



Qualcomm Technologies, Inc.

Telematics SDK

API Reference
v1.33.83

80-79289-15 Rev. AA

August 28, 2025

Revision History

Revision	Date	Description
AA	Aug 2025	Initial release

Software Revision History

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

Contents

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

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

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

5.1	Telematics SDK APIs	133
5.2	Phone Factory	134
5.2.1	Data Structure Documentation	134
5.2.1.1	class telux::tel::PhoneFactory	134
5.3	Phone	139
5.3.1	Data Structure Documentation	139
5.3.1.1	class telux::tel::GsmCellIdentity	139
5.3.1.2	class telux::tel::CdmaCellIdentity	140
5.3.1.3	class telux::tel::LteCellIdentity	142
5.3.1.4	class telux::tel::WcdmaCellIdentity	143
5.3.1.5	class telux::tel::TdscdmaCellIdentity	145
5.3.1.6	class telux::tel::CellInfo	146
5.3.1.7	class telux::tel::GsmCellInfo	147
5.3.1.8	class telux::tel::CdmaCellInfo	148
5.3.1.9	class telux::tel::LteCellInfo	149
5.3.1.10	class telux::tel::WcdmaCellInfo	150
5.3.1.11	class telux::tel::TdscdmaCellInfo	151
5.3.1.12	struct telux::tel::ECallMsdOptionals	152
5.3.1.13	struct telux::tel::ECallMsdControlBits	153
5.3.1.14	struct telux::tel::ECallVehicleIdentificationNumber	153
5.3.1.15	struct telux::tel::ECallVehiclePropulsionStorageType	154
5.3.1.16	struct telux::tel::ECallVehicleLocation	156
5.3.1.17	struct telux::tel::ECallVehicleLocationDelta	156
5.3.1.18	struct telux::tel::ECallDeltaV	157
5.3.1.19	struct telux::tel::ECallOptionalEuroNcapData	157
5.3.1.20	struct telux::tel::ECallOptionalPdu	158
5.3.1.21	struct telux::tel::ECallMsdData	158
5.3.1.22	struct telux::tel::ECallModelInfo	159
5.3.1.23	struct telux::tel::ECallHlapTimerStatus	159
5.3.1.24	struct telux::tel::ECallHlapTimerEvents	160
5.3.1.25	struct telux::tel::EcallConfig	160
5.3.1.26	class telux::tel::IPhone	161
5.3.1.27	class telux::tel::ISignalStrengthCallback	165
5.3.1.28	class telux::tel::IVoiceServiceStateCallback	165
5.3.1.29	struct telux::tel::SimRatCapability	166
5.3.1.30	struct telux::tel::CellularCapabilityInfo	166
5.3.1.31	class telux::tel::IPhoneListener	167
5.3.1.32	class telux::tel::IPhoneManager	170
5.3.1.33	class telux::tel::ICellularCapabilityCallback	173
5.3.1.34	class telux::tel::IOperatingModeCallback	174
5.3.1.35	class telux::tel::SignalStrength	175
5.3.1.36	class telux::tel::LteSignalStrengthInfo	177
5.3.1.37	class telux::tel::GsmSignalStrengthInfo	179
5.3.1.38	class telux::tel::CdmaSignalStrengthInfo	180
5.3.1.39	class telux::tel::WcdmaSignalStrengthInfo	182
5.3.1.40	class telux::tel::TdscdmaSignalStrengthInfo	183
5.3.1.41	class telux::tel::Nr5gSignalStrengthInfo	184
5.3.1.42	class telux::tel::VoiceServiceInfo	185
5.3.2	Enumeration Type Documentation	186
5.3.2.1	CellType	186

	5.3.2.2	ECallVariant	186
	5.3.2.3	EmergencyCallType	187
	5.3.2.4	ECallMsdTransmissionStatus	187
	5.3.2.5	ECallCategory	187
	5.3.2.6	ECallVehicleType	187
	5.3.2.7	ECallOptionalDataType	188
	5.3.2.8	ECallLocationOfImpact	188
	5.3.2.9	ECallMode	189
	5.3.2.10	ECallModeReason	189
	5.3.2.11	HlapTimerStatus	189
	5.3.2.12	HlapTimerEvent	189
	5.3.2.13	ECallNumType	190
	5.3.2.14	RadioState	190
	5.3.2.15	ServiceState	190
	5.3.2.16	RadioTechnology	190
	5.3.2.17	RATCapability	191
	5.3.2.18	VoiceServiceTechnology	191
	5.3.2.19	OperatingMode	191
	5.3.2.20	EcbMode	192
	5.3.2.21	SignalStrengthLevel	192
	5.3.2.22	VoiceServiceState	192
	5.3.2.23	VoiceServiceDenialCause	192
5.4	Call		194
5.4.1	Data Structure Documentation		194
	5.4.1.1	class telux::tel::ICall	194
	5.4.1.2	class telux::tel::ICallListener	201
	5.4.1.3	class telux::tel::ICallManager	204
	5.4.1.4	class telux::tel::IMakeCallCallback	217
5.4.2	Enumeration Type Documentation		218
	5.4.2.1	CallDirection	218
	5.4.2.2	CallState	218
	5.4.2.3	CallEndCause	218
5.5	SMS		223
5.5.1	Data Structure Documentation		223
	5.5.1.1	struct telux::tel::MessageAttributes	223
	5.5.1.2	struct telux::tel::MessagePartInfo	223
	5.5.1.3	class telux::tel::SmsMessage	223
	5.5.1.4	class telux::tel::ISmsManager	226
	5.5.1.5	class telux::tel::ISmsListener	230
	5.5.1.6	class telux::tel::ISmscAddressCallback	232
5.5.2	Enumeration Type Documentation		232
	5.5.2.1	SmsEncoding	232
5.6	SIM Card Services		233
5.6.1	Data Structure Documentation		233
	5.6.1.1	class telux::tel::ICardApp	233
	5.6.1.2	struct telux::tel::IccResult	235
	5.6.1.3	class telux::tel::ICardManager	236
	5.6.1.4	class telux::tel::ICard	238
	5.6.1.5	class telux::tel::ICardChannelCallback	243
	5.6.1.6	class telux::tel::ICardCommandCallback	243

5.6.1.7	class telux::tel::ICardListener	244
5.6.1.8	struct telux::tel::CardReaderStatus	244
5.6.1.9	class telux::tel::ISapCardManager	245
5.6.1.10	class telux::tel::IAtrResponseCallback	250
5.6.1.11	class telux::tel::ISapCardCommandCallback	251
5.6.1.12	class telux::tel::ICardReaderCallback	251
5.6.1.13	class telux::tel::ISapCardListener	252
5.6.2	Enumeration Type Documentation	252
5.6.2.1	CardState	252
5.6.2.2	CardError	252
5.6.2.3	CardLockType	253
5.6.2.4	AppType	253
5.6.2.5	AppState	253
5.6.2.6	SapState	253
5.6.2.7	SapCondition	254
5.7	Location Services	255
5.7.1	Data Structure Documentation	255
5.7.1.1	class telux::loc::IDgnssStatusListener	255
5.7.1.2	class telux::loc::IDgnssManager	255
5.7.1.3	struct telux::loc::GnssKinematicsData	258
5.7.1.4	struct telux::loc::LLAInfo	258
5.7.1.5	struct telux::loc::TimeInfo	259
5.7.1.6	struct telux::loc::GlonassTimeInfo	259
5.7.1.7	union telux::loc::SystemTimeInfo	260
5.7.1.8	struct telux::loc::SystemTime	261
5.7.1.9	struct telux::loc::GnssMeasurementInfo	261
5.7.1.10	struct telux::loc::SvUsedInPosition	261
5.7.1.11	struct telux::loc::GnssData	262
5.7.1.12	struct telux::loc::SvBlackListInfo	262
5.7.1.13	struct telux::loc::LeverArmParams	263
5.7.1.14	struct telux::loc::ExternalLocationInfo	263
5.7.1.15	struct telux::loc::GnssMeasurementsData	263
5.7.1.16	struct telux::loc::GnssMeasurementsClock	264
5.7.1.17	struct telux::loc::GnssMeasurements	265
5.7.1.18	struct telux::loc::LeapSecondChangeInfo	265
5.7.1.19	struct telux::loc::LeapSecondInfo	266
5.7.1.20	struct telux::loc::LocationSystemInfo	266
5.7.1.21	struct telux::loc::GnssEnergyConsumedInfo	266
5.7.1.22	struct telux::loc::RobustLocationVersion	267
5.7.1.23	struct telux::loc::RobustLocationConfiguration	267
5.7.1.24	struct telux::loc::NmeaConfig	267
5.7.1.25	struct telux::loc::BodyToSensorMountParams	267
5.7.1.26	struct telux::loc::DREngineConfiguration	268
5.7.1.27	struct telux::loc::XtraConfig	269
5.7.1.28	struct telux::loc::XtraStatus	270
5.7.1.29	class telux::loc::ILocationInfoBase	270
5.7.1.30	class telux::loc::ILocationInfoEx	274
5.7.1.31	class telux::loc::ISVInfo	284
5.7.1.32	class telux::loc::IGnssSVInfo	287
5.7.1.33	class telux::loc::IGnssSignalInfo	288

5.7.1.34	class telux::loc::LocationFactory	288
5.7.1.35	class telux::loc::ILocationListener	289
5.7.1.36	class telux::loc::ILocationSystemInfoListener	292
5.7.1.37	class telux::loc::ILocationInjectionListener	292
5.7.1.38	class telux::loc::ILocationConfigListener	293
5.7.2	Enumeration Type Documentation	294
5.7.2.1	DgnssDataFormat	294
5.7.2.2	DgnssStatus	294
5.7.2.3	HorizontalAccuracyLevel	294
5.7.2.4	LocationReliability	294
5.7.2.5	AltitudeType	295
5.7.2.6	GnssConstellationType	295
5.7.2.7	SVHealthStatus	295
5.7.2.8	SVStatus	295
5.7.2.9	SVInfoAvailability	296
5.7.2.10	GnssPositionTechType	296
5.7.2.11	KinematicDataValidityType	296
5.7.2.12	GnssSystem	297
5.7.2.13	GnssTimeValidityType	297
5.7.2.14	GlonassTimeValidity	297
5.7.2.15	GnssSignalType	297
5.7.2.16	LocationTechnologyType	298
5.7.2.17	LocationValidityType	298
5.7.2.18	LocationInfoExValidityType	299
5.7.2.19	GnssDataSignalTypes	300
5.7.2.20	GnssDataValidityType	300
5.7.2.21	DrCalibrationStatusType	300
5.7.2.22	LocReqEngineType	301
5.7.2.23	LocationAggregationType	301
5.7.2.24	PositioningEngineType	301
5.7.2.25	LeverArmType	301
5.7.2.26	GnssMeasurementsDataValidityType	302
5.7.2.27	GnssMeasurementsStateValidityType	302
5.7.2.28	GnssMeasurementsAdrStateValidityType	303
5.7.2.29	GnssMeasurementsMultipathIndicator	303
5.7.2.30	GnssMeasurementsClockValidityType	303
5.7.2.31	LeapSecondInfoValidityType	303
5.7.2.32	LocationSystemInfoValidityType	304
5.7.2.33	GnssEnergyConsumedInfoValidityType	304
5.7.2.34	AidingDataType	304
5.7.2.35	TerrestrialTechnologyType	304
5.7.2.36	RobustLocationConfigType	304
5.7.2.37	NmeaSentenceType	304
5.7.2.38	GeodeticDatumType	305
5.7.2.39	DRConfigValidityType	305
5.7.2.40	GnssReportType	305
5.7.2.41	EngineType	306
5.7.2.42	ReportStatus	306
5.7.2.43	DebugLogLevel	306
5.7.2.44	XtraDataStatus	306

	5.7.2.45	LocConfigIndicationsType	307
5.7.3		Function Documentation	307
	5.7.3.1	isSubsystemReady	307
	5.7.3.2	onSubsystemReady	307
	5.7.3.3	configureCTunc	307
	5.7.3.4	configurePACE	308
	5.7.3.5	deleteAllAidingData	308
	5.7.3.6	configureLeverArm	309
	5.7.3.7	configureConstellations	309
	5.7.3.8	configureRobustLocation	310
	5.7.3.9	requestRobustLocation	310
	5.7.3.10	configureMinGpsWeek	310
	5.7.3.11	requestMinGpsWeek	311
	5.7.3.12	configureMinSVElevation	311
	5.7.3.13	requestMinSVElevation	312
	5.7.3.14	deleteAidingData	312
	5.7.3.15	configureDR	313
	5.7.3.16	configureNmeaTypes	313
	5.7.3.17	configureNmea	314
	5.7.3.18	provideConsentForTerrestrialPositioning	314
	5.7.3.19	registerLocationInjector	315
	5.7.3.20	deregisterLocationInjector	315
	5.7.3.21	injectLocationData	316
	5.7.3.22	configureEngineIntegrityRisk	316
	5.7.3.23	configureXtraParams	317
	5.7.3.24	requestXtraStatus	317
	5.7.3.25	registerListener	318
	5.7.3.26	deRegisterListener	318
	5.7.3.27	~ILocationConfigurator	318
	5.7.3.28	isSubsystemReady	319
	5.7.3.29	onSubsystemReady	319
	5.7.3.30	registerListenerEx	319
	5.7.3.31	deRegisterListenerEx	319
	5.7.3.32	startDetailedReports	320
	5.7.3.33	startDetailedEngineReports	320
	5.7.3.34	startBasicReports	321
	5.7.3.35	registerForSystemInfoUpdates	322
	5.7.3.36	deRegisterForSystemInfoUpdates	323
	5.7.3.37	requestEnergyConsumedInfo	323
	5.7.3.38	stopReports	324
	5.7.3.39	getTerrestrialPosition	324
	5.7.3.40	cancelTerrestrialPositionRequest	325
	5.7.3.41	~ILocationManager	325
5.7.4		Variable Documentation	325
	5.7.4.1	UNKNOWN_CARRIER_FREQ	325
	5.7.4.2	UNKNOWN_SIGNAL_MASK	325
	5.7.4.3	UNKNOWN_TIMESTAMP	325
	5.7.4.4	DEFAULT_TUNC_THRESHOLD	325
	5.7.4.5	DEFAULT_TUNC_ENERGY_THRESHOLD	325
	5.7.4.6	DEFAULT_GNSS_REPORT	325

5.8	Data Services	326
5.8.1	Define Documentation	326
5.8.1.1	PROFILE_ID_MAX	326
5.8.1.2	MAX_QOS_FILTERS	326
5.8.1.3	IP_PROT_UNKNOWN	326
5.8.2	Data Structure Documentation	326
5.8.2.1	class telux::data::IDataConnectionManager	326
5.8.2.2	class telux::data::IDataCall	331
5.8.2.3	class telux::data::IDataConnectionListener	335
5.8.2.4	struct telux::data::DataRestrictMode	336
5.8.2.5	struct telux::data::PortInfo	337
5.8.2.6	struct telux::data::ProfileParams	337
5.8.2.7	struct telux::data::DataCallStats	337
5.8.2.8	struct telux::data::IpAddrInfo	338
5.8.2.9	struct telux::data::DataCallEndReason	338
5.8.2.10	struct telux::data::LatencyConfig	338
5.8.2.11	struct telux::data::BackhaulInfo	338
5.8.2.12	struct telux::data::VlanConfig	339
5.8.2.13	struct telux::data::FlowDataRate	339
5.8.2.14	struct telux::data::QosIPFlowInfo	339
5.8.2.15	class telux::data::DataFactory	339
5.8.2.16	class telux::data::IDataFilterListener	345
5.8.2.17	class telux::data::IDataFilterManager	346
5.8.2.18	class telux::data::DataProfile	351
5.8.2.19	class telux::data::IDataProfileListener	354
5.8.2.20	class telux::data::IDataProfileManager	355
5.8.2.21	class telux::data::IDataCreateProfileCallback	359
5.8.2.22	class telux::data::IDataProfileListCallback	359
5.8.2.23	class telux::data::IDataProfileCallback	360
5.8.2.24	class telux::data::IDataSettingsManager	360
5.8.2.25	class telux::data::IDataSettingsListener	364
5.8.2.26	class telux::data::IServingSystemManager	364
5.8.2.27	class telux::data::IServingSystemListener	368
5.8.2.28	union telux::data::DataCallEndReason.__unnamed__	369
5.8.3	Enumeration Type Documentation	369
5.8.3.1	IpFamilyType	369
5.8.3.2	TechPreference	369
5.8.3.3	AuthProtocolType	370
5.8.3.4	DataRestrictModeType	370
5.8.3.5	ApnMaskType	370
5.8.3.6	DataCallStatus	370
5.8.3.7	DataBearerTechnology	371
5.8.3.8	EndReasonType	371
5.8.3.9	MobileIpReasonCode	372
5.8.3.10	InternalReasonCode	372
5.8.3.11	CallManagerReasonCode	374
5.8.3.12	SpecReasonCode	377
5.8.3.13	PPPReasonCode	378
5.8.3.14	EHRPDRReasonCode	379
5.8.3.15	Ipv6ReasonCode	379

5.8.3.16	HandoffReasonCode	379
5.8.3.17	ProfileChangeEvent	380
5.8.3.18	OperationType	380
5.8.3.19	Direction	380
5.8.3.20	InterfaceType	380
5.8.3.21	BackhaulType	381
5.8.3.22	LatencyLevel	381
5.8.3.23	ServiceState	381
5.8.3.24	QosFlowStateChangeEvent	381
5.8.3.25	IpTrafficClassType	381
5.8.3.26	QosIPFlowMaskType	382
5.8.3.27	QosFlowMaskType	382
5.9	Subscription Management	383
5.9.1	Data Structure Documentation	383
5.9.1.1	class telux::tel::ISubscription	383
5.9.1.2	class telux::tel::ISubscriptionListener	386
5.9.1.3	class telux::tel::ISubscriptionManager	387
5.10	Network Selection	390
5.10.1	Data Structure Documentation	390
5.10.1.1	struct telux::tel::PreferredNetworkInfo	390
5.10.1.2	struct telux::tel::OperatorStatus	390
5.10.1.3	class telux::tel::INetworkSelectionManager	390
5.10.1.4	class telux::tel::OperatorInfo	394
5.10.1.5	class telux::tel::INetworkSelectionListener	395
5.10.2	Enumeration Type Documentation	396
5.10.2.1	RatType	396
5.10.2.2	NetworkSelectionMode	396
5.10.2.3	InUseStatus	396
5.10.2.4	RoamingStatus	396
5.10.2.5	ForbiddenStatus	397
5.10.2.6	PreferredStatus	397
5.11	Serving System	398
5.11.1	Data Structure Documentation	398
5.11.1.1	struct telux::tel::DcStatus	398
5.11.1.2	class telux::tel::IServingSystemManager	398
5.11.1.3	class telux::tel::IServingSystemListener	402
5.11.2	Enumeration Type Documentation	403
5.11.2.1	ServiceDomainPreference	403
5.11.2.2	RatPrefType	403
5.11.2.3	EndcAvailability	404
5.11.2.4	DcnrRestriction	404
5.12	Common	405
5.12.1	Data Structure Documentation	405
5.12.1.1	class telux::common::ICommandCallback	405
5.12.1.2	class telux::common::ICommandResponseCallback	405
5.12.1.3	class telux::common::DeviceConfig	406
5.12.1.4	class telux::common::Log	406
5.12.1.5	struct telux::common::SdkVersion	406
5.12.1.6	class telux::common::Version	406
5.12.2	Enumeration Type Documentation	407

