port that are used for the bind.

#### Detailed description

You can execute any socketopts that are appropriate for this type of socket (AF\_INET6).

**Returns**

0 – On success.

Otherwise:

- EINVAL – On failure to find the interface or get bad parameters.
- EPERM – Socket operation failed; for more details, check errno.h.

**Dependencies**

The interface used for IP communication must be pre-initialized with `v2x_radio_init()`. The handle from that function must be used as the parameter in this function.

### 5.23.4.32 `v2x_status_enum_type v2x_set_global_IPaddr ( uint8_t prefix_len, uint8_t * ipv6_addr )`

Set CV2X global IP address for the IP interface.

#### Parameters

in	<i>prefix_len</i>	CV2X global IP address prefix length in bits, range [1, 128]
in	<i>ipv6_addr</i>	CV2X global IP address.

#### Returns

V2X\_STATUS\_SUCCESS on success. Error status otherwise.

### 5.23.4.33 `v2x_status_enum_type v2x_set_ip_routing_info ( uint8_t * dest_mac_addr )`

Set CV2X IP interface global IP unicast routing information.

#### Parameters

in	<i>dest_mac_addr</i>	CV2X destination L2 address for unicast routing purpose. expecting a 6 bytes array address, in which the L2 addr stored in the last 3 entries in big endian order.
----	----------------------	--

#### Returns

V2X\_STATUS\_SUCCESS on success. Error status otherwise.

## 5.24 V2x\_api\_vehicle

Abstraction of the vehicle system parameters required for CAM/BSM ITS beacons.

### 5.24.1 Define Documentation

#### 5.24.1.1 #define V2X\_VDATA\_HANDLE\_BAD (-1)

Invalid handle returned when there is an error.

#### 5.24.1.2 #define V2X\_J2735\_TRACTION\_CONTROL\_MAX (4)

Guard check value on [v2x\\_transmission\\_state\\_enum\\_type](#) for [V2X\\_TRANSMISSION\\_MAX](#). This value is used in a 3-bit bitfield in J2735.

#### 5.24.1.3 #define V2X\_TRACTION\_CTRL\_MAX (4)

Guard check value on [v2x\\_TractionControlStatus\\_enum\\_type](#) for [V2X\\_TRACTION\\_CTRL\\_MAX](#). This value is used in a 2-bit bitfield in J2735.

#### 5.24.1.4 #define J2735\_ABS\_MAX (4)

Guard check value on [v2x\\_AntiLockBrakeStatus\\_enum\\_type](#).

This value cannot be part of the enumeration because the value eventually ends up in 2-bit bitfield over the air.

#### 5.24.1.5 #define V2X\_STABILITY\_CONTROL\_MAX (4)

Guard check value on [v2x\\_StabilityControlStatus\\_enum\\_type](#).

This value is eventually used over the air in a 2-bit bitfield. The enumeration value must never be larger than 4.

#### 5.24.1.6 #define V2X\_AUX\_BRAKE\_MAX (4)

Guard check value on [v2x\\_AuxBrakeStatus\\_enum\\_type](#). This value must never be set this high.

## 5.24.2 Data Structure Documentation

### 5.24.2.1 union v2x\_control\_status\_ut

Contains information related to ABS, TCS, stability control state, and other vehicle output controls that might occur and be ongoing. This structure mirrors the J2735 bit frames.

#### Data fields

Type	Field	Description
struct <a href="#">v2x_control_status_ut</a>	bits	Bit values for control status information. Bit values for control status information.
unsigned short	word	Byte data access to the packed <a href="#">v2x_control_status</a> union structure.

### 5.24.2.2 struct v2x\_control\_status\_ut.bits

Bit values for control status information.

#### Data fields

Type	Field	Description
unsigned	unused_padding: 1	Reserved for 16-bit alignment. This field is critical because of 16-bit word access to the packed <a href="#">v2x_control_status_ut</a> union structure.
<a href="#">v2x_AuxBrakeStatus_enum_type</a>	aux_brake_status: 2	Indicates whether the auxiliary braking system is on. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Off</li> <li>• 1 – On</li> </ul>
<a href="#">v2x_BrakeBoostApplied_enum_type</a>	brake_boost_applied: 2	Indicates whether the brakes are actively being boosted. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not boosted</li> <li>• 1 – Boosted</li> </ul>
<a href="#">v2x_StabilityControlStatus_enum_type</a>	stability_control_status: 2	Indicates whether stability control is on and engaged. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Off</li> <li>• 1 – On</li> </ul>
<a href="#">v2x_AntiLockBrakeStatus_enum_type</a>	antilock_brake_status: 2	Indicates the status of the ABS.
<a href="#">v2x_TractionControlStatus_enum_type</a>	traction_control_status: 2	Indicates whether status of the TCS.

Type	Field	Description
unsigned	rightRear: 1	Indicates whether the right rear brakes are actively being applied. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not applied</li> <li>• 1 – Applied</li> </ul>
unsigned	rightFront: 1	Indicates whether the right front brakes are actively being applied. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not applied</li> <li>• 1 – Applied</li> </ul>
unsigned	leftRear: 1	Indicates whether the left rear brakes are actively being applied <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not applied</li> <li>• 1 – Applied</li> </ul>
unsigned	leftFront: 1	Indicates whether the front left brakes are actively being applied. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not applied</li> <li>• 1 – Applied</li> </ul>
unsigned	unavailable: 1	No information is available.