	5.12.2.1	Status	407
	5.12.2.2	ErrorCode	408
	5.12.2.3	ServiceStatus	412
	5.12.2.4	LogLevel	412
5.13	C-V2X		414
5.13.1	Data Structure Documentation		414
	5.13.1.1	class telux::cv2x::ICv2xConfigListener	414
	5.13.1.2	class telux::cv2x::ICv2xConfig	414
	5.13.1.3	class telux::cv2x::Cv2xFactory	416
	5.13.1.4	class telux::cv2x::ICv2xRadio	417
	5.13.1.5	class telux::cv2x::ICv2xRadioListener	428
	5.13.1.6	class telux::cv2x::ICv2xRadioManager	429
	5.13.1.7	struct telux::cv2x::SocketInfo	432
	5.13.1.8	struct telux::cv2x::Cv2xStatus	433
	5.13.1.9	struct telux::cv2x::Cv2xPoolStatus	433
	5.13.1.10	struct telux::cv2x::Cv2xStatusEx	433
	5.13.1.11	struct telux::cv2x::TxPoolIdInfo	434
	5.13.1.12	struct telux::cv2x::EventFlowInfo	434
	5.13.1.13	struct telux::cv2x::SpsFlowInfo	434
	5.13.1.14	struct telux::cv2x::Cv2xRadioCapabilities	435
	5.13.1.15	struct telux::cv2x::MacDetails	436
	5.13.1.16	struct telux::cv2x::SpsSchedulingInfo	437
	5.13.1.17	struct telux::cv2x::TrustedUEInfo	437
	5.13.1.18	struct telux::cv2x::TrustedUEInfoList	437
	5.13.1.19	struct telux::cv2x::IPv6Address	438
	5.13.1.20	struct telux::cv2x::DataSessionSettings	438
	5.13.1.21	struct telux::cv2x::ConfigEventInfo	438
	5.13.1.22	struct telux::cv2x::RFTxInfo	439
	5.13.1.23	struct telux::cv2x::TxStatusReport	439
	5.13.1.24	class telux::cv2x::ICv2xRxSubscription	440
	5.13.1.25	class telux::cv2x::ICv2xTxFlow	442
	5.13.1.26	class telux::cv2x::ICv2xTxRxSocket	443
5.13.2	Enumeration Type Documentation		444
	5.13.2.1	TrafficCategory	445
	5.13.2.2	Cv2xStatusType	445
	5.13.2.3	Cv2xCauseType	445
	5.13.2.4	TrafficIpType	445
	5.13.2.5	RadioConcurrencyMode	446
	5.13.2.6	Cv2xEvent	446
	5.13.2.7	Priority	446
	5.13.2.8	Periodicity	447
	5.13.2.9	ConfigSourceType	447
	5.13.2.10	ConfigEvent	447
	5.13.2.11	RFTxStatus	447
	5.13.2.12	SegmentType	448
	5.13.2.13	TxType	448
5.13.3	Variable Documentation		448
	5.13.3.1	MAX_ANTENNAS_SUPPORTED	448
5.14	Audio		449
5.14.1	Data Structure Documentation		449

5.14.1.1	struct telux::audio::FormatParams	449
5.14.1.2	struct telux::audio::Amrwbparams	449
5.14.1.3	struct telux::audio::StreamConfig	449
5.14.1.4	struct telux::audio::FormatInfo	450
5.14.1.5	struct telux::audio::ChannelVolume	450
5.14.1.6	struct telux::audio::StreamVolume	450
5.14.1.7	struct telux::audio::StreamMute	450
5.14.1.8	struct telux::audio::StreamBuffer	451
5.14.1.9	struct telux::audio::DtmfTone	451
5.14.1.10	class telux::audio::AudioFactory	451
5.14.1.11	class telux::audio::IVoiceListener	452
5.14.1.12	class telux::audio::IPlayListener	452
5.14.1.13	class telux::audio::ITranscodeListener	453
5.14.1.14	class telux::audio::IAudioListener	454
5.14.1.15	class telux::audio::IAudioBuffer	454
5.14.1.16	class telux::audio::IStreamBuffer	456
5.14.1.17	class telux::audio::IAudioManager	456
5.14.1.18	class telux::audio::IAudioDevice	459
5.14.1.19	class telux::audio::IAudioStream	460
5.14.1.20	class telux::audio::IAudioVoiceStream	462
5.14.1.21	class telux::audio::IAudioPlayStream	465
5.14.1.22	class telux::audio::IAudioCaptureStream	467
5.14.1.23	class telux::audio::IAudioLoopbackStream	468
5.14.1.24	class telux::audio::IAudioToneGeneratorStream	469
5.14.1.25	class telux::audio::ITranscoder	470
5.14.2	Enumeration Type Documentation	473
5.14.2.1	DeviceType	473
5.14.2.2	DeviceDirection	474
5.14.2.3	Direction	474
5.14.2.4	StreamType	474
5.14.2.5	StreamDirection	474
5.14.2.6	ChannelType	474
5.14.2.7	AudioFormat	475
5.14.2.8	DtmfLowFreq	475
5.14.2.9	DtmfHighFreq	475
5.14.2.10	Amrwbframeformat	475
5.14.2.11	StopType	475
5.15	Thermal Management	477
5.15.1	Data Structure Documentation	477
5.15.1.1	class telux::therm::ThermalFactory	477
5.15.1.2	struct telux::therm::BoundCoolingDevice	478
5.15.1.3	class telux::therm::IThermalManager	478
5.15.1.4	class telux::therm::ITripPoint	479
5.15.1.5	class telux::therm::IThermalZone	480
5.15.1.6	class telux::therm::ICoolingDevice	482
5.15.1.7	class telux::therm::IThermalShutdownListener	483
5.15.1.8	class telux::therm::IThermalShutdownManager	484
5.15.2	Enumeration Type Documentation	487
5.15.2.1	AutoShutdownMode	487
5.15.2.2	TripType	487

5.15.3	Variable Documentation	487
5.15.3.1	DEFAULT_TIMEOUT	487
5.16	TCU Activity Manager	488
5.16.1	Data Structure Documentation	488
5.16.1.1	class telux::power::PowerFactory	488
5.16.1.2	class telux::power::ITcuActivityListener	488
5.16.1.3	class telux::power::ITcuActivityManager	489
5.16.2	Enumeration Type Documentation	493
5.16.2.1	TcuActivityState	493
5.16.2.2	TcuActivityStateAck	493
5.16.2.3	ClientType	493
5.17	Remote SIM Provisioning	494
5.17.1	Data Structure Documentation	494
5.17.1.1	class telux::tel::SimProfile	494
5.17.1.2	class telux::tel::ISimProfileListener	497
5.17.1.3	class telux::tel::ISimProfileManager	498
5.17.2	Enumeration Type Documentation	503
5.17.2.1	ProfileType	503
5.17.2.2	IconType	503
5.17.2.3	ProfileClass	503
5.17.2.4	DownloadStatus	503
5.17.2.5	DownloadErrorCause	504
5.17.2.6	PolicyRuleType	504
5.18	Remote SIM	505
5.18.1	Data Structure Documentation	505
5.18.1.1	class telux::tel::IRemoteSimListener	505
5.18.1.2	class telux::tel::IRemoteSimManager	506
5.18.2	Enumeration Type Documentation	511
5.18.2.1	CardErrorCause	511
5.19	Modem Config	512
5.19.1	Data Structure Documentation	512
5.19.1.1	class telux::config::ConfigFactory	512
5.19.1.2	struct telux::config::ConfigInfo	512
5.19.1.3	class telux::config::IModemConfigListener	513
5.19.1.4	class telux::config::IModemConfigManager	513
5.19.2	Enumeration Type Documentation	518
5.19.2.1	ConfigType	518
5.19.2.2	AutoSelectionMode	518
5.19.2.3	ConfigUpdateStatus	518
5.20	Wlan	519
5.20.1	Define Documentation	519
5.20.1.1	INVALID_AP_ID	519
5.20.2	Data Structure Documentation	519
5.20.2.1	struct telux::wlan::ApNetConfig	519
5.20.2.2	struct telux::wlan::ApConfig	519
5.20.2.3	struct telux::wlan::DeviceIndInfo	519
5.20.2.4	struct telux::wlan::DeviceInfo	520
5.20.2.5	struct telux::wlan::DeviceStats	520
5.20.2.6	class telux::wlan::IStaInterfaceManager	520
5.20.2.7	class telux::wlan::IStaListener	520

5.20.2.8	struct telux::wlan::ApInfo	521
5.20.2.9	struct telux::wlan::ApNetInfo	521
5.20.2.10	struct telux::wlan::ApStatus	521
5.20.2.11	struct telux::wlan::StaStatus	521
5.20.2.12	struct telux::wlan::InterfaceStatus	522
5.20.2.13	class telux::wlan::IWlanDeviceManager	522
5.20.2.14	class telux::wlan::IWlanListener	522
5.20.2.15	class telux::wlan::WlanFactory	523
5.20.3	Enumeration Type Documentation	524
5.20.3.1	ApInterworking	524
5.20.3.2	ApDeviceConnectionEvent	525
5.20.3.3	BandType	525
5.20.3.4	ConnectionStatus	525
5.20.3.5	Id	525
5.20.3.6	ApType	525
5.20.3.7	StaInterfaceStatus	526
5.20.3.8	OperationType	526
5.20.3.9	IpFamilyType	526
5.20.3.10	ServiceOperation	526
5.20.3.11	InterfaceState	526
5.20.3.12	HwDeviceType	527
5.20.4	Function Documentation	527
5.20.4.1	setConfig	527
5.20.4.2	getConfig	527
5.20.4.3	getStatus	528
5.20.4.4	getConnectedDevices	528
5.20.4.5	manageApService	529
5.20.4.6	registerListener	529
5.20.4.7	deregisterListener	529
5.20.4.8	~IApInterfaceManager	530
5.20.4.9	getConnectedDevicesStats	530
5.20.4.10	onApDeviceStatusChanged	530
5.20.4.11	onApBandChanged	530
5.20.4.12	~IApListener	531
5.20.4.13	onApDeviceStatusChanged	531
5.20.4.14	getServiceStatus	531
5.20.4.15	enable	531
5.20.4.16	setMode	532
5.20.4.17	getConfig	532
5.20.4.18	getStatus	533
5.20.4.19	registerListener	533
5.20.4.20	deregisterListener	534
5.20.4.21	~IWlanDeviceManager	534
5.20.4.22	onServiceStatusChange	534
5.20.4.23	onEnableChanged	534
5.20.4.24	~IWlanListener	534
5.20.5	Variable Documentation	534
5.20.5.1	info	534
5.20.5.2	interworking	535
5.20.5.3	id	535

5.20.5.4	network	535
5.20.5.5	id	535
5.20.5.6	macAddress	535
5.20.5.7	id	535
5.20.5.8	name	535
5.20.5.9	ipv4Address	535
5.20.5.10	ipv6Address	535
5.20.5.11	macAddress	535
5.20.5.12	id	535
5.20.5.13	macAddress	535
5.20.5.14	bytesTx	536
5.20.5.15	bytesRx	536
5.20.5.16	device	536
5.20.5.17	apStatus	536
5.20.5.18	staStatus	536
5.21	Wlan Device Management	537
5.22	Access Point Management	538
5.22.1	Data Structure Documentation	538
5.22.1.1	class telux::wlan::IApInterfaceManager	538
5.22.1.2	class telux::wlan::IApListener	538
5.23	Station Management	539
5.23.1	Data Structure Documentation	539
5.23.1.1	struct telux::wlan::StaStaticIpConfig	539
5.23.1.2	struct telux::wlan::StaConfig	539
5.23.2	Enumeration Type Documentation	539
5.23.2.1	StaIpConfig	539
5.23.2.2	StaBridgeMode	539
5.23.3	Function Documentation	540
5.23.3.1	setIpConfig	540
5.23.3.2	setBridgeMode	540
5.23.3.3	getConfig	541
5.23.3.4	getStatus	541
5.23.3.5	manageStaService	541
5.23.3.6	registerListener	542
5.23.3.7	deregisterListener	542
5.23.3.8	~IStaInterfaceManager	542
5.23.3.9	onStationStatusChanged	543
5.23.3.10	onStationBandChanged	543
5.23.3.11	~IStaListener	543
5.23.4	Variable Documentation	543
5.23.4.1	ipAddr	543
5.23.4.2	gwIpAddr	543
5.23.4.3	netMask	543
5.23.4.4	dnsAddr	543
5.23.4.5	stald	543
5.23.4.6	ipConfig	543
5.23.4.7	staticIpConfig	544
5.23.4.8	bridgeMode	544
5.24	Telematics_cv2x_cpp	545
5.24.1	Data Structure Documentation	545

	5.24.1.1	class telux::cv2x::ICv2xThrottleManagerListener	545
	5.24.1.2	class telux::cv2x::ICv2xThrottleManager	546
	5.24.1.3	class telux::cv2x::ICv2xTxStatusReportListener	548
5.25	Telematics_net		549
	5.25.1	Data Structure Documentation	549
	5.25.1.1	class telux::data::net::IBridgeManager	549
	5.25.1.2	class telux::data::net::IBridgeListener	553
	5.25.1.3	class telux::data::net::IFirewallManager	553
	5.25.1.4	class telux::data::net::IFirewallEntry	563
	5.25.1.5	class telux::data::net::IFirewallListener	564
	5.25.1.6	struct telux::data::net::L2tpSessionConfig	565
	5.25.1.7	struct telux::data::net::L2tpTunnelConfig	565
	5.25.1.8	struct telux::data::net::L2tpSysConfig	565
	5.25.1.9	class telux::data::net::IL2tpManager	566
	5.25.1.10	class telux::data::net::IL2tpListener	569
	5.25.1.11	struct telux::data::net::NatConfig	570
	5.25.1.12	class telux::data::net::INatManager	570
	5.25.1.13	class telux::data::net::INatListener	573
	5.25.1.14	class telux::data::net::ISocksManager	574
	5.25.1.15	class telux::data::net::ISocksListener	576
	5.25.1.16	class telux::data::net::IVlanManager	577
	5.25.1.17	class telux::data::net::IVlanListener	583
	5.25.2	Enumeration Type Documentation	584
	5.25.2.1	L2tpProtocol	584
5.26	Telematics_platform_deviceinfo		585
	5.26.1	Data Structure Documentation	585
	5.26.1.1	class telux::platform::IDeviceInfoListener	585
	5.26.1.2	struct telux::platform::PlatformVersion	585
	5.26.1.3	class telux::platform::IDeviceInfoManager	585
5.27	Telematics_platform		588
	5.27.1	Data Structure Documentation	588
	5.27.1.1	class telux::platform::PlatformFactory	588
5.28	Telematics_cellbroadcast		590
	5.28.1	Data Structure Documentation	590
	5.28.1.1	struct telux::tel::CellBroadcastFilter	590
	5.28.2	Enumeration Type Documentation	590
	5.28.2.1	GeographicalScope	590
	5.28.2.2	MessagePriority	590
	5.28.2.3	MessageType	590
	5.28.2.4	EtwsWarningType	591
	5.28.2.5	CmasMessageClass	591
	5.28.2.6	CmasSeverity	591
	5.28.2.7	CmasUrgency	591
	5.28.2.8	CmasCertainty	592
	5.28.2.9	GeometryType	592
5.29	Telematics_ecall		593
	5.29.1	Data Structure Documentation	593
	5.29.1.1	class telux::tel::IEcallManager	593
	5.29.1.2	class telux::tel::IEcallListener	595
5.30	Telematics_ims_settings		596

5.30.1	Data Structure Documentation	596
5.30.1.1	struct telux::tel::ImsServiceConfig	596
5.30.1.2	class telux::tel::ImsSettingsManager	596
5.30.1.3	class telux::tel::ImsSettingsListener	599
5.30.2	Enumeration Type Documentation	600
5.30.2.1	ImsServiceConfigType	600
5.31	Telematics_multi_sim	601
5.31.1	Data Structure Documentation	601
5.31.1.1	struct telux::tel::SlotStatus	601
5.31.1.2	class telux::tel::IMultiSimManager	601
5.31.1.3	class telux::tel::IMultiSimListener	605
5.31.2	Enumeration Type Documentation	606
5.31.2.1	SlotState	606
6	Namespace Documentation	607
6.1	telux Namespace Reference	607
6.2	telux::audio Namespace Reference	607
6.2.1	Typedef Documentation	609
6.2.1.1	ChannelTypeMask	609
6.2.1.2	GetDevicesResponseCb	609
6.2.1.3	GetStreamTypesResponseCb	609
6.2.1.4	CreateStreamResponseCb	609
6.2.1.5	CreateTranscoderResponseCb	610
6.2.1.6	DeleteStreamResponseCb	610
6.2.1.7	GetStreamDeviceResponseCb	610
6.2.1.8	GetStreamVolumeResponseCb	611
6.2.1.9	GetStreamMuteResponseCb	611
6.2.1.10	WriteResponseCb	611
6.2.1.11	ReadResponseCb	611
6.2.1.12	TranscoderReadResponseCb	612
6.2.1.13	TranscoderWriteResponseCb	612
6.2.2	Variable Documentation	613
6.2.2.1	INFINITE_DTMF_DURATION	613
6.2.2.2	INFINITE_TONE_DURATION	613
6.3	telux::common Namespace Reference	613
6.3.1	Typedef Documentation	613
6.3.1.1	ResponseCallback	613
6.3.1.2	InitResponseCb	614
6.4	telux::config Namespace Reference	614
6.4.1	Typedef Documentation	614
6.4.1.1	ConfigId	614
6.4.1.2	ConfigListCallback	614
6.4.1.3	GetAutoSelectionModeCallback	615
6.4.1.4	GetActiveConfigCallback	615
6.5	telux::cv2x Namespace Reference	615
6.5.1	Typedef Documentation	616
6.5.1.1	CreateRxSubscriptionCallback	616
6.5.1.2	CreateTxSpsFlowCallback	617
6.5.1.3	CreateTxEventFlowCallback	617
6.5.1.4	CloseTxFlowCallback	617

6.5.1.5	CloseRxSubscriptionCallback	618
6.5.1.6	ChangeSpsFlowInfoCallback	618
6.5.1.7	RequestSpsFlowInfoCallback	618
6.5.1.8	ChangeEventFlowInfoCallback	619
6.5.1.9	RequestCapabilitiesCallback	619
6.5.1.10	RequestDataSessionSettingsCallback	619
6.5.1.11	UpdateTrustedUEListCallback	619
6.5.1.12	UpdateSrcL2InfoCallback	620
6.5.1.13	CreateTcpSocketCallback	620
6.5.1.14	CloseTcpSocketCallback	620
6.5.1.15	StartCv2xCallback	621
6.5.1.16	StopCv2xCallback	621
6.5.1.17	RequestCv2xStatusCallback	621
6.5.1.18	RequestCv2xStatusCallbackEx	621
6.5.1.19	UpdateConfigurationCallback	622
6.5.1.20	setVerificationLoadCallback	622
6.6	telux::data Namespace Reference	622
6.6.1	Data Structure Documentation	625
6.6.1.1	struct telux::data::IpFamilyInfo	625
6.6.1.2	struct telux::data::QosFilterRule	625
6.6.1.3	struct telux::data::TrafficFlowTemplate	625
6.6.1.4	struct telux::data::TftChangeInfo	626
6.6.1.5	struct telux::data::IPv4Info	626
6.6.1.6	struct telux::data::IPv6Info	626
6.6.1.7	struct telux::data::UdpInfo	627
6.6.1.8	struct telux::data::TcpInfo	627
6.6.1.9	struct telux::data::IcmpInfo	627
6.6.1.10	struct telux::data::EspInfo	627
6.6.1.11	struct telux::data::RoamingStatus	628
6.6.1.12	struct telux::data::ServiceStatus	628
6.6.2	Typedef Documentation	628
6.6.2.1	DataCallResponseCb	628
6.6.2.2	StatisticsResponseCb	629
6.6.2.3	DataCallListResponseCb	629
6.6.2.4	TrafficFlowTemplateCb	630
6.6.2.5	ApnTypes	630
6.6.2.6	IpProtocol	630
6.6.2.7	QosFlowId	630
6.6.2.8	QosIPFlowMask	630
6.6.2.9	QosFlowMask	630
6.6.2.10	DataRestrictModeCb	630
6.6.2.11	TypeOfService	631
6.6.2.12	TrafficClass	631
6.6.2.13	FlowLabel	631
6.6.2.14	RequestServiceStatusResponseCb	631
6.6.2.15	RequestRoamingStatusResponseCb	631
6.6.3	Enumeration Type Documentation	631
6.6.3.1	RoamingType	631
6.6.3.2	DataServiceState	632
6.6.3.3	NetworkRat	632

6.7	telux::data::net Namespace Reference	632
6.7.1	Data Structure Documentation	633
6.7.1.1	struct telux::data::net::BridgeInfo	633
6.7.1.2	struct telux::data::net::FirewallConfig	634
6.7.1.3	struct telux::data::net::DmzConfig	634
6.7.1.4	struct telux::data::net::FirewallEntryInfo	634
6.7.1.5	struct telux::data::net::VlanBindConfig	634
6.7.2	Typedef Documentation	634
6.7.2.1	BridgeInfoResponseCb	635
6.7.2.2	FirewallConfigCb	635
6.7.2.3	FirewallEntryInfoCb	635
6.7.2.4	DmzEntryInfoCb	635
6.7.2.5	FirewallStatusCb	636
6.7.2.6	FirewallEntriesCb	636
6.7.2.7	DmzEntriesCb	636
6.7.2.8	L2tpConfigCb	637
6.7.2.9	StaticNatEntriesCb	637
6.7.2.10	CreateVlanCb	637
6.7.2.11	QueryVlanResponseCb	638
6.7.2.12	VlanMappingResponseCb	638
6.7.2.13	VlanBindingsResponseCb	638
6.7.3	Enumeration Type Documentation	639
6.7.3.1	BridgeIFaceType	639
6.8	telux::loc Namespace Reference	639
6.8.1	Typedef Documentation	641
6.8.1.1	SbasCorrection	641
6.8.1.2	GnssPositionTech	641
6.8.1.3	KinematicDataValidity	641
6.8.1.4	LocConfigIndications	641
6.8.1.5	GnssReportTypeMask	641
6.8.1.6	DRConfigValidity	641
6.8.1.7	NmeaSentenceConfig	641
6.8.1.8	RobustLocationConfig	641
6.8.1.9	TerrestrialTechnology	641
6.8.1.10	AidingData	641
6.8.1.11	GnssEnergyConsumedInfoValidity	641
6.8.1.12	LocationSystemInfoValidity	642
6.8.1.13	LeapSecondInfoValidity	642
6.8.1.14	GnssMeasurementsClockValidity	642
6.8.1.15	GnssMeasurementsAdrStateValidity	642
6.8.1.16	GnssMeasurementsStateValidity	642
6.8.1.17	GnssMeasurementsDataValidity	642
6.8.1.18	PositioningEngine	642
6.8.1.19	LocReqEngine	642
6.8.1.20	DrCalibrationStatus	642
6.8.1.21	GnssDataValidity	642
6.8.1.22	LocationInfoExValidity	642
6.8.1.23	LocationInfoValidity	642
6.8.1.24	LocationTechnology	642
6.8.1.25	GnssSignal	642

6.8.1.26	GnssTimeValidity	643
6.8.1.27	TimeValidity	643
6.8.2	Enumeration Type Documentation	643
6.8.2.1	SbasCorrectionType	643
6.9	telux::platform Namespace Reference	643
6.10	telux::power Namespace Reference	643
6.11	telux::tel Namespace Reference	644
6.11.1	Data Structure Documentation	649
6.11.1.1	struct telux::tel::Point	649
6.11.1.2	struct telux::tel::ImsRegistrationInfo	650
6.11.1.3	struct telux::tel::NetworkTimeInfo	650
6.11.2	Typedef Documentation	651
6.11.2.1	MakeCallCallback	651
6.11.2.2	ECallHlapTimerStatusCallback	651
6.11.2.3	EcbmStatusCallback	651
6.11.2.4	PinOperationResponseCb	652
6.11.2.5	QueryFdnLockResponseCb	652
6.11.2.6	QueryPin1LockResponseCb	652
6.11.2.7	EidResponseCallback	652
6.11.2.8	RequestFiltersResponseCallback	653
6.11.2.9	RequestActivationStatusResponseCallback	653
6.11.2.10	EcallConfigValidity	653
6.11.2.11	ImsRegistrationInfoCb	653
6.11.2.12	ImsServiceConfigValidity	654
6.11.2.13	ImsServiceConfigCb	654
6.11.2.14	HighCapabilityCallback	654
6.11.2.15	SlotStatusCallback	655
6.11.2.16	RatMask	655
6.11.2.17	SelectionModeResponseCallback	655
6.11.2.18	PreferredNetworksCallback	655
6.11.2.19	NetworkScanCallback	656
6.11.2.20	VoiceRadioTechResponseCb	656
6.11.2.21	CellInfoCallback	656
6.11.2.22	ECallGetOperatingModeCallback	656
6.11.2.23	RATCapabilitiesMask	657
6.11.2.24	VoiceServiceTechnologiesMask	657
6.11.2.25	SapStateResponseCallback	657
6.11.2.26	RatPreference	657
6.11.2.27	RatPreferenceCallback	657
6.11.2.28	ServiceDomainPreferenceCallback	657
6.11.2.29	NetworkTimeResponseCallback	658
6.11.2.30	PolicyRuleMask	658
6.11.2.31	ProfileListResponseCb	658
6.11.2.32	EidResponseCb	658
6.11.2.33	PduBuffer	659
6.11.2.34	SmsResponseCb	659
6.11.3	Enumeration Type Documentation	659
6.11.3.1	EcallConfigType	659
6.11.3.2	EcallHlapTimerId	660
6.11.3.3	RegistrationStatus	660

6.12	telux::therm Namespace Reference	660
6.12.1	Typedef Documentation	661
6.12.1.1	GetAutoShutdownModeResponseCb	661
6.13	telux::wlan Namespace Reference	661
7	Data Structure Documentation	663
7.1	telux::tel::CellBroadcastMessage Class Reference	663
7.1.1	Constructors and Destructors	663
7.1.1.1	CellBroadcastMessage	663
7.1.1.2	CellBroadcastMessage	663
7.1.2	Member Function Documentation	664
7.1.2.1	getMessageType	664
7.1.2.2	getEtwsInfo	664
7.1.2.3	getCmasInfo	664
7.2	telux::tel::Circle Class Reference	664
7.2.1	Constructors and Destructors	665
7.2.1.1	Circle	665
7.2.2	Member Function Documentation	665
7.2.2.1	getCenter	665
7.2.2.2	getRadius	665
7.3	telux::tel::CmasInfo Class Reference	665
7.3.1	Constructors and Destructors	666
7.3.1.1	CmasInfo	666
7.3.2	Member Function Documentation	666
7.3.2.1	getGeographicalScope	667
7.3.2.2	getMessageId	667
7.3.2.3	getSerialNumber	667
7.3.2.4	getLanguageCode	667
7.3.2.5	getMessageBody	668
7.3.2.6	getPriority	668
7.3.2.7	getMessageCode	668
7.3.2.8	getUpdateNumber	669
7.3.2.9	getMessageClass	669
7.3.2.10	getSeverity	669
7.3.2.11	getUrgency	669
7.3.2.12	getCertainty	670
7.3.2.13	getWarningAreaInfo	670
7.4	telux::tel::EtwsInfo Class Reference	670
7.4.1	Constructors and Destructors	671
7.4.1.1	EtwsInfo	671
7.4.2	Member Function Documentation	671
7.4.2.1	getGeographicalScope	671
7.4.2.2	getMessageId	672
7.4.2.3	getSerialNumber	672
7.4.2.4	getLanguageCode	672
7.4.2.5	getMessageBody	673
7.4.2.6	getPriority	673
7.4.2.7	getMessageCode	673
7.4.2.8	getUpdateNumber	673
7.4.2.9	getEtwsWarningType	674

	7.4.2.10	isEmergencyUserAlert	674
	7.4.2.11	isPopupAlert	674
	7.4.2.12	isPrimary	674
	7.4.2.13	getWarningSecurityInformation	675
7.5	telux::tel::Geometry	Class Reference	675
7.5.1	Constructors and Destructors		675
	7.5.1.1	Geometry	675
	7.5.1.2	Geometry	675
7.5.2	Member Function Documentation		676
	7.5.2.1	getType	676
	7.5.2.2	getPolygon	676
	7.5.2.3	getCircle	676
7.6	telux::tel::ICellBroadcastListener	Class Reference	676
7.6.1	Constructors and Destructors		677
	7.6.1.1	~ICellBroadcastListener	677
7.6.2	Member Function Documentation		677
	7.6.2.1	onIncomingMessage	677
	7.6.2.2	onMessageFilterChange	677
7.7	telux::tel::ICellBroadcastManager	Class Reference	678
7.7.1	Constructors and Destructors		678
	7.7.1.1	~ICellBroadcastManager	678
7.7.2	Member Function Documentation		678
	7.7.2.1	isSubsystemReady	678
	7.7.2.2	onSubsystemReady	679
	7.7.2.3	getSlotId	679
	7.7.2.4	updateMessageFilters	679
	7.7.2.5	requestMessageFilters	680
	7.7.2.6	setActivationStatus	680
	7.7.2.7	requestActivationStatus	680
	7.7.2.8	registerListener	681
	7.7.2.9	deregisterListener	681
7.8	telux::cv2x::ICv2xListener	Class Reference	682
7.8.1	Constructors and Destructors		682
	7.8.1.1	~ICv2xListener	682
7.8.2	Member Function Documentation		682
	7.8.2.1	onStatusChanged	682
	7.8.2.2	onStatusChanged	682
7.9	telux::data::IEspFilter	Class Reference	682
7.9.1	Constructors and Destructors		683
	7.9.1.1	~IEspFilter	683
7.9.2	Member Function Documentation		683
	7.9.2.1	getEspInfo	683
	7.9.2.2	setEspInfo	683
7.10	telux::data::IlcmpFilter	Class Reference	684
7.10.1	Constructors and Destructors		684
	7.10.1.1	~IlcmpFilter	684
7.10.2	Member Function Documentation		684
	7.10.2.1	getIcmpInfo	684
	7.10.2.2	setIcmpInfo	684
7.11	telux::tel::IImServingSystemListener	Class Reference	685

7.11.1	Constructors and Destructors	685
7.11.1.1	~IImServingSystemListener	685
7.11.2	Member Function Documentation	685
7.11.2.1	onServiceStatusChange	685
7.11.2.2	onImSRegStatusChange	686
7.12	telux::tel::IImServingSystemManager Class Reference	686
7.12.1	Constructors and Destructors	686
7.12.1.1	~IImServingSystemManager	686
7.12.2	Member Function Documentation	686
7.12.2.1	getServiceStatus	687
7.12.2.2	requestRegistrationInfo	687
7.12.2.3	registerListener	687
7.12.2.4	deregisterListener	688
7.13	telux::data::IIpFilter Class Reference	688
7.13.1	Constructors and Destructors	689
7.13.1.1	~IIpFilter	689
7.13.2	Member Function Documentation	689
7.13.2.1	getIPv4Info	689
7.13.2.2	setIPv4Info	689
7.13.2.3	getIPv6Info	689
7.13.2.4	setIPv6Info	690
7.13.2.5	getIpProtocol	690
7.13.2.6	getIpFamily	690
7.14	telux::loc::ILocationConfigurator Class Reference	690
7.14.1	Member Typedef Documentation	692
7.14.1.1	GetMinGpsWeekCallback	692
7.14.1.2	GetMinSVElevationCallback	693
7.14.1.3	GetRobustLocationCallback	693
7.14.1.4	GetXtraStatusCallback	693
7.15	telux::loc::ILocationManager Class Reference	694
7.15.1	Member Typedef Documentation	695
7.15.1.1	GetEnergyConsumedCallback	695
7.15.1.2	GetTerrestrialInfoCallback	695
7.16	telux::common::IServiceStatusListener Class Reference	695
7.16.1	Constructors and Destructors	695
7.16.1.1	~IServiceStatusListener	696
7.16.2	Member Function Documentation	696
7.16.2.1	onServiceStatusChange	696
7.17	telux::data::ITcpFilter Class Reference	696
7.17.1	Constructors and Destructors	696
7.17.1.1	~ITcpFilter	696
7.17.2	Member Function Documentation	696
7.17.2.1	getTcpInfo	696
7.17.2.2	setTcpInfo	697
7.18	telux::data::IUDPFilter Class Reference	697
7.18.1	Constructors and Destructors	697
7.18.1.1	~IUDPFilter	697
7.18.2	Member Function Documentation	697
7.18.2.1	getUdpInfo	697
7.18.2.2	setUdpInfo	698

7.19	telux::tel::Polygon Class Reference	698
7.19.1	Constructors and Destructors	698
7.19.1.1	Polygon	698
7.19.2	Member Function Documentation	698
7.19.2.1	getVertices	699
7.20	telux::tel::WarningAreaInfo Class Reference	699
7.20.1	Constructors and Destructors	699
7.20.1.1	WarningAreaInfo	699
7.20.2	Member Function Documentation	699
7.20.2.1	getGeoFenceMaxWaitTime	699
7.20.2.2	getGeometries	700

List of Figures

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

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

1 Introduction

1.1 Purpose

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

1.2 Scope

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

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

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

1.3 Conventions

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

```
#include
```

Parameter directions are indicated as follows:

Parameters

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

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

1.4 SDK Versioning

The following convention is used for versioning the SDK releases

SDK version (major.minor.patch)

SDK_VERSION = 1.0.0

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

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

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

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

1.5 Public API Status

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

as follows:

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

2 Functional Overview

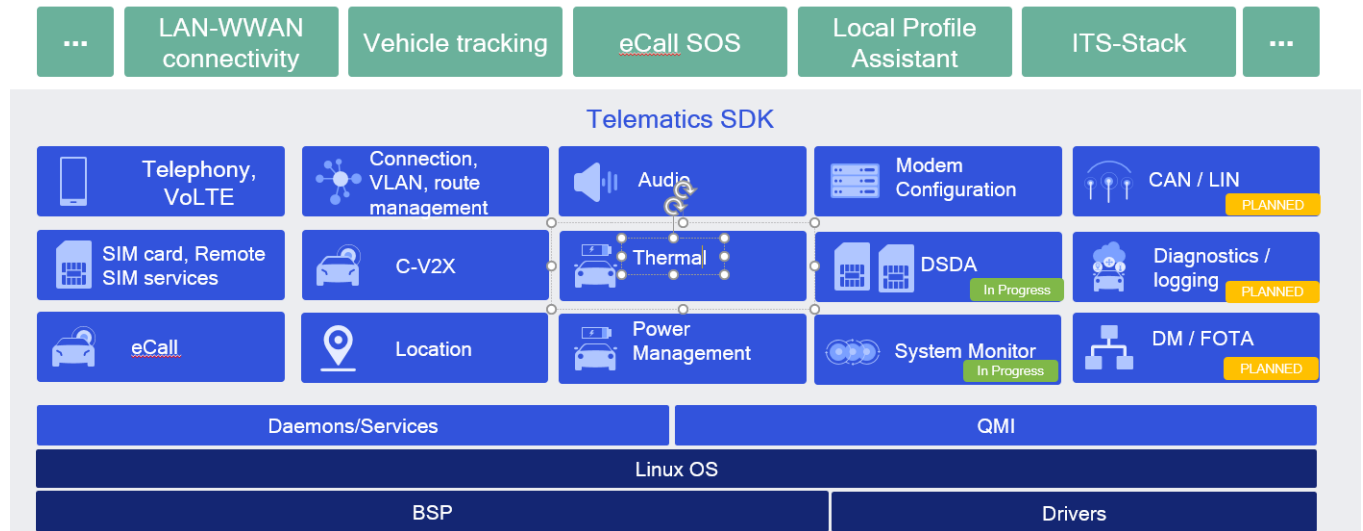


Figure 2-1 SDK-Overview

2.1 Overview

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

Telephony

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

SIM Card Services

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

Location Services

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

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

Connection Management

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

Audio Management

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

Thermal Management

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

Thermal Shutdown 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.