### 5.24.2.3 union vehicleEventFlags\_ut

Contains critical events and communication of ongoing events. Also is used for combinations of critical and not critical (wipers) events

This typedef can match the J2735 2016 version or another version you are working with.

#### Data fields

Type	Field	Description
struct <a href="#">vehicle↔ EventFlags_ut</a>	bits	Bit values for vehicle event flags. A flag indicates the state of the event.Bit values for vehicle event flags.
unsigned short	data	Sixteen-bit word access to the packed vehicleEventFlags union structure.

### 5.24.2.4 struct vehicleEventFlags\_ut.bits

Bit values for vehicle event flags. A flag indicates the state of the event.

#### Data fields

Type	Field	Description
unsigned	unused: 3	Reserved for 16-bit alignment in the union access.
unsigned	eventAirBag↔ Deployment: 1	Indicates whether the airbag is deployed. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not deployed</li> <li>• 1 – Deployed</li> </ul>
unsigned	event↔ Disabled↔ Vehicle: 1	Indicates whether the vehicle is disabled. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not disabled</li> <li>• 1 – Disabled</li> </ul>
unsigned	eventFlatTire: 1	Indicates whether the tire is flat. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not flat</li> <li>• 1 – Flat</li> </ul>
unsigned	eventWipers↔ Changed: 1	Indicates the status of the windshield wipers. For more information, See the wiper state variables in <a href="#">current_dynamic_vehicle_state_t</a> .
unsigned	eventLights↔ Changed: 1	Indicates the status of one or more lights (such as blinkers and fog).
unsigned	eventHard↔ Braking: 1	Indicates whether hard braking is activated. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not activated</li> <li>• 1 – Activated</li> </ul>
unsigned	event↔ Reserved1: 1	Event bit reserved for future use. Do not use.
unsigned	event↔ Hazardous↔ Materials: 1	Indicates whether a hazmat load is present. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not present</li> <li>• 1 – Present</li> </ul>
unsigned	eventStability↔ Controlactivated↔ : 1	Indicates whether stability control is on. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Off</li> <li>• 1 – On</li> </ul>
unsigned	eventTraction↔ ControlLoss: 1	Indicates whether traction control is activated (1) or not (0). <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not applied</li> <li>• 1 – Applied</li> </ul>



Type	Field	Description
unsigned	eventAB↔ Sactivated: 1	Indicates whether ABS is activated.  <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not activated</li> <li>• 1 – Activated</li> </ul>
unsigned	eventStop↔ LineViolation: 1	Indicates whether the vehicle has detected that a violation of the Stop Line is imminent.  <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not imminent</li> <li>• 1 – Imminent</li> </ul>
unsigned	eventHazard↔ Lights: 1	Indicates whether the hazard lights are on.  <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Off</li> <li>• 1 – On</li> </ul>

### 5.24.2.5 union ExteriorLights\_ut

Contains information about the state of the exterior lights.

#### Data fields

Type	Field	Description
struct <a href="#">Exterior↔ Lights_ut</a>	bits	Bit values for exterior light flags.Bit values for exterior light flags.
unsigned short	data	16-bit short word access to the packed ExteriorLights union structure.

### 5.24.2.6 struct ExteriorLights\_ut.bits

Bit values for exterior light flags.

#### Data fields

Type	Field	Description
unsigned	parking↔ LightsOn: 1	Indicates whether the parking lights are on.  <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Off</li> <li>• 1 – On</li> </ul>
unsigned	fogLightOn: 1	Indicates whether the fog lights are on.  <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Off</li> <li>• 1 – On</li> </ul>

Type	Field	Description
unsigned	daytime↔ Running↔ LightsOn: 1	Indicates whether the running lights are on. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Off</li> <li>• 1 – On</li> </ul>
unsigned	automatic↔ LightControl↔ On: 1	Indicates whether the automatic light control is on. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Off</li> <li>• 1 – On</li> </ul>
unsigned	hazardSignal↔ On: 1	Indicates whether the hazard lights are on. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Off</li> <li>• 1 – On</li> </ul>
unsigned	rightTurn↔ SignalOn: 1	Indicates whether the right turn light is on. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Off</li> <li>• 1 – On</li> </ul>
unsigned	leftTurn↔ SignalOn: 1	Indicates whether the left turn light is on. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Off</li> <li>• 1 – On</li> </ul>
unsigned	highBeam↔ HeadlightsOn: 1	Indicates whether the high beam headlights are on. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Off</li> <li>• 1 – On</li> </ul>
unsigned	lowBeam↔ HeadlightsOn: 1	Indicates whether the low beam headlights are on. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Off</li> <li>• 1 – On</li> </ul>
unsigned	unused: 7	Unused padding bits.

### 5.24.2.7 struct high\_resolution\_motion\_t

Contains high-resolution motion parameters.

#### Data fields

Type	Field	Description
double	vehicle_speed	Vehicle speed in meters/second.
double	longitudinal_↔ acceleration	Acceleration in a longitudinal direction, in meters/second <sup>2</sup> .
double	yaw_rate	Yaw rate in degrees/second, per SAE J2735.

### 5.24.2.8 struct current\_dynamic\_vehicle\_state\_t

Contains information about the dynamic state of the vehicle.