Data Network Management

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

Wlan Management

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

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

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

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

2.2 Features

Telematics SDK provides APIs for the following features:

2.2.1 Call Management

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

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

CallManager and PhoneManager also provides additional functionality such as

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

2.2.2 SMS

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

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

2.2.3 SIM Card Services

The SIM Card operations are performed by CardManager and SapCardManager.

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

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

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

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

2.2.4 Phone Information

Phone APIs of Telematics SDK provides phone related information such as

- Get Service state of phone i.e. EMERGENCY_ONLY, IN_SERVICE and OUT_OF_SERVICE
- Get Radio state of device i.e RADIO_STATE_OFF, RADIO_STATE_ON and RADIO_STATE_UNAVAILABLE
- Retrieve the signal strength corresponding to the technology supported by SIM
- Device Identity
- Set or Request Operating Mode
- Subscription Information

2.2.5 Location Services

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

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

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

2.2.6 Data Services

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

Data Connection Manager APIs provide functionality such as

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

Data Profile Manager APIs provide functionality such as

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

Data Filter Manager APIs provide functionality such as

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

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

Data VLAN Manager APIs provide functionality such as

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

Data Static LAN Manager APIs provide functionality such as

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

Data Firewall Manager APIs provide functionality such as

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

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

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

2.2.7 Network Selection and Serving System Management

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

Network Selection Manager APIs provide functionality such as

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

Serving System Manager APIs provide functionality such as

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

2.2.8 C-V2X

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

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

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

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

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

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

2.2.9 Audio

The Audio subSystem contains of APIs that support Audio operation.

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

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

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

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

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

- Set mute

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

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

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

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

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

- read audio samples

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

- Start loopback
- Stop loopback

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

- Play tone
- Stop tone

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

- Convert one audio format to another

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

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

Example: `DEVICE_TYPE_XXXX`

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

In current release it is mapped per below table.

Current Device Mapping Table:

SDK Audio Device Representation	Audio HAL Representation
DEVICE_TYPE_SPEAKER	AUDIO_DEVICE_OUT_SPEAKER
DEVICE_TYPE_MIC	AUDIO_DEVICE_IN_BACK_MIC

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

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

NUM_DEVICES specifies the number of device types supported

DEVICE_TYPE specifies the SDK type of each device (in comma separated values)

DEVICE_DIR specifies the device direction for each device in order above (in comma separated values)

AHAL_DEVICE_TYPE specifies the mapped Audio HAL type of each device (in comma separated values)

Example:

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

NUM_DEVICES=6

DEVICE_TYPE=1,2,3,257,258,259

DEVICE_DIR=1,1,1,2,2,2

AHAL_DEVICE_TYPE=2,1,4,2147483776,2147483652,2147483664

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

NOTE FOR SYSTEM INTEGRATORS:

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

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

Example: VOICE_CALL, PLAY, CAPTURE etc

Volume Support Table:

This table captures scenarios where the volume could be modified.

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

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

Mute Support Table:

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

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

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

2.2.10 Thermal Management

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

Thermal Management APIs provide functionality such as

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

2.2.11 Thermal Shutdown Management

Thermal Shutdown Management APIs provide functionality such as

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

2.2.12 TCU Activity Management

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

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

2.2.13 Remote SIM

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

Remote SIM APIs provide functionality such as

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

2.2.14 Modem Config Management

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

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

2.2.15 Wlan Management

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

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

2.3 Security

2.3.1 SELinux

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

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

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

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

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

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

#Granting SDK data client permissions to the application

telux_allow_data(app_t);
```

3 Call Flow Diagrams

3.1 Application initialization call flow

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

Example:

3.1.1 Phone manager initialization

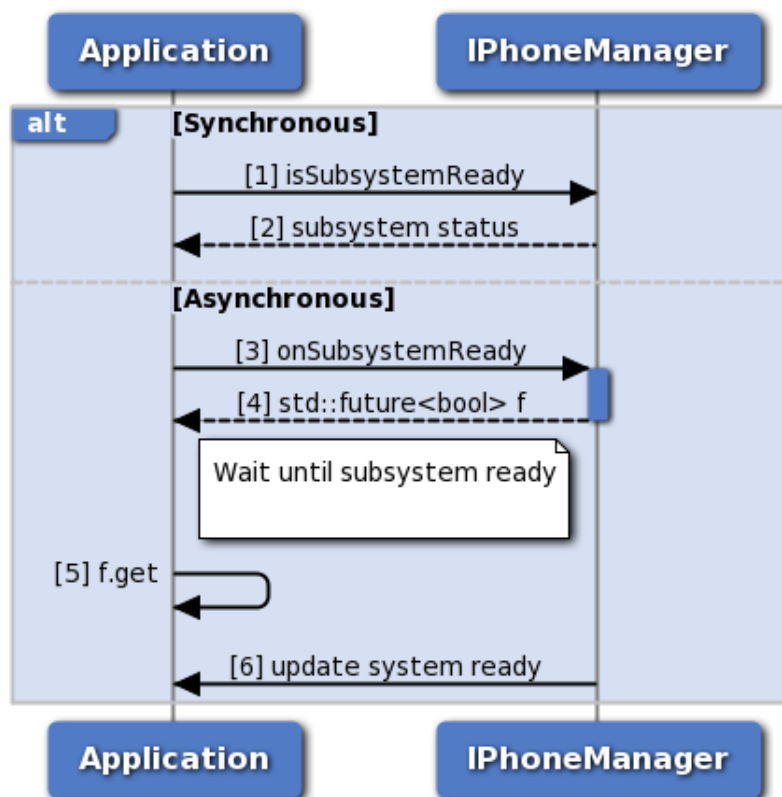


Figure 3-1 Phone manager initialization

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

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

3.2 Telephony

3.2.1 Dial call flow

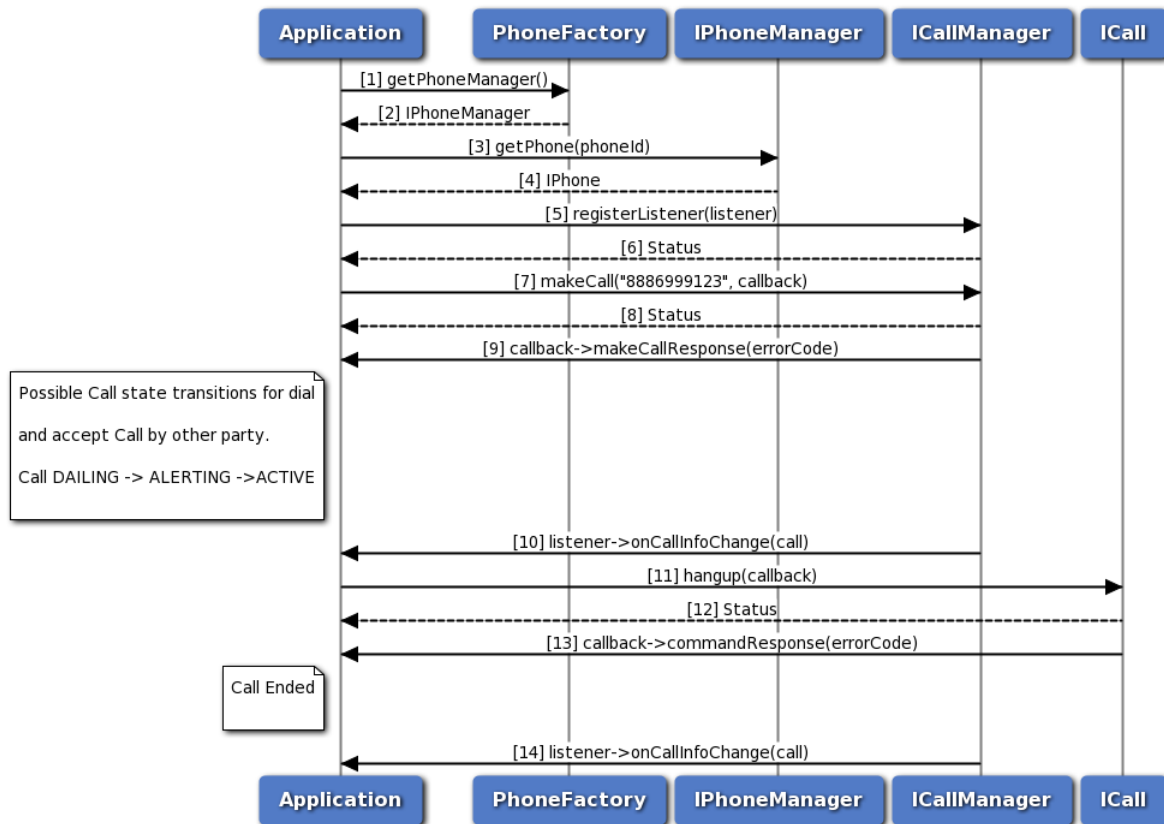


Figure 3-2 Dial call flow

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

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

3.2.2 ECall call flow

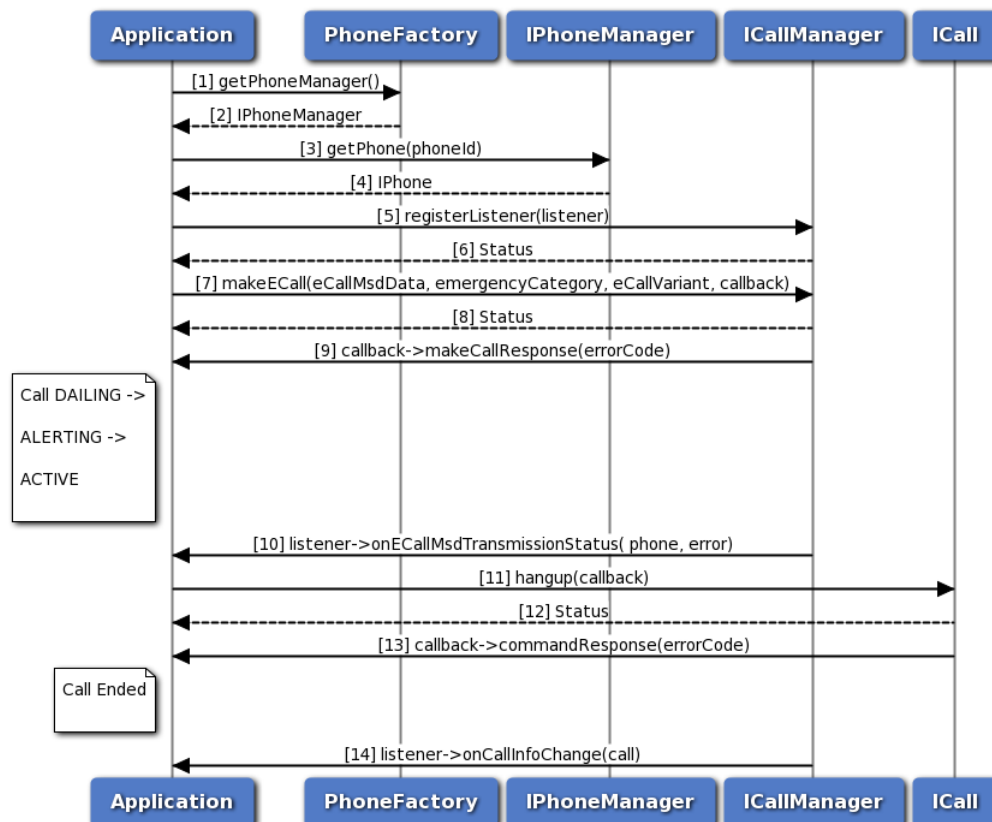


Figure 3-3 ECall call flow

1. The application gets the PhoneManager object using PhoneFactory.

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

3.2.3 Signal strength call flow

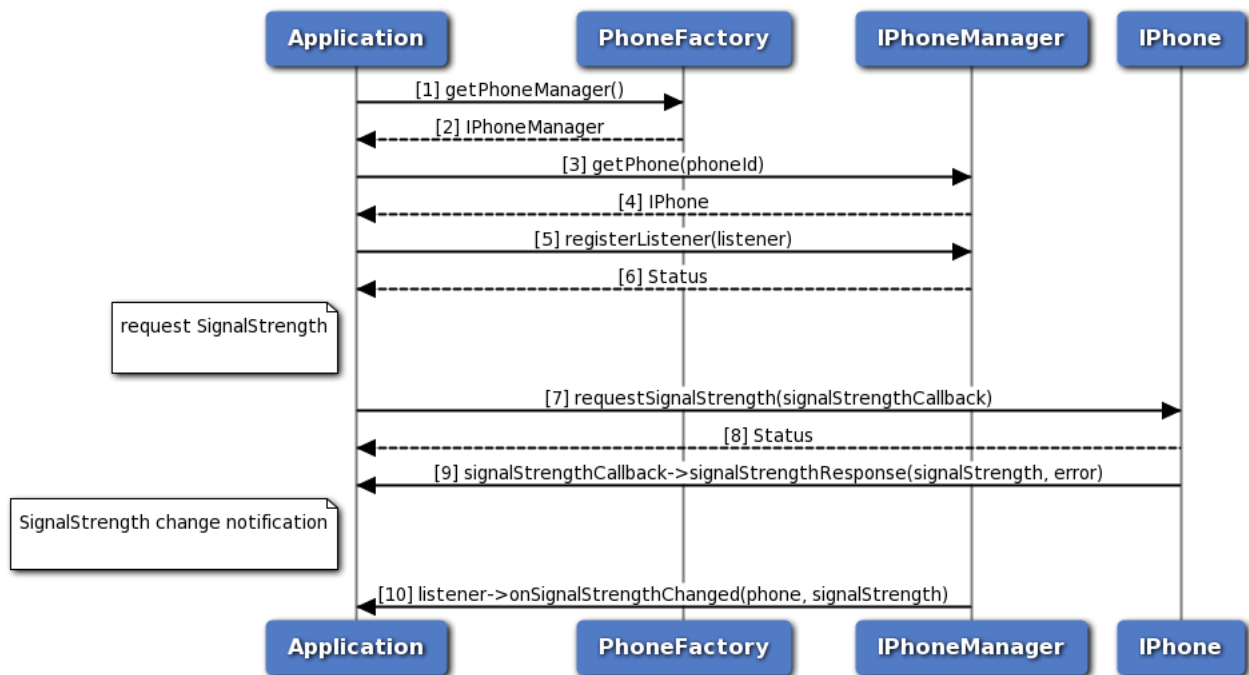


Figure 3-4 Signal strength call flow

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

3.2.4 Answer, Reject, RejectWithSMS call flow

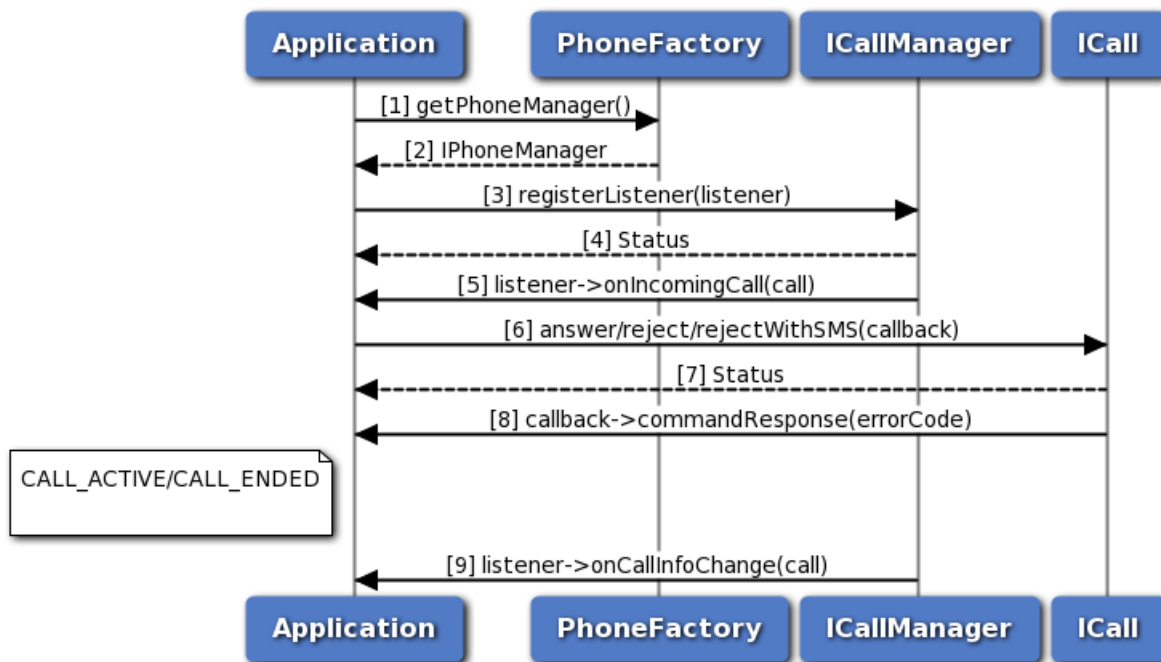


Figure 3-5 Answer, Reject, RejectWithSMS call flow

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

3.2.5 Hold call flow

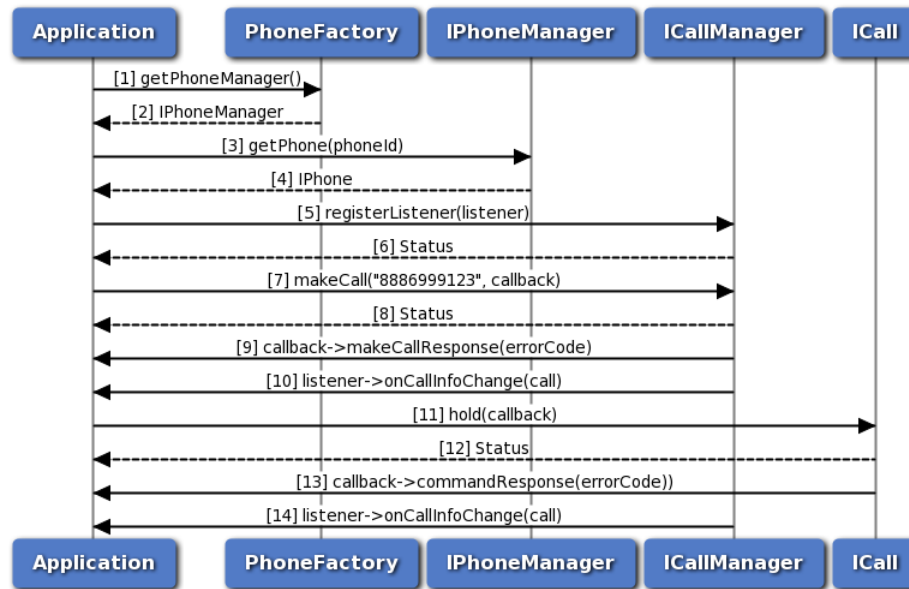


Figure 3-6 Hold call flow