#### Data fields

Type	Field	Description
<a href="#">v2x_↔ transmission_↔ state_enum_↔ type</a>	prndl	Specifies the current state of the transmission gear: forward, reverse, and so on.
<a href="#">vehicleEvent↔ Flags_ut</a>	events	Flags all critical events and combinations of critical events.
double	throttle_↔ position	Per the J2735 definition, indicates the throttle position from 0% to 100%. However, this value is in double precision between 0 and 1.
double	throttle_↔ confidence	Per the J2735 definition, double precision degrees of confidence.
double	steering_↔ wheel_angle	Per the J2735 definition, double precision degrees of the wheel angle. <b>Supported values:</b> -192.0 through 189.0 degrees, with positive being turned to the right
<a href="#">v2x_control_↔ status_ut</a>	brake_status	Indicates whether brakes or emergency brakes (ABS) are activated. <b>Supported values:</b> <ul style="list-style-type: none"> <li>• 0 – Not activated</li> <li>• 1 – Activated</li> </ul>
<a href="#">Exterior↔ Lights_ut</a>	exterior_lights	Conglomeration of bits that indicate the status of the exterior lights, such as blinkers.
unsigned char	front_wiper_↔ status	Status of the front windshield wipers.
unsigned char	rear_wiper_↔ status	Status of the rear windshield wipers.

### 5.24.2.9 struct static\_vehicle\_parameters\_t

Contains static vehicle parameters.

#### Data fields

Type	Field	Description
double	vehicle_↔ height_cm	Vehicle height in centimeters. This parameter is 0 if the value is not yet available from the vehicle network.
double	vehicle_↔ width_cm	Vehicle width in centimeters. This parameter is 0 if the value is not yet available from the vehicle network.
double	vehicle_↔ length_cm	Vehicle length in centimeters. This parameter is 0 if the value is not yet available from the vehicle network.

Type	Field	Description
double	front_bumper_↔ _height_cm	Height of the front bumper, in centimeters. This parameter is 0 if the value is not yet available from the vehicle network.
double	rear_bumper_↔ height_cm	Height of the rear bumper, in centimeters. This parameter is 0 if the value is not yet available from the vehicle network.
double	vehicle_mass_↔ _kg	Mass of the vehicle, in kilograms. This parameter is 0 if the value is not yet available from the vehicle network.
double	trailer_↔ weight_kg	Weight of a trailer connected to the vehicle, in kilograms. This parameter is 0 if the value is not yet available from the vehicle network.
char *	make	Pointer to the NULL-terminated string with the vehicle manufacturer that this software build supports (such as Ford and GM).
char *	model	Pointer to the NULL-terminated string with the vehicle model name that this software build supports (such as Prius, Mustang, Rogue).
unsigned short	begin_model_↔ _year	Beginning of the model years that this software build supports.
unsigned short	end_model_↔ year	End of model year that this software build supports. This year might be the same as begin_model_year.

## 5.24.3 Enumeration Type Documentation

### 5.24.3.1 enum v2x\_transmission\_state\_enum\_type

Valid types for main transmission drive states.

#### Enumerator

**V2X\_TRANSMISSION\_NEUTRAL**  
**V2X\_TRANSMISSION\_PARK**  
**V2X\_TRANSMISSION\_FORWARD\_GEARs** One of the gears: D, 1, 2, 3, ... .  
**V2X\_TRANSMISSION\_REVERSE\_GEARs**  
**V2X\_TRANSMISSION\_RESERVED1**  
**V2X\_TRANSMISSION\_RESERVED2**  
**V2X\_TRANSMISSION\_RESERVED3**  
**V2X\_TRANSMISSION\_UNAVAILABLE** Status is unknown.  
**V2X\_TRANSMISSION\_MAX** Sentry variable that must not be exceeded.

### 5.24.3.2 enum v2x\_BrakeBoostApplied\_enum\_type

Valid types for brake boosting states.

#### Enumerator

**V2X\_BRAKEBOOST\_UNAVAIL** Status is unknown.  
**V2X\_BRAKEBOOST\_OFF**  
**V2X\_BRAKEBOOST\_ON**  
**V2X\_BRAKEBOOST\_MAX** Sentry variable that must not be exceeded.

### 5.24.3.3 enum v2x\_TractionControlStatus\_enum\_type

Valid types for traction control states.

This enumeration currently matches the J2735 2016 version for the Traction Control System (TCS).

#### Enumerator

**V2X\_TRACTION\_CTRL\_UNAVAIL** Status is unknown.  
**V2X\_TRACTION\_CTRL\_OFF**  
**V2X\_TRACTION\_CTRL\_ON** On but currently not engaged.  
**V2X\_TRACTION\_CTRL\_ENGAGED** Actively being engaged.

#### 5.24.3.4 enum v2x\_AntiLockBrakeStatus\_enum\_type

Valid types for antilock-braking states.

This enumeration matches the J2735 2016 version for the Antilock Braking System (ABS) to help BSM packing and unpacking.

##### Enumerator

**V2X\_ABS\_Unavailable** ABS is not equipped, or the status is unknown.

**V2X\_ABS\_Off**

**V2X\_ABS\_On** On but currently not active.

**V2X\_ABS\_Engaged** Actively being engaged on one or more wheels.

#### 5.24.3.5 enum v2x\_StabilityControlStatus\_enum\_type

Valid types for the stability control status. This enumeration should be equivalent to the J2735 BSM version you are working with.