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

3.2.6 Hold, Conference, Swap call flow

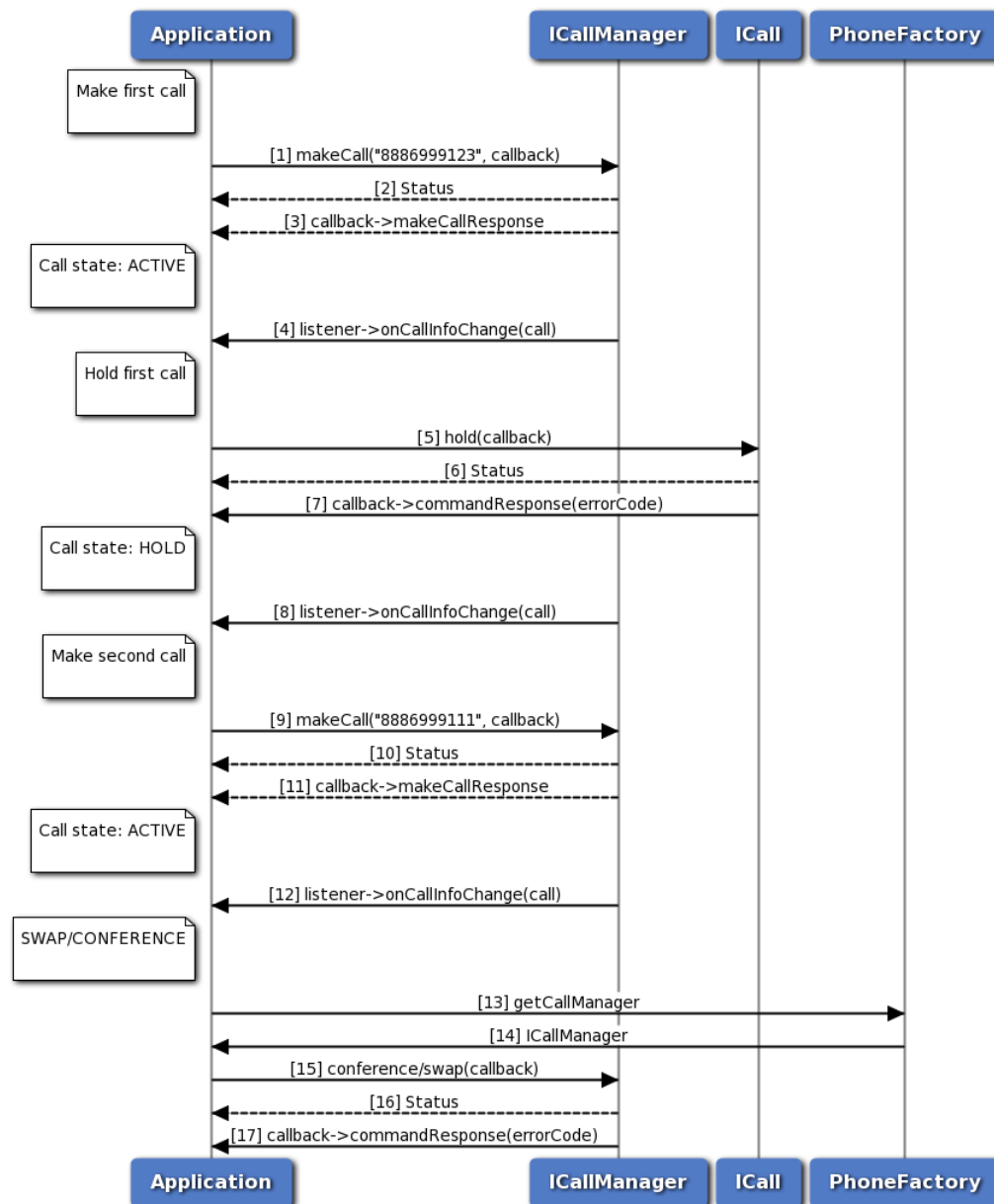


Figure 3-7 Hold, Conference, Swap call flow

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

asynchronous response.

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

3.2.7 SMS call flow

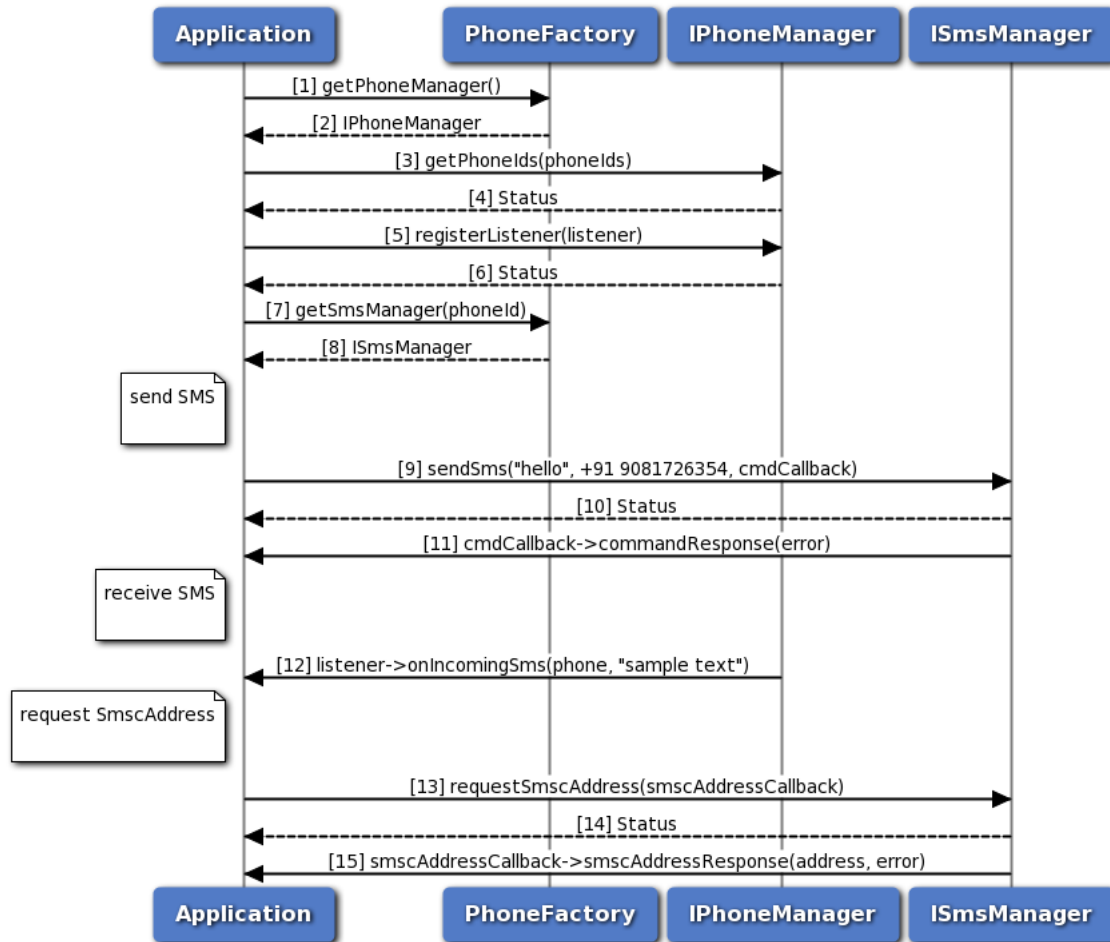


Figure 3-8 SMS call flow

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

in SmsManager.

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

3.2.8 Radio and Service state call flow

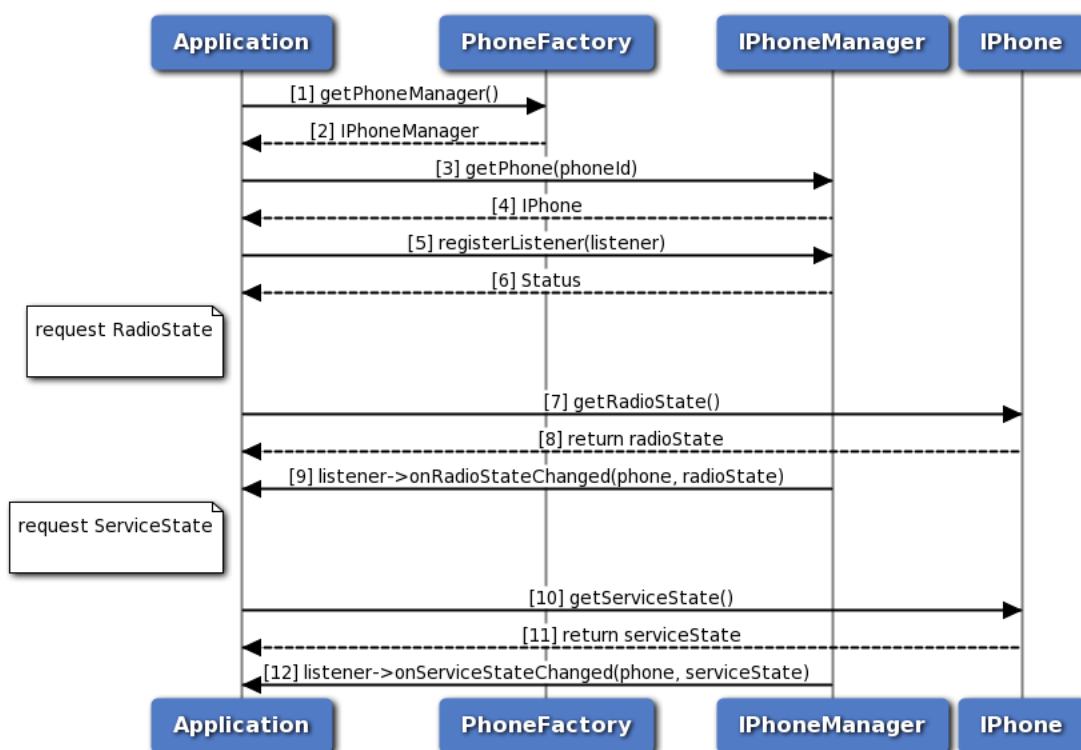


Figure 3-9 Radio and Service state call flow

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

8. The application receives the radio state like RADIO_STATE_ON, OFF or UNAVAILABLE.
9. Application receives a notification when there is a change in radio state.
10. The application request the Phone to get service state.
11. The application receives the service state like IN_SERVICE, OUT_OF_SERVICE, EMERGENCY_ONLY or RADIO_OFF.
12. Application receives a notification when there is a change in service state.

3.2.9 Network Selection Manager call flow

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

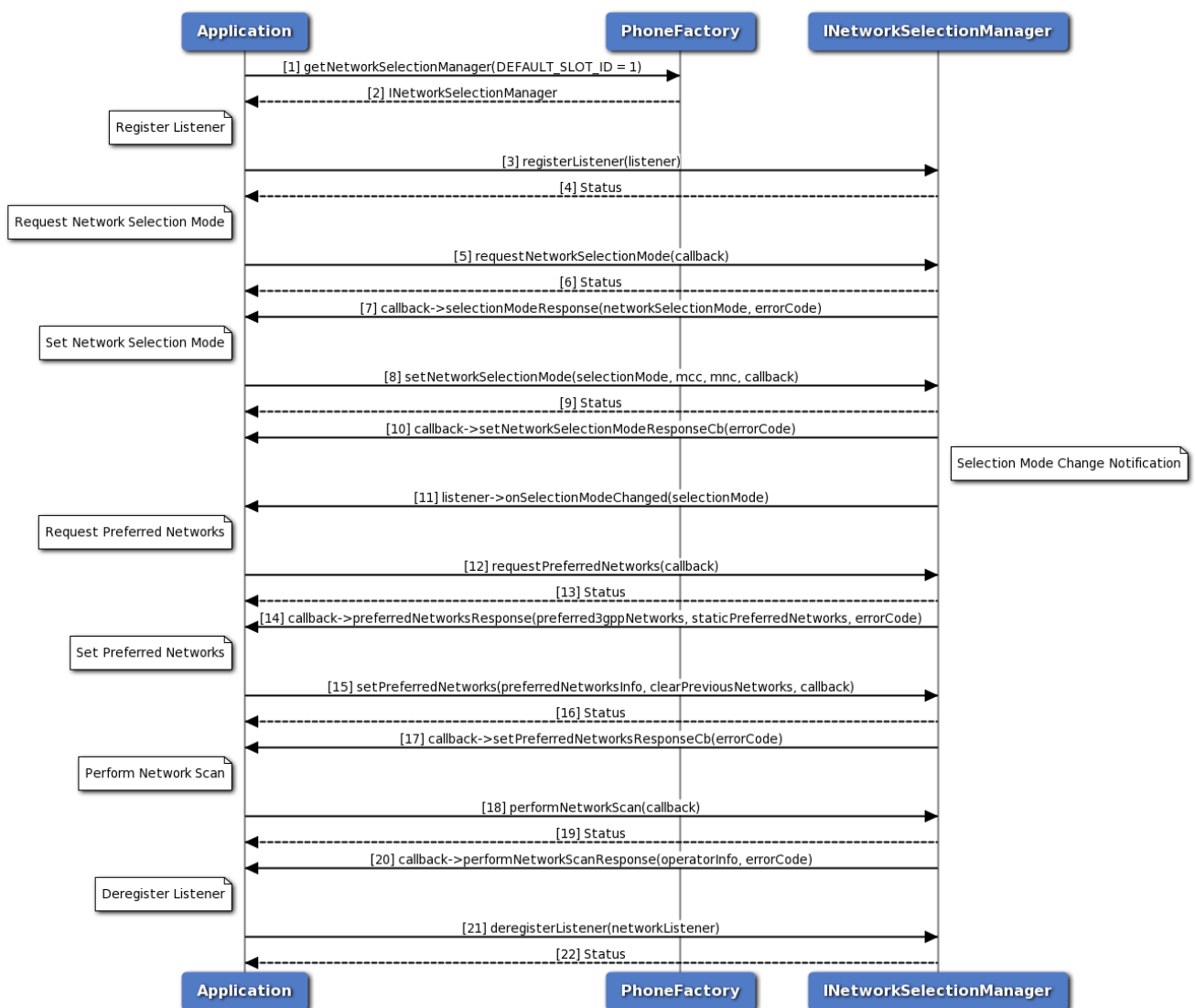


Figure 3-10 Network selection manager call flow

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

3.2.10 Serving System Manager Call Flow

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

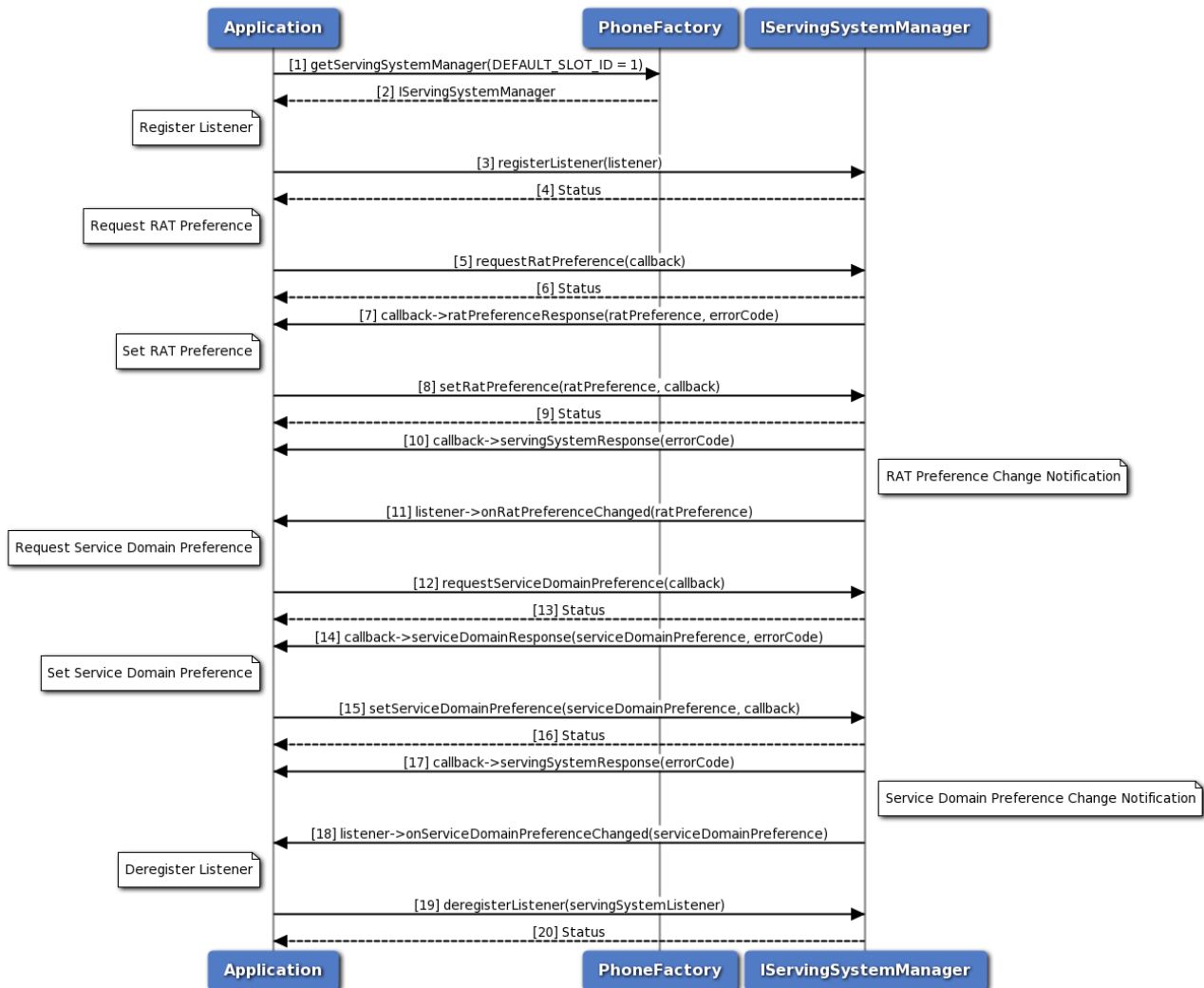


Figure 3-11 Serving System Manager Call Flow

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

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

3.3 Card Services

3.3.1 Get applications call flow

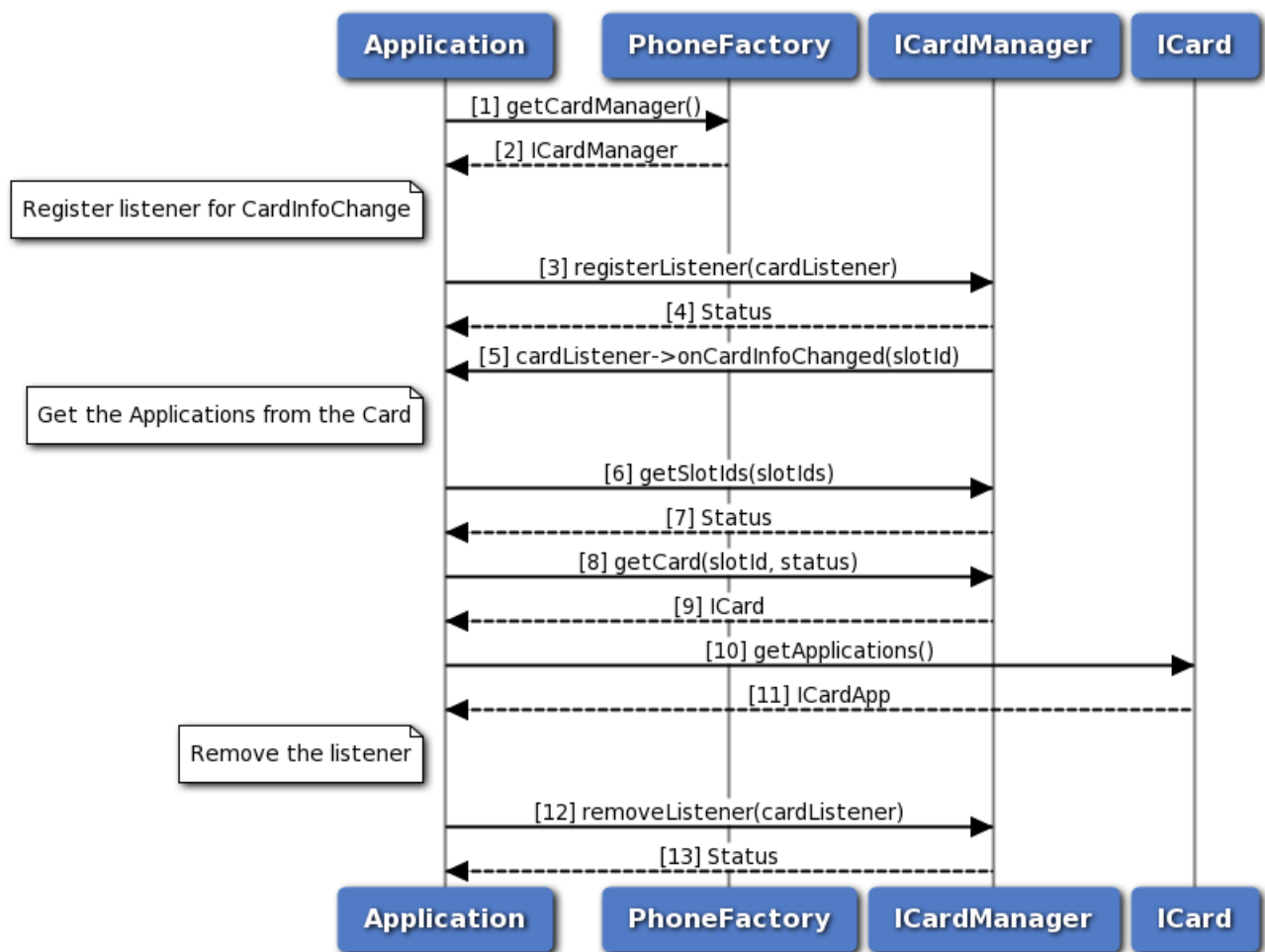


Figure 3-12 Get applications call flow

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

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

3.3.2 Transmit APDU call flow

3.3.2.1 On logical channel

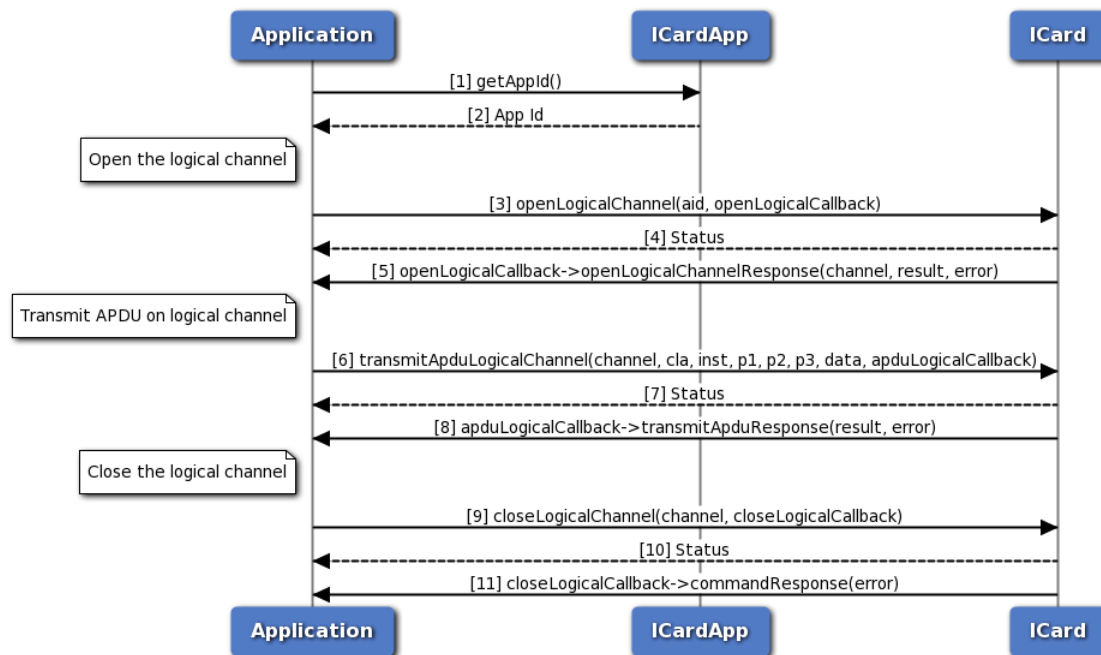


Figure 3-13 On logical channel

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

case of failure.

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

3.3.2.2 On basic channel

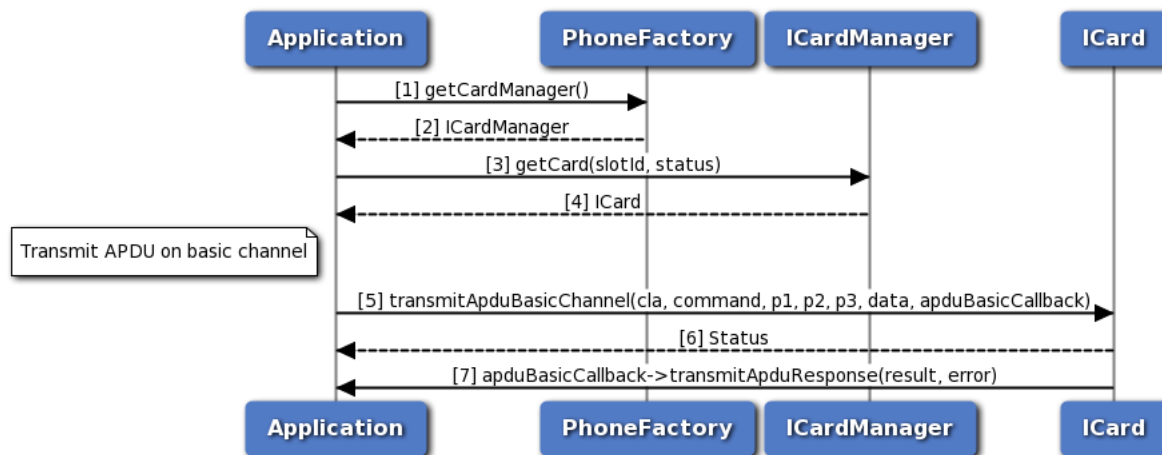


Figure 3-14 On basic channel

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

3.3.3 SAP card manager call flow

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

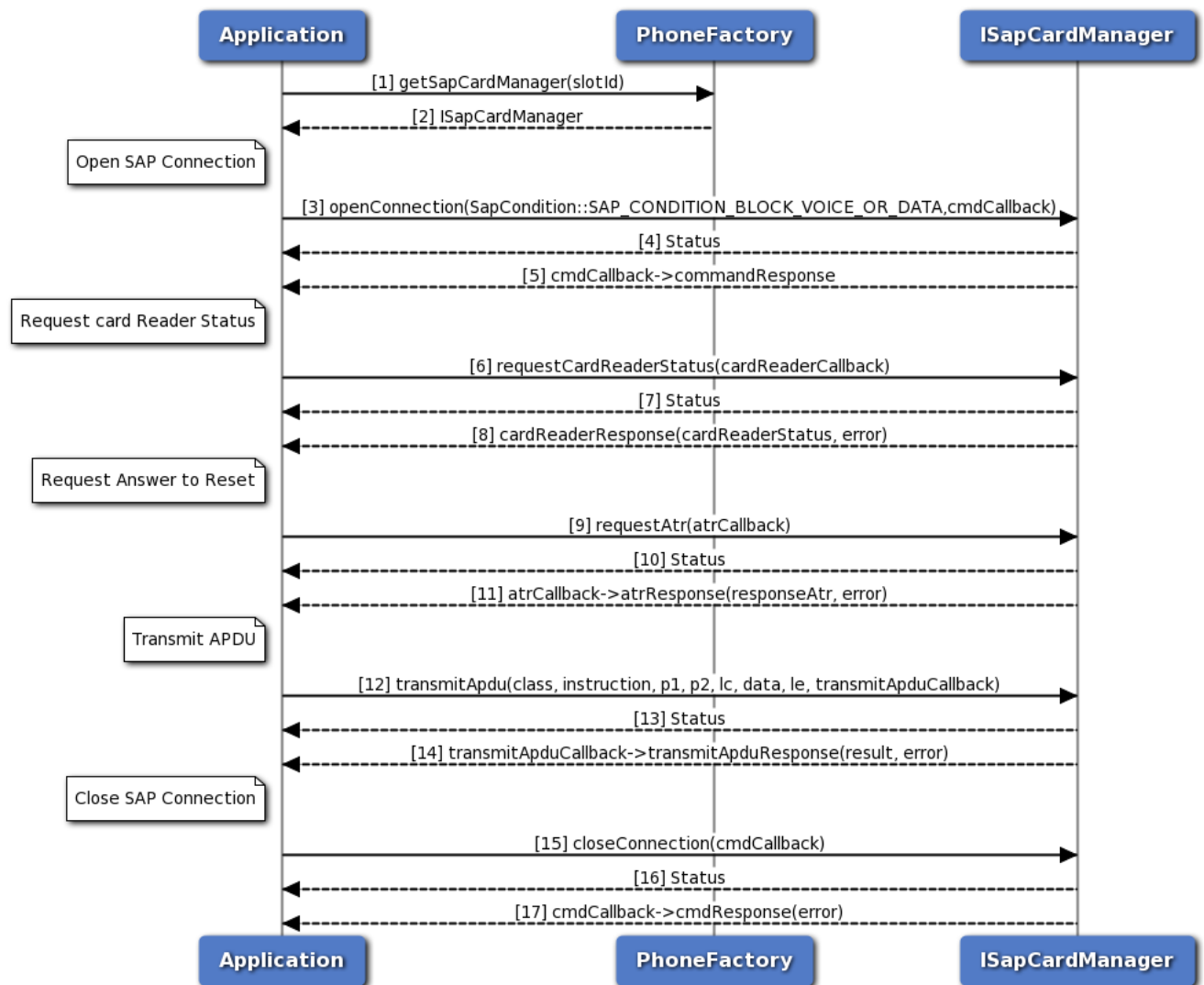


Figure 3-15 Request card reader status, Request ATR, Transmit APDU call flow

{sap_card_operations.png,Request card reader status, Request ATR, Transmit APDU call flow,80,80,Request card reader status, Request ATR, Transmit APDU call flow }

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

6. The application sends request card reader status command and optionally, gets asynchronous response using ICardReaderCallback.
7. The application receives the status i.e. either SUCCESS or FAILED based on execution of requestCardReaderStatus API in SapCardManager.
8. Optionally, the response for card reader status is received by the application.
9. Similarly, the application can send SAP Answer To Reset command and optionally, gets asynchronous response using 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.

3.3.3.2 SIM Turn off, Turn on and Reset call flow

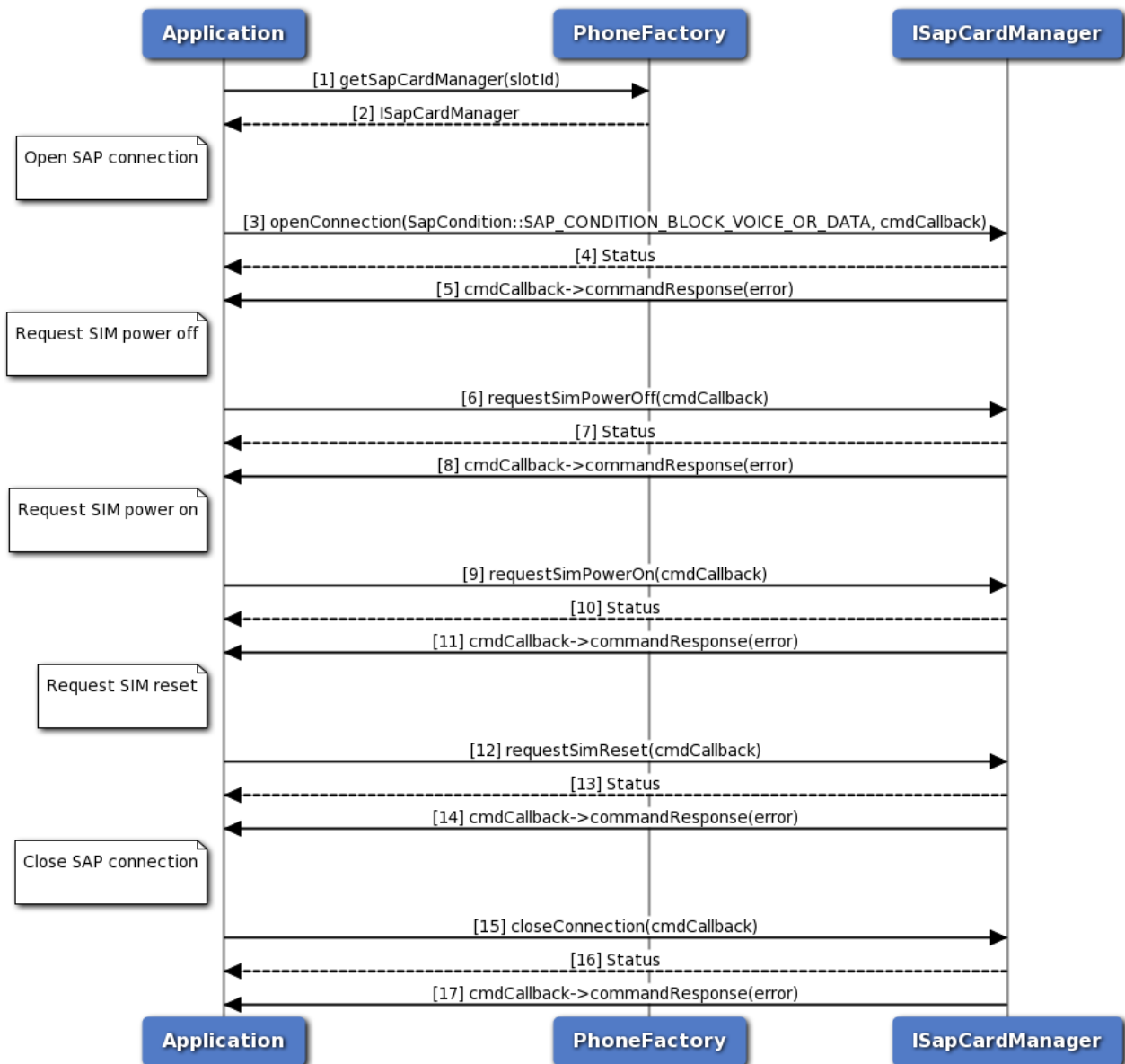


Figure 3-16 SIM Turn off, Turn on and Reset call flow

{sap_mgr_sim_call_flow.png, SIM Turn off, Turn on and Reset call flow, 80, 80, SIM Turn off, Turn on and Reset call flow}

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

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

3.3.4 Subscription Call flow

3.3.4.1 Subscription initialization

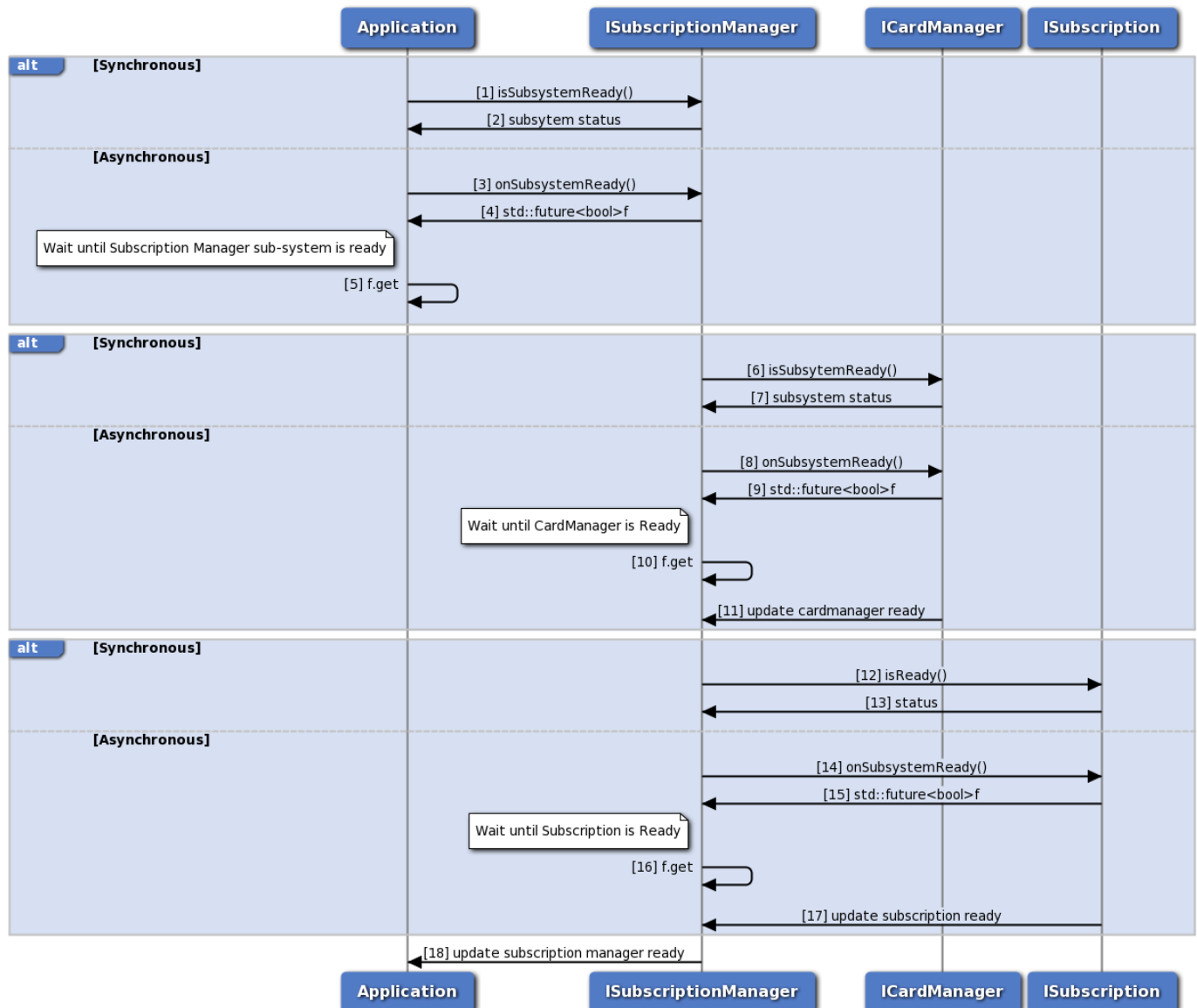


Figure 3-17 Subscription initialization call flow

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

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

3.3.4.2 Subscription call flow

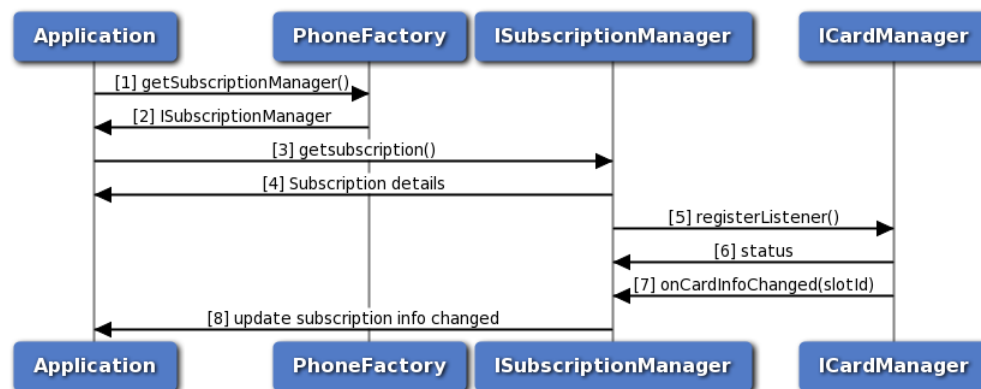


Figure 3-18 Subscription call flow

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

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

3.4 Call flow for location services

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

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

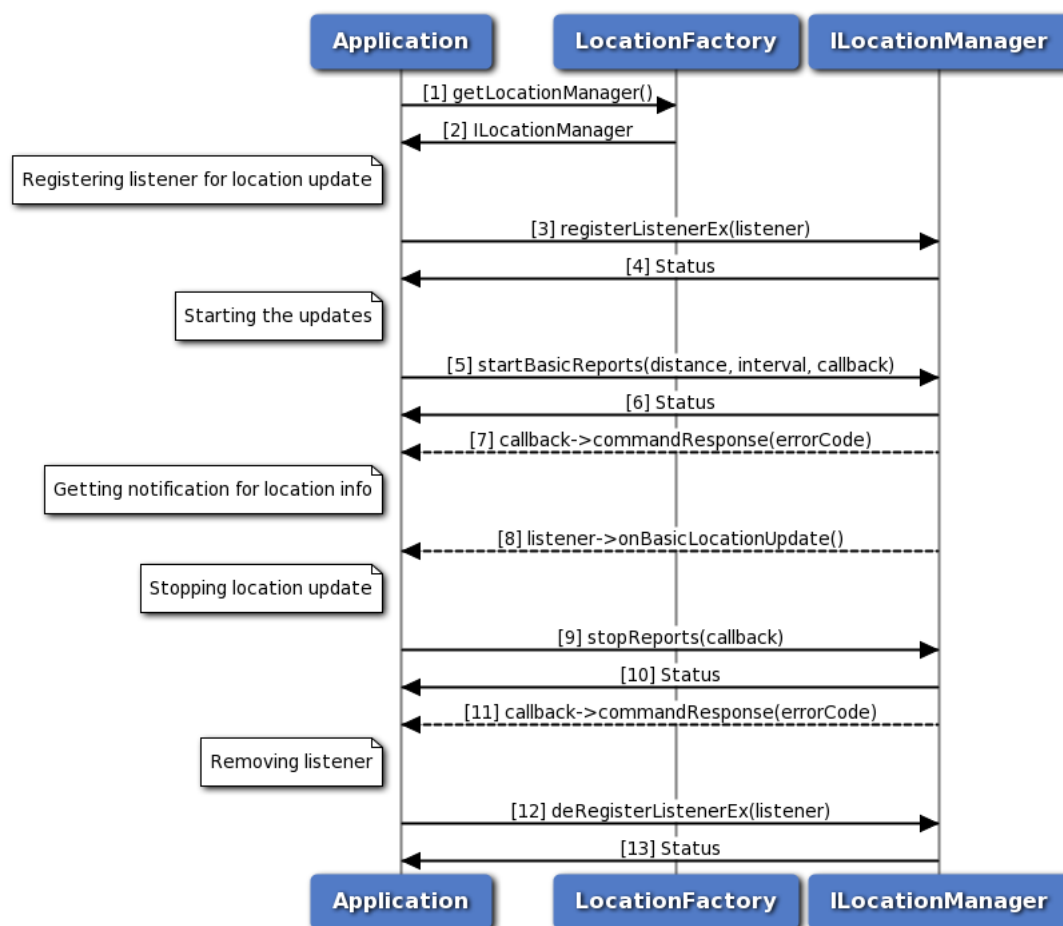


Figure 3-19 Call flow to register/remove listener for generating basic reports

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

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

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

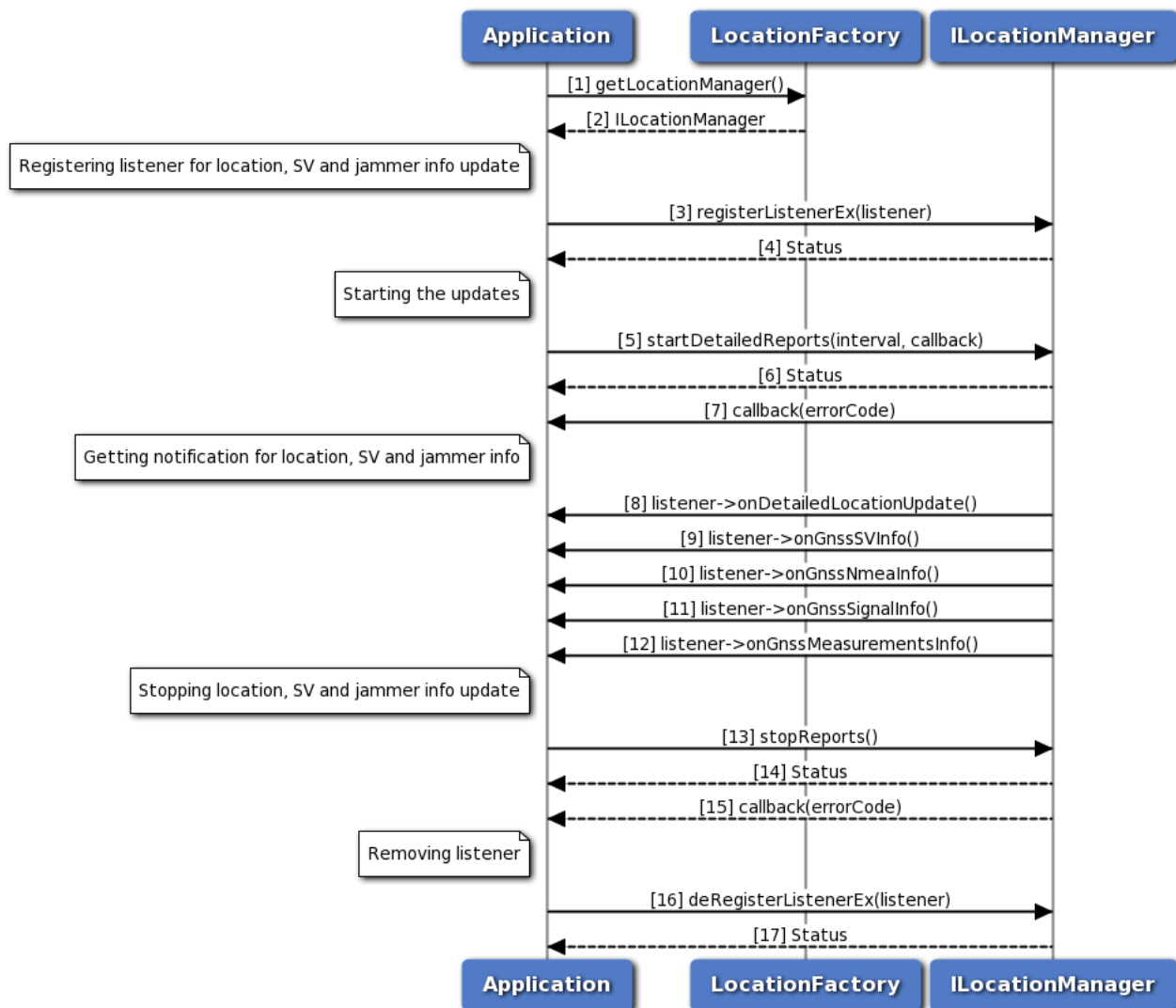


Figure 3-20 Call flow to register/remove listener for generating detailed reports

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

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

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

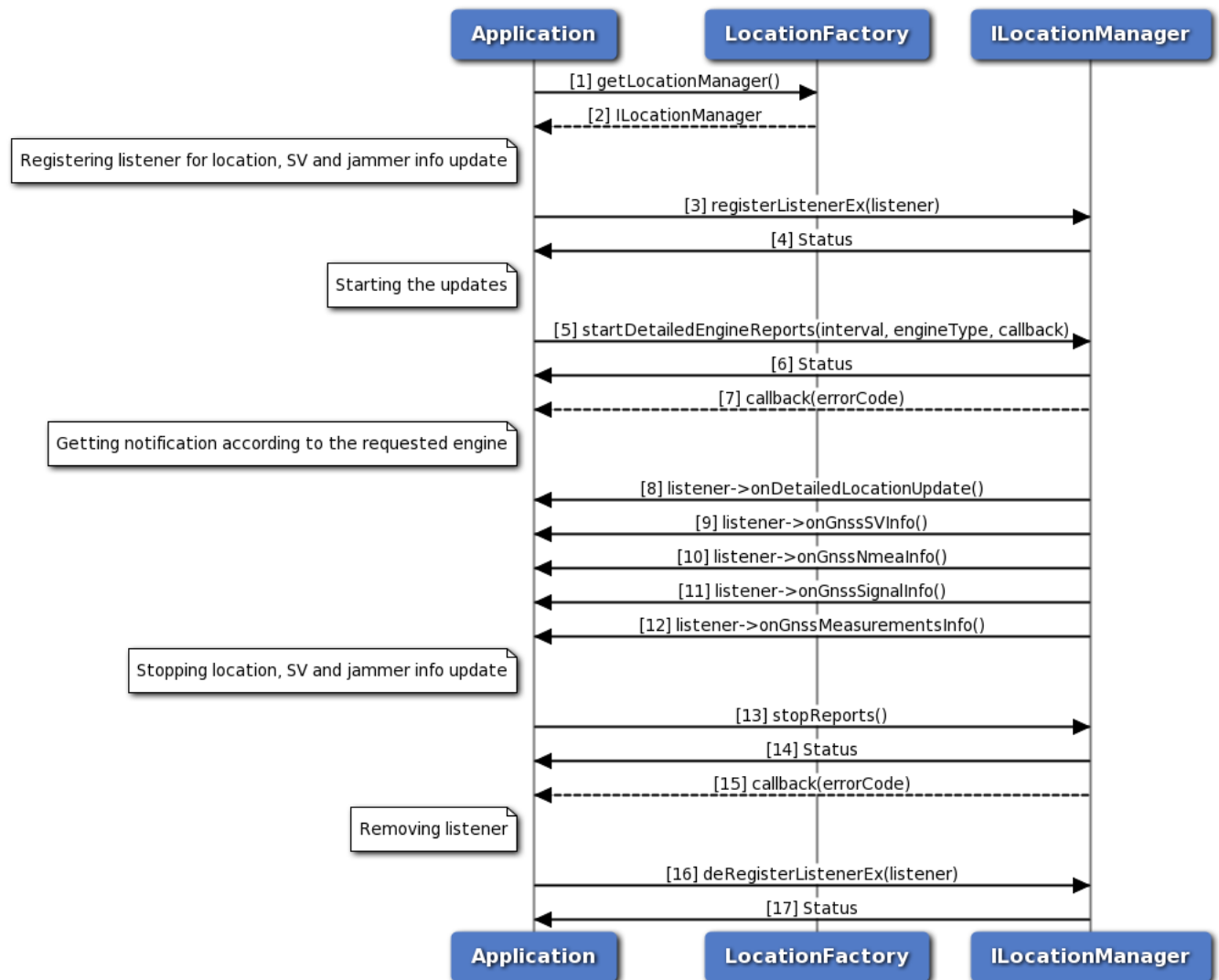


Figure 3-21 Call flow to register/remove listener for generating detailed engine reports

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

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

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

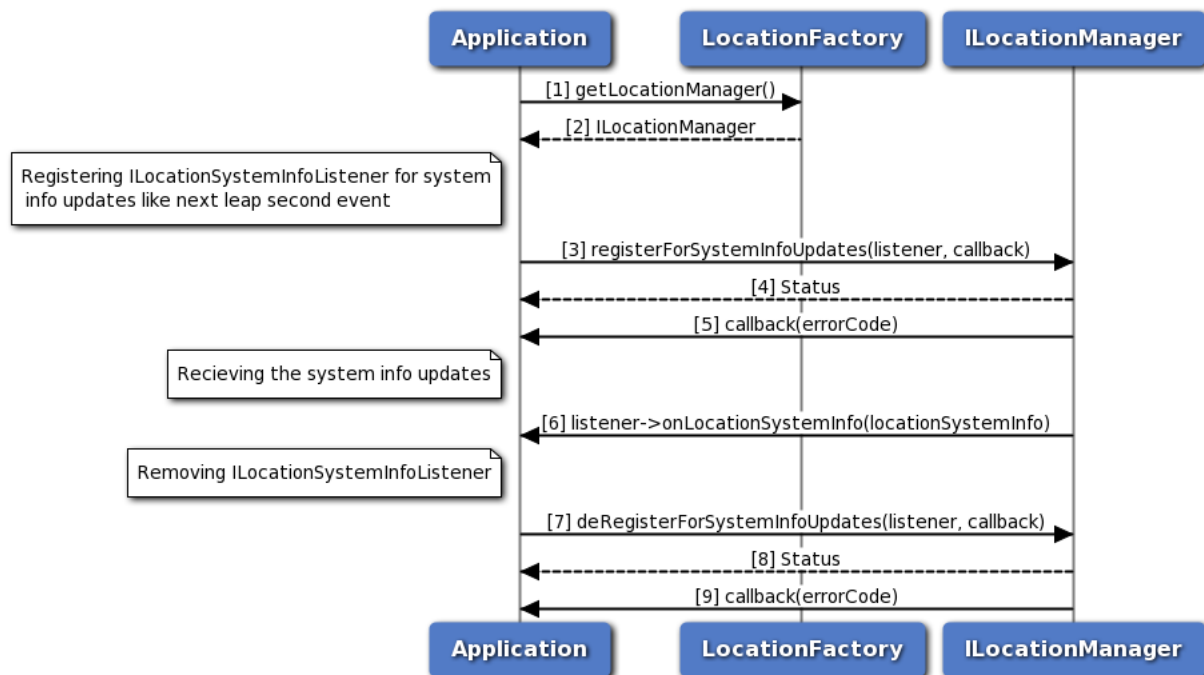


Figure 3-22 Call flow to register/remove listener for system info updates

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

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

3.4.5 Call flow to request energy consumed information

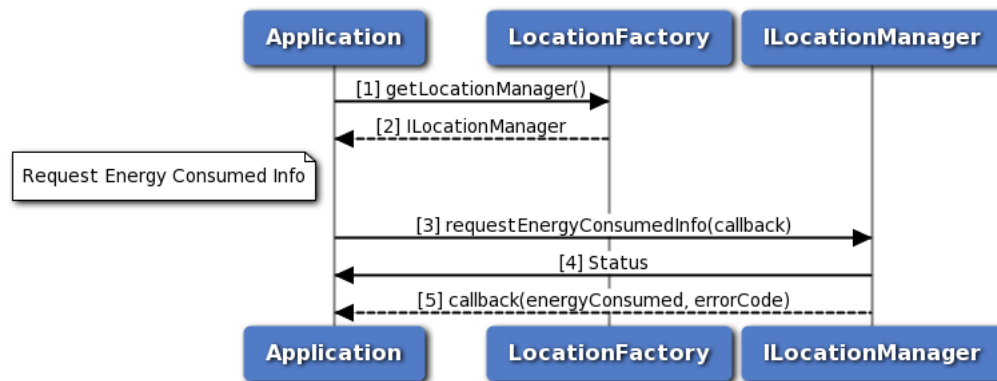


Figure 3-23 Call flow to request energy consumed information

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