##### Enumerator

**V2X\_STABILITY\_CONTROL\_UNAVAILBLE** Stability Control status is unknown.

**V2X\_STABILITY\_CONTROL\_OFF** Stability Control is not applied.

**V2X\_STABILITY\_CONTROL\_ON** Stability Control is on, but currently it is not engaged.

**V2X\_STABILITY\_CONTROL\_ENGAGED** Stability Control is actively being engaged.

#### 5.24.3.6 enum v2x\_AuxBrakeStatus\_enum\_type

Valid types for the auxiliary brake status.

This enumeration should match the J2735 2016 version or any other version you are working with.

##### Enumerator

**V2X\_AUX\_BRAKE\_UNAVAILBLE** Vehicle has no auxiliary brake equipment or the status is unknown.

**V2X\_AUX\_BRAKE\_OFF**

**V2X\_AUX\_BRAKE\_ON**

**V2X\_AUX\_BRAKE\_RESERVED**

## 5.24.4 Function Documentation

### 5.24.4.1 `v2x_api_ver_t v2x_vehicle_api_version ( void )`

Gets the compiled API version interface (as an integer number).

#### Returns

`v2x_api_ver_t` – Filled with the version number, build date, and detailed build information.

### 5.24.4.2 `v2x_status_enum_type v2x_vehicle_get_static_params ( static_vehicle_↔ parameters_t * parameters )`

Returns (via a reference pointer) the `static_vehicle_parameters_t` structure that enumerates static (unchanging) data items used by ITS stacks.

#### Associated data types

`static_vehicle_parameters_t`

#### Parameters

out	<i>parameters</i>	Pointer to the static vehicle parameters, including vehicle dimensions, make, model, and so on.
-----	-------------------	---

#### Detailed description

This call is a nonblocking call. If the values are not yet available from the vehicle, the data element is 0 (NULL).

Because this function is sometimes populated with data from an in-vehicle network, it might be incomplete and only partially populated early in a system start-up. However, all values can be statically compiled in or loaded from an initialization file. In this case, the data is fully provided on the first call.

#### Returns

`V2X_STATUS_SUCCESS` – This function is successfully populated with the results.

Error code – If there is a problem (see `v2x_status_enum_type`).

### 5.24.4.3 `v2x_motion_data_handle_t v2x_high_res_motion_register_listener ( v2x_high_res_motion_listener_t cb )`

Registers for high-resolution motion callbacks from the vehicle data network (CAN bus) when the data changes.

#### Associated data types

[v2x\\_high\\_res\\_motion\\_listener\\_t](#)

#### Parameters

<code>in</code>	<code>cb</code>	Callback function to use for this listener.
-----------------	-----------------	---

#### Returns

Handle number to use with subsequent deregister calls.

-1 – If there is an error in registering a callback.

### 5.24.4.4 `v2x_status_enum_type v2x_high_res_motion_deregister_listener ( v2x_motion_data_handle_t handle )`

Deregisters a previously registered high-resolution motion data callback that was requested via [v2x\\_high\\_res\\_motion\\_register\\_listener\(\)](#).

#### Associated data types

[v2x\\_motion\\_data\\_handle\\_t](#)

#### Parameters

<code>in</code>	<code>handle</code>	Handle of the listener callback previously set up.
-----------------	---------------------	--

#### Returns

[V2X\\_STATUS\\_SUCCESS](#).

#### Dependencies

The callback must have been previously registered with [v2x\\_high\\_res\\_motion\\_register\\_listener\(\)](#).



#### 5.24.4.5 `v2x_vehicle_handle_t v2x_vehicle_register_listener ( v2x_vehicle_event_listener_t cb, void * context )`

Registers for a callback for state updates from the vehicle data network (CAN bus). This function requests vehicle data callbacks when data changes or events occur.

##### Associated data types

[v2x\\_vehicle\\_event\\_listener\\_t](#)

##### Parameters

in	<i>cb</i>	Callback function to use for this listener.
in	<i>context</i>	Pointer to the application context for use with the callbacks, which can help the caller code.

##### Returns

Handle number to use with subsequent deregister calls.

-1 – If there is an error in registering a callback.

#### 5.24.4.6 `v2x_status_enum_type v2x_vehicle_deregister_for_callback ( v2x_vehicle_handle_t handle )`

Deregisters a previously registered dynamic event callback that was requested via [v2x\\_vehicle\\_register\\_listener\(\)](#).

##### Associated data types

[v2x\\_vehicle\\_handle\\_t](#)

##### Parameters

in	<i>handle</i>	Handle of the listener callback previously set up.
----	---------------	--

##### Returns

[V2X\\_STATUS\\_SUCCESS](#) – If the callback is successfully deregistered.

##### Dependencies

The callback must have been previously registered.

## 5.25 Telematics\_net

### 5.25.1 Data Structure Documentation

#### 5.25.1.1 class telux::data::net::IFirewallManager

IFirewallManager is a primary interface that filters and controls the network traffic on a pre-configured set of rules.

##### Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status setFirewall](#) (bool enable, bool allowPackets, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status requestFirewallStatus](#) ([FirewallStatusCb](#) callback)=0
- virtual [telux::common::Status addFirewallEntry](#) (std::shared\_ptr< [IFirewallEntry](#) > entry, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status requestFirewallEntry](#) ([FirewallEntriesCb](#) callback)=0
- virtual [telux::common::Status removeFirewallEntry](#) (const std::shared\_ptr< [IFirewallEntry](#) > &entry, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status addDmz](#) (const std::string ipAddr, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status removeDmz](#) (const std::string ipAddr, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status requestDmzEntries](#) ([DmzEntriesCb](#) dmzCb)=0
- virtual [telux::data::OperationType getOperationType](#) ()=0
- virtual [~IFirewallManager](#) ()

#### 5.25.1.1.1 Constructors and Destructors

5.25.1.1.1.1 virtual [telux::data::net::IFirewallManager::~~IFirewallManager](#) ( ) [virtual]

Destructor for [IFirewallManager](#)

#### 5.25.1.1.2 Member Function Documentation

**5.25.1.1.2.1 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.

**5.25.1.1.2.2 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.

**5.25.1.1.2.3 virtual telux::common::Status telux::data::net::IFirewallManager::setFirewall ( 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>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.

**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::IFirewallManager::requestFirewallStatus ( FirewallStatusCb callback ) [pure virtual]`

Request status of firewall

##### Parameters

in	<i>callback</i>	callback to get the response of requestFirewallStatus
----	-----------------	---

##### Returns

Status of requestFirewallStatus 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::IFirewallManager::addFirewallEntry ( std::shared_ptr< IFirewallEntry > entry, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Adds the firewall rule

##### Parameters

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.

##### 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::IFirewallManager::requestFirewallEntry ( FirewallEntriesCb callback ) [pure virtual]`

Request Firewall rules

##### Parameters

in	<i>callback</i>	callback to get the response requestFirewallEntry
----	-----------------	---

**Returns**

Status of requestFirewallEntry 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::IFirewallManager::removeFirewallEntry ( const std::shared_ptr< IFirewallEntry > & entry, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Remove firewall entry

**Parameters**

in	<i>entry</i>	Firewall entry to be removed, get the available entries from <a href="#">requestFirewallEntry()</a> API
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.

**5.25.1.1.2.8** `virtual telux::common::Status telux::data::net::IFirewallManager::addDmz ( const std::string ipAddr, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Adds demilitarized zone (DMZ) IP address

**Parameters**

in	<i>ipAddr</i>	IP address to add
in	<i>callback</i>	optional callback to get the response addDmz

**Returns**

Status of addDmz 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.

**5.25.1.1.2.9** `virtual telux::common::Status telux::data::net::IFirewallManager::removeDmz ( const std::string ipAddr, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Removes demilitarized zone (DMZ) IP address

#### Parameters

in	<i>ipAddr</i>	IP address to remove
in	<i>callback</i>	optional callback to get the response removeDmz

#### Returns

Status of removeDmz 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.

**5.25.1.1.2.10** `virtual telux::common::Status telux::data::net::IFirewallManager::requestDmzEntries ( DmzEntriesCb dmzCb ) [pure virtual]`

Request DMZ entries that was previously set using addDmz API

#### Parameters

in	<i>dmzCb</i>	callback to get the response requestDmzEntries
----	--------------	--

#### Returns

Status of requestDmzEntries 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.

**5.25.1.1.2.11** `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.

**5.25.1.2 class telux::data::net::IFirewallEntry**

Firewall entry class is used for configuring firewall rules.

**Public member functions**

- virtual `std::shared_ptr< IipFilter > getIProtocolFilter ()=0`
- virtual `telux::data::Direction getDirection ()=0`
- virtual `telux::data::IpFamilyType getIpFamilyType ()=0`
- virtual `~IFirewallEntry ()`

**5.25.1.2.1 Constructors and Destructors**

**5.25.1.2.1.1 virtual telux::data::net::IFirewallEntry::~IFirewallEntry ( ) [virtual]**

Destructor for [IFirewallEntry](#)

**5.25.1.2.2 Member Function Documentation**

**5.25.1.2.2.1 virtual std::shared\_ptr<IipFilter> telux::data::net::IFirewallEntry::getIProtocolFilter ( ) [pure virtual]**

Get IProtocol filter type

**Returns**

[telux::data::IipFilter](#).

**5.25.1.2.2.2 virtual telux::data::Direction telux::data::net::IFirewallEntry::getDirection ( ) [pure virtual]**

Get firewall direction

**Returns**

[telux::data::Direction](#).

### 5.25.1.2.2.3 virtual telux::data::IpFamilyType telux::data::net::IFirewallEntry::getIpFamilyType ( ) [pure virtual]

Get Ip FamilyType

#### Returns

[telux::data::IpFamilyType](#).

### 5.25.1.3 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 <a href="#">telux::net::IpProtocol</a>

### 5.25.1.4 class telux::data::net::INatManager

NatManager is a primary interface for configuring static network address translation(SNAT) and DMZ (demilitarized zone)

#### Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status addStaticNatEntry](#) (const [NatConfig](#) &snatConfig, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status removeStaticNatEntry](#) (const [NatConfig](#) &snatConfig, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status requestStaticNatEntries](#) ([StaticNatEntriesCb](#) snatEntriesCb)=0
- virtual [telux::data::OperationType getOperationType](#) ()=0
- virtual [~INatManager](#) ()

#### 5.25.1.4.1 Constructors and Destructors

##### 5.25.1.4.1.1 virtual telux::data::net::INatManager::~~INatManager ( ) [virtual]

Destructor for [INatManager](#)



### 5.25.1.4.2 Member Function Documentation

#### 5.25.1.4.2.1 `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.

#### 5.25.1.4.2.2 `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.

#### 5.25.1.4.2.3 `virtual telux::common::Status telux::data::net::INatManager::addStaticNatEntry ( 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>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.

**5.25.1.4.2.4** `virtual telux::common::Status telux::data::net::INatManager::removeStaticNatEntry ( 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>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.

**5.25.1.4.2.5** `virtual telux::common::Status telux::data::net::INatManager::requestStaticNatEntries ( StaticNatEntriesCb snatEntriesCb ) [pure virtual]`

Request list of static nat entries available in the NAT table

**Parameters**

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.

#### 5.25.1.4.2.6 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.

### 5.25.1.5 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.

#### 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 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 [telux::data::OperationType getOperationType](#) ()=0
- virtual [~IVlanManager](#) ()

#### 5.25.1.5.1 Constructors and Destructors

##### 5.25.1.5.1.1 virtual telux::data::net::IVlanManager::~~IVlanManager ( ) [virtual]

Destructor for [IVlanManager](#)

### 5.25.1.5.2 Member Function Documentation

#### 5.25.1.5.2.1 `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.

#### 5.25.1.5.2.2 `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.

#### 5.25.1.5.2.3 `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.

#### Parameters

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.

#### 5.25.1.5.2.4 `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.

#### 5.25.1.5.2.5 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.

#### 5.25.1.5.2.6 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.

**Note**

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

#### 5.25.1.5.2.7 **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

**Note**

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

#### 5.25.1.5.2.8 **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

**Note**

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

#### 5.25.1.5.2.9 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.

# 6 Namespace Documentation

---

## 6.1 telux Namespace Reference

### Namespaces

- [audio](#)
- [common](#)
- [config](#)
- [cv2x](#)
- [data](#)
- [loc](#)
- [power](#)
- [rsp](#)
- [tel](#)
- [therm](#)

## 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 [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](#)
- struct [PlatformVersion](#)
- 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>
----	------------------	---------------------------

### 6.3.2 Enumeration Type Documentation

#### 6.3.2.1 `enum telux::common::ServiceStatus [strong]`

Service status.

#### Enumerator

***SERVICE\_UNAVAILABLE***  
***SERVICE\_AVAILABLE***

## 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

- class [Cv2xFactory](#)

*Cv2xFactory* is the factory that creates the Cv2x Radio. [More...](#)

- struct [Cv2xPoolStatus](#)

- struct [Cv2xRadioCapabilities](#)

- struct [Cv2xStatus](#)

- struct [Cv2xStatusEx](#)

- struct [DataSessionSettings](#)

- struct [EventFlowInfo](#)

- struct [GlobalIPUnicastRoutingInfo](#)

- 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](#)
- struct [IPv6Address](#)
- struct [IPv6AddrType](#)
- struct [MacDetails](#)
- struct [SocketInfo](#)
- struct [SpsFlowInfo](#)
- struct [SpsSchedulingInfo](#)
- struct [TrustedUEInfo](#)
- struct [TrustedUEInfoList](#)
- struct [TxPoolIdInfo](#)

## 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.

**Parameters**

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 • SUCCESS • GENERIC_FAILURE

#### Note

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

### 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 • SUCCESS • GENERIC_FAILURE

#### Note

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

### 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.

**Parameters**

in	<i>error</i>	<ul style="list-style-type: none"> <li>- SUCCESS if update succeeded</li> <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>
----	--------------	---

**Note**

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

### 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>	<ul style="list-style-type: none"> <li>- SUCCESS if Tx reservation change succeeded</li> <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>	<ul style="list-style-type: none"> <li>- Indicates whether TCP socket creation succeeded</li> <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 <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> </ul>
----	--------------	---

### 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 <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> </ul>
----	--------------	---

### 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 <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> </ul>

**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 <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> </ul>

#### Note

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

**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 <ul style="list-style-type: none"> <li>• SUCCESS</li> <li>• GENERIC_FAILURE</li> </ul>
----	--------------	---

## 6.6 telux::data Namespace Reference

### Namespaces

- [net](#)

### Data Structures

- 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 [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. [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 [IEspFilter](#)

*This class represents a IP Filter for the ESP, get the new instance from [telux::data::DataFactory](#).*
- class [IicmpFilter](#)

*This class represents a IP Filter for the ICMP, get the new instance from [telux::data::DataFactory](#).*
- class [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.*
- struct [IpAddrInfo](#)
- struct [IPv4Info](#)
- struct [IPv6Info](#)

- 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 [PortInfo](#)
- struct [ProfileParams](#)
- struct [TcpInfo](#)
- 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::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 )

### 6.6.1.4 struct telux::data::IPv6Info

IPv6 header info

#### Data fields

Type	Field	Description
string	srcAddr	address of the device that sends the packet.
string	destAddr	address of receiving end
<a href="#">IpProtocol</a>	nextProtoId	Protocol ID (i.e TCP, UDP or ICMP )
<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	

### 6.6.1.5 struct telux::data::TcpInfo

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

### 6.6.1.6 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.

**Parameters**

in	<i>dataCall</i>	Pointer to <a href="#">IDataCall</a>
in	<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**

in	<i>dataStats</i>	Data Call statistics
in	<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::DataRestrictModeCb = typedef std::function<void(DataRestrictMode 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.

### 6.6.2.5 using telux::data::TypeOfService = typedef uint8\_t

### 6.6.2.6 using telux::data::TrafficClass = typedef uint8\_t

### 6.6.2.7 using telux::data::FlowLabel = typedef uint32\_t

## 6.7 telux::data::net Namespace Reference

### Data Structures

- class [IFirewallEntry](#)  
*Firewall entry class is used for configuring firewall rules. [More...](#)*
- class [IFirewallManager](#)  
*FirewallManager is a primary interface that filters and controls the network traffic on a pre-configured set of rules. [More...](#)*
- class [INatManager](#)  
*NatManager is a primary interface for configuring static network address translation(SNAT) and DMZ (demilitarized zone) [More...](#)*
- 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. [More...](#)*
- struct [NatConfig](#)

### 6.7.1 Typedef Documentation

#### 6.7.1.1 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>

#### Note

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

#### 6.7.1.2 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 requestFirewallEntry()

**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>

**Note**

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

### 6.7.1.3 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>

**Note**

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

### 6.7.1.4 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.

### 6.7.1.5 using `telux::data::net::CreateVlanCb = typedef std::function<void(bool isAccelerated, telux::common::ErrorCode error)>`

This function is called as a response to `createVlan()`



**Parameters**

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.

### 6.7.1.6 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.

### 6.7.1.7 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.

## 6.8 telux::loc Namespace Reference

## Data Structures

- struct [GlonassTimeInfo](#)
- struct [GnssData](#)
- struct [GnssKinematicsData](#)
- struct [GnssMeasurementInfo](#)
- 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 [IGpsTime](#)

*IGpsTime* provides interface to get current GPS week and elapsed time in current GPS week. [More...](#)
- class [ILocationConfigurator](#)

*ILocationConfigurator* allows for the enablement/disablement of the time uncertainty. It also allows to set the threshold and the required power level for the `configureCTunc` API. [More...](#)
- class [ILocationInfo](#)

*ILocationInfo* provides interface to get basic position related information like latitude, longitude, altitude, timestamp and other information like time stamp, session status,. [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 [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 [ISensorDataUsage](#)

*Specifies* the sensors used for calculating the fixes and the type of measurements which were aided by sensor data. [More...](#)
- class [ISVInfo](#)

*ISVInfo* provides interface to retrieve information about Satellite Vehicles, their position and health status. [More...](#)

- class [LocationFactory](#)  
*LocationFactory* allows creation of location manager. [More...](#)
- struct [SystemTime](#)
- union [SystemTimeInfo](#)
- struct [TimeInfo](#)

## 6.9 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...](#)

## 6.10 telux::rsp Namespace Reference

### Data Structures

- struct [CustomHeader](#)
- class [IHttpTransactionListener](#)  
*The interface listens for indication to perform HTTP request and send back the response for HTTP request to modem. [More...](#)*
- class [IHttpTransactionManager](#)  
*IHttpTransactionManager* is the interface to service HTTP related requests from the modem, for Sim profile update related operations. [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 on the eUICC. [More...](#)
- class [SimProfile](#)  
*SimProfile* class represents single eUICC profile on the card. [More...](#)

- class [SimProfileFactory](#)

*SimProfileFactory* is the central factory to create all eUICC manager class instances. [More...](#)

## 6.10.1 Typedef Documentation

### 6.10.1.1 `using telux::rsp::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.

#### Note

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

## 6.11 telux::tel Namespace Reference

### Data Structures

- struct [CardReaderStatus](#)
- class [CdmaCellIdentity](#)
- class [CdmaCellInfo](#)
- class [CdmaSignalStrengthInfo](#)
- class [CellInfo](#)
- struct [CellularCapabilityInfo](#)
- struct [ECallModeInfo](#)
- struct [ECallMsdControlBits](#)
- struct [ECallMsdData](#)
- struct [ECallMsdOptionals](#)
- struct [ECallOptionalPdu](#)
- struct [ECallVehicleIdentificationNumber](#)
- struct [ECallVehicleLocation](#)
- struct [ECallVehicleLocationDelta](#)

- struct [ECallVehiclePropulsionStorageType](#)
- 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 [ICellularCapabilityCallback](#)

- 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...](#)*

- 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 [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](#)

*Serving System Manager class provides the API to request and set service domain preference and RAT preference. [More...](#)*

- 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 [ISmscAddressCallback](#)

- class [ISmsListener](#)

*A listener class for monitoring incoming SMS. Override the methods for the state that you wish to receive updates for. [More...](#)*

- class [ISmsManager](#)

*SMS Manager class is the primary interface to send and receive SMS messages. It allows to send an SMS in several formats and sizes. [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...](#)

- class [OperatorInfo](#)
- struct [OperatorStatus](#)
- class [PhoneFactory](#)

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

- struct [PreferredNetworkInfo](#)
- class [SignalStrength](#)
- struct [SimRatCapability](#)
- class [SmsMessage](#)

A Short Message Service message. [More...](#)

- class [TdscdmaCellIdentity](#)
- class [TdscdmaCellInfo](#)
- class [TdscdmaSignalStrengthInfo](#)
- class [VoiceServiceInfo](#)
- class [WcdmaCellIdentity](#)
- class [WcdmaCellInfo](#)
- class [WcdmaSignalStrengthInfo](#)

## 6.11.1 Typedef Documentation

### 6.11.1.1 `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

### 6.11.1.2 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

**Note**

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

### 6.11.1.3 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

**Note**

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

### 6.11.1.4 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



**Note**

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

### 6.11.1.5 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

**Note**

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

### 6.11.1.6 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.

### 6.11.1.7 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.

**Parameters**

in	<i>mode</i>	<a href="#">NetworkSelectionMode</a>
in	<i>error</i>	Return code which indicates whether the operation succeeded or not ErrorCode

### 6.11.1.8 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. ErrorCode

**6.11.1.9 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. ErrorCode

**6.11.1.10 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>

**6.11.1.11 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
-----	--------------	---

**6.11.1.12** 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

#### Note

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

**6.11.1.13** using `telux::tel::RATCapabilitiesMask = typedef std::bitset<16>`

**6.11.1.14** using `telux::tel::VoiceServiceTechnologiesMask = typedef std::bitset<16>`

**6.11.1.15** 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

**6.11.1.16** 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.

**6.11.1.17** 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

**6.11.1.18 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

**6.12 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...](#)*

# 7 Data Structure Documentation

---

## 7.1 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.

### 7.1.1 Constructors and Destructors

#### 7.1.1.1 virtual telux::cv2x::ICv2xListener::~~ICv2xListener ( ) [virtual]

Destructor for `ICv2xListener`

#### Note

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

### 7.1.2 Member Function Documentation

#### 7.1.2.1 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.
----	---------------	----------------------

#### Note

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

**Deprecated** use `onStatusChanged(Cv2xStatusEx status)`

### 7.1.2.2 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.
----	---------------	----------------------

#### Note

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

## 7.2 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](#).

### 7.2.1 Constructors and Destructors

#### 7.2.1.1 virtual telux::data::IEspFilter::~~IEspFilter ( ) [virtual]

Destructor for [IEspFilter](#)

### 7.2.2 Member Function Documentation

#### 7.2.2.1 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.

### 7.2.2.2 virtual telux::common::Status telux::data::IEspFilter::setEspInfo ( const EspInfo & espInfo ) [pure virtual]

sets the ICMP header info

#### Parameters

in	<i>espInfo</i>	EspInfo 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.

## 7.3 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](#).

### 7.3.1 Constructors and Destructors

#### 7.3.1.1 virtual telux::data::IcmpFilter::~~IcmpFilter ( ) [virtual]

Destructor for [IcmpFilter](#)

### 7.3.2 Member Function Documentation

#### 7.3.2.1 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.

### 7.3.2.2 virtual telux::common::Status telux::data::IcmpFilter::setIcmpInfo ( const IcmpInfo & icmpInfo ) [pure virtual]

sets the ICMP header info

**Parameters**

in	<i>icmpInfo</i>	TcpInfo 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.

## 7.4 telux::data::IIPFilter 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 [~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.

### 7.4.1 Constructors and Destructors

#### 7.4.1.1 virtual telux::data::IIPFilter::~~IIPFilter ( ) [virtual]

Destructor for [IIPFilter](#)

## 7.4.2 Member Function Documentation

### 7.4.2.1 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.

### 7.4.2.2 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.

### 7.4.2.3 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.

### 7.4.2.4 virtual telux::common::Status telux::data::llpFilter::setIPv6Info ( const IPv6Info & *ipv6Info* ) [pure virtual]

sets the IPv6 header info

**Parameters**

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.

**7.4.2.5 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.

**7.5 telux::common::IServiceStatusListener Class Reference****Public member functions**

- virtual void [onServiceStatusChange](#) ([ServiceStatus](#) status)
- virtual [~IServiceStatusListener](#) ()

**7.5.1 Constructors and Destructors****7.5.1.1 virtual telux::common::IServiceStatusListener::~~IServiceStatusListener ( ) [virtual]****7.5.2 Member Function Documentation****7.5.2.1 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>
----	---------------	---------------------------------

## 7.6 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](#).

### 7.6.1 Constructors and Destructors

#### 7.6.1.1 virtual `telux::data::ITcpFilter::~~ITcpFilter ( ) [virtual]`

Destructor for [ITcpFilter](#)

### 7.6.2 Member Function Documentation

#### 7.6.2.1 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.

#### 7.6.2.2 virtual `telux::common::Status telux::data::ITcpFilter::setTcpInfo ( const TcpInfo & tcpInfo ) [pure virtual]`

sets the TCP header info

#### Parameters

in	<i>tcpInfo</i>	<a href="#">TcpInfo</a> 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.

## 7.7 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](#).

### 7.7.1 Constructors and Destructors

#### 7.7.1.1 virtual `telux::data::IUdpFilter::~~IUdpFilter ( ) [virtual]`

Destructor for [IUdpFilter](#)

### 7.7.2 Member Function Documentation

#### 7.7.2.1 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.

#### 7.7.2.2 virtual `telux::common::Status telux::data::IUdpFilter::setUdpInfo ( const UdpInfo & udpInfo ) [pure virtual]`

sets the UDP header info

#### Parameters

in	<i>udpInfo</i>	<a href="#">UdpInfo</a> 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.