3.4.6 Call flow to get terrestrial positioning information

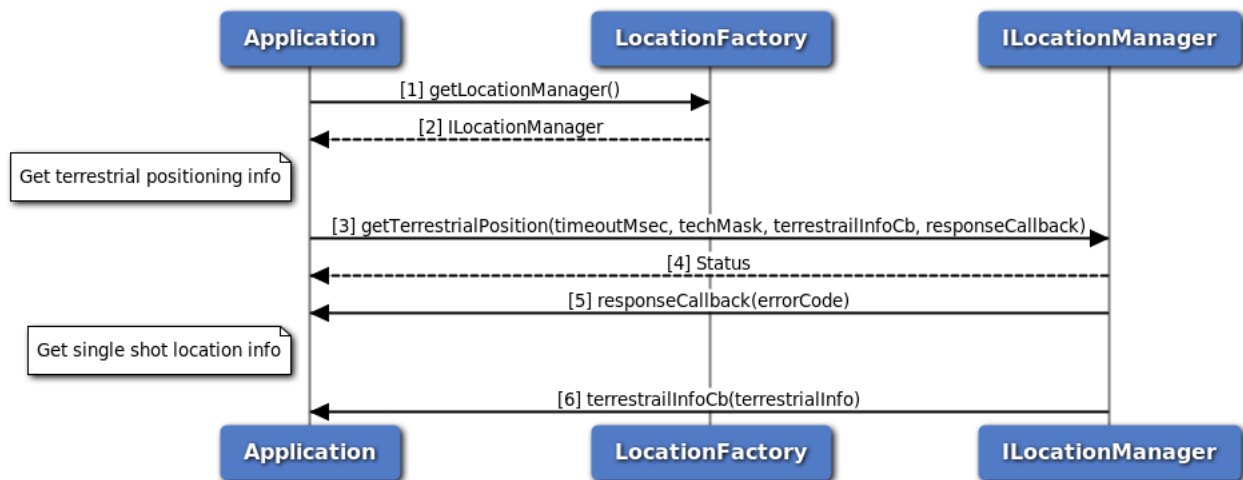


Figure 3-24 Call flow to get terrestrial positioning information

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

3.4.7 Call flow to cancel terrestrial positioning information

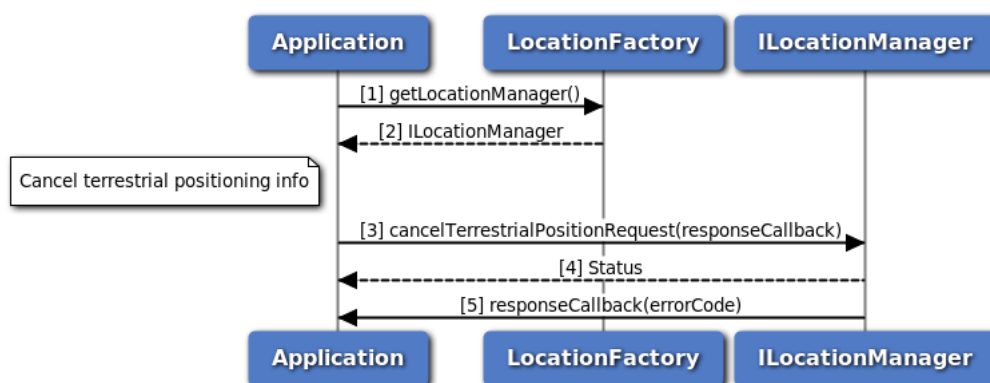


Figure 3-25 Call flow to cancel terrestrial positioning information

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

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

3.4.8 Call flow to enable/disable constraint time uncertainty

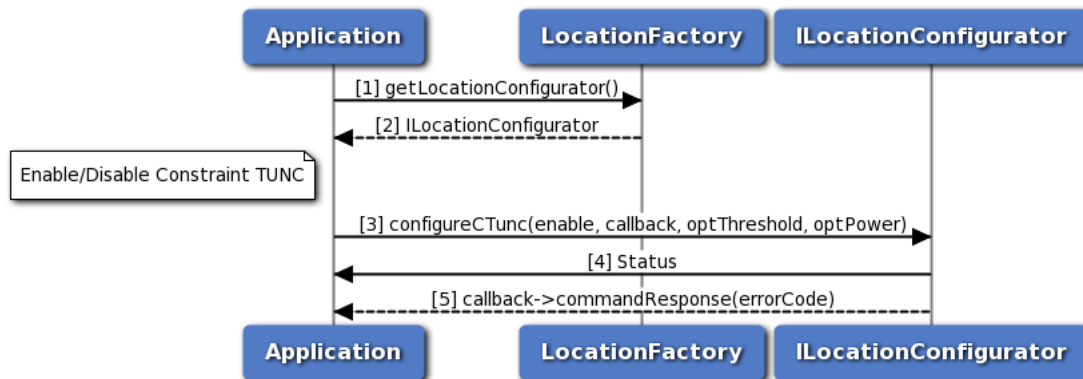


Figure 3-26 Call flow to enable/disable constraint time uncertainty

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

3.4.9 Call flow to enable/disable PACE

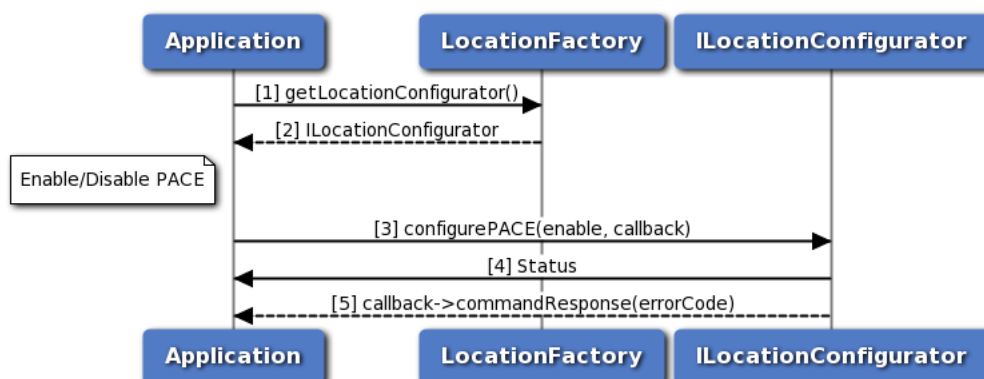


Figure 3-27 Call flow to enable/disable PACE

1. Application requests location factory for location configurator object.

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

3.4.10 Call flow to delete all aiding data

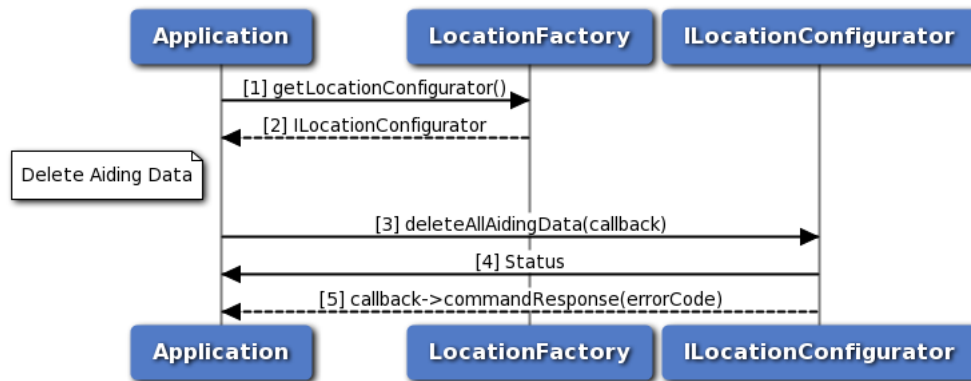


Figure 3-28 Call flow to delete all aiding data

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

3.4.11 Call flow to configure lever arm parameters

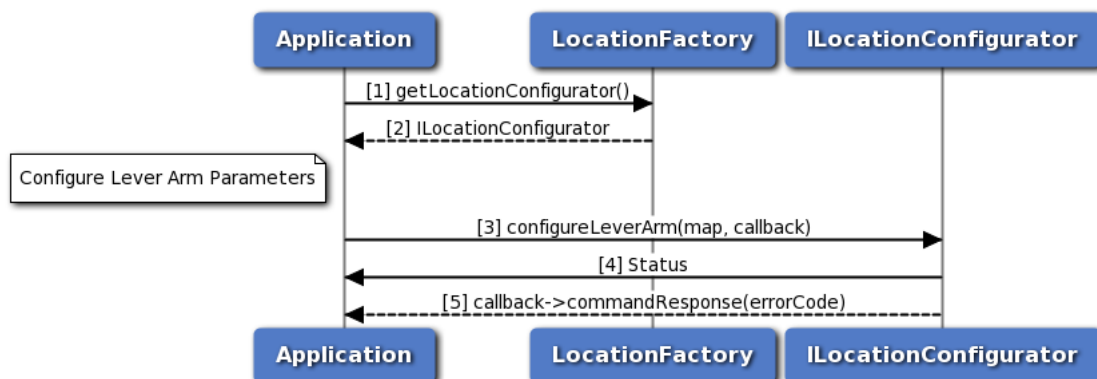


Figure 3-29 Call flow to configure lever arm parameters

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

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

3.4.12 Call flow to configure blacklisted constellations

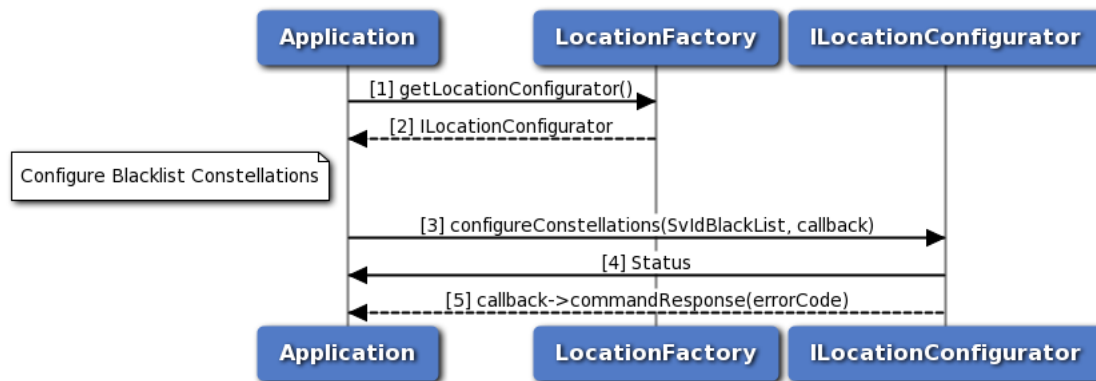


Figure 3-30 Call flow to configure blacklisted constellations

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

3.4.13 Call flow to configure robust location

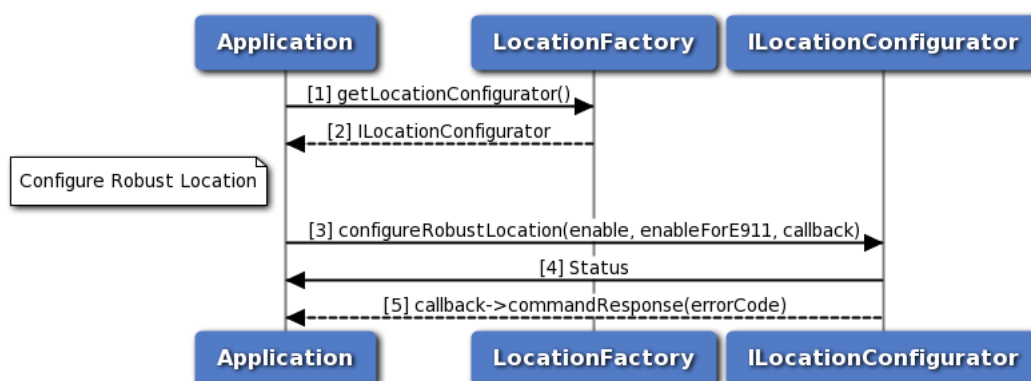


Figure 3-31 Call flow to configure robust location

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

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

3.4.14 Call flow to configure min gps week

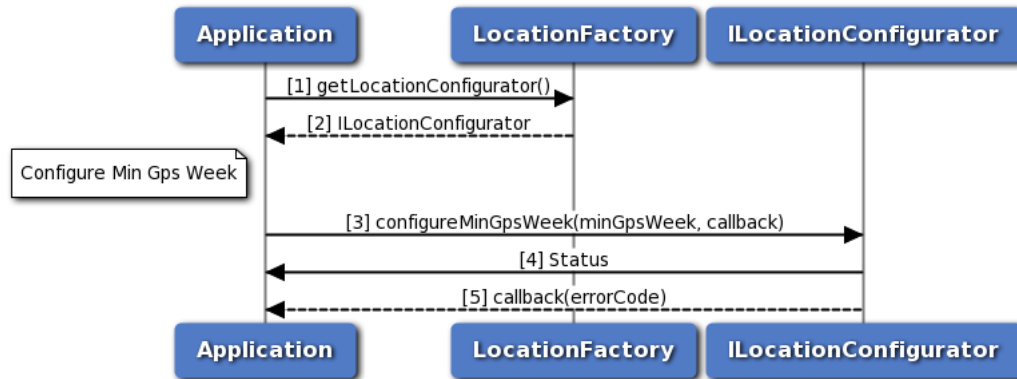


Figure 3-32 Call flow to configure min gps week

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

3.4.15 Call flow to request min gps week

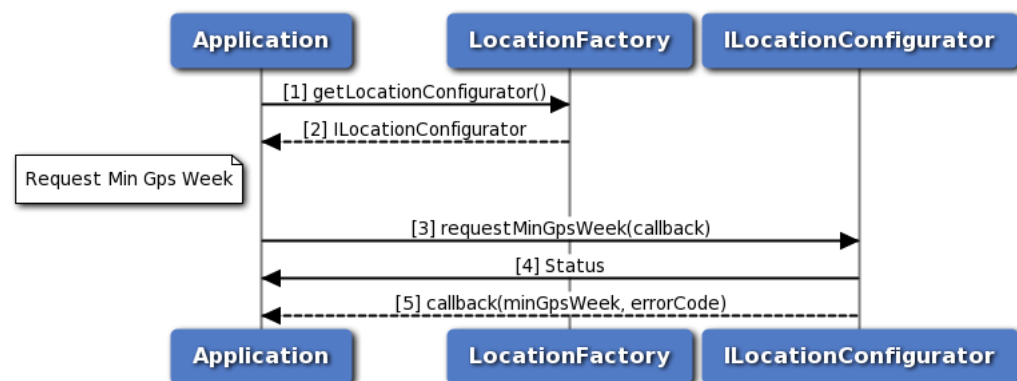


Figure 3-33 Call flow to request min gps week

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

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

3.4.16 Call flow to delete specified data

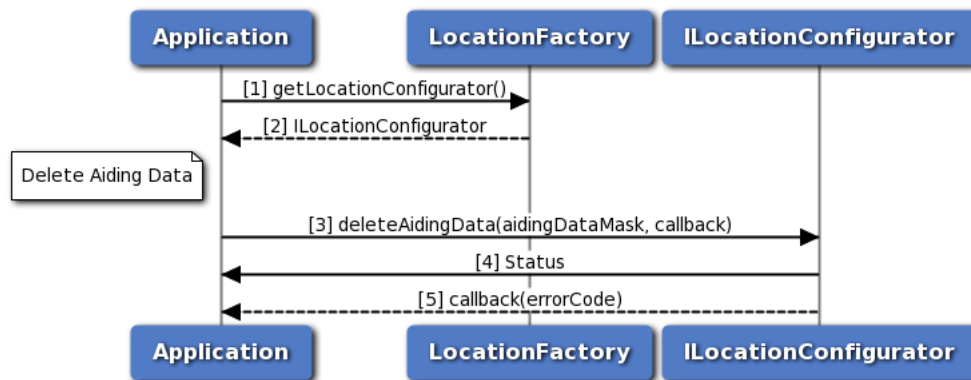


Figure 3-34 Call flow to delete specified data

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

3.4.17 Call flow to configure min sv elevation

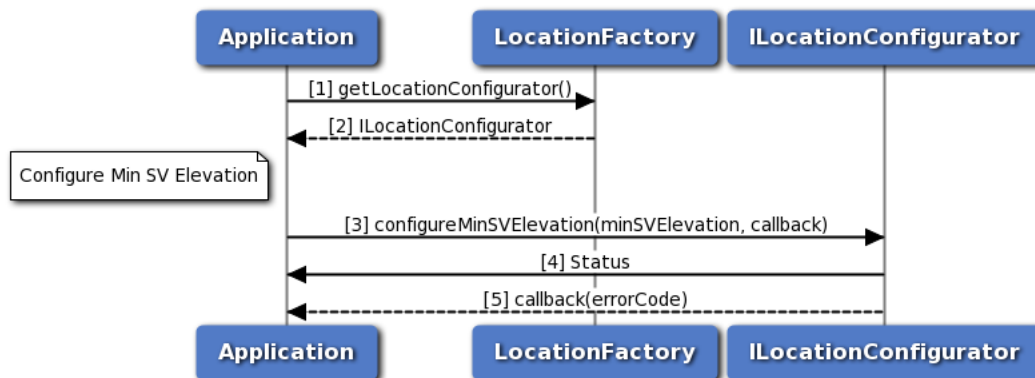


Figure 3-35 Call flow to configure min sv elevation

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

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

3.4.18 Call flow to request min sv elevation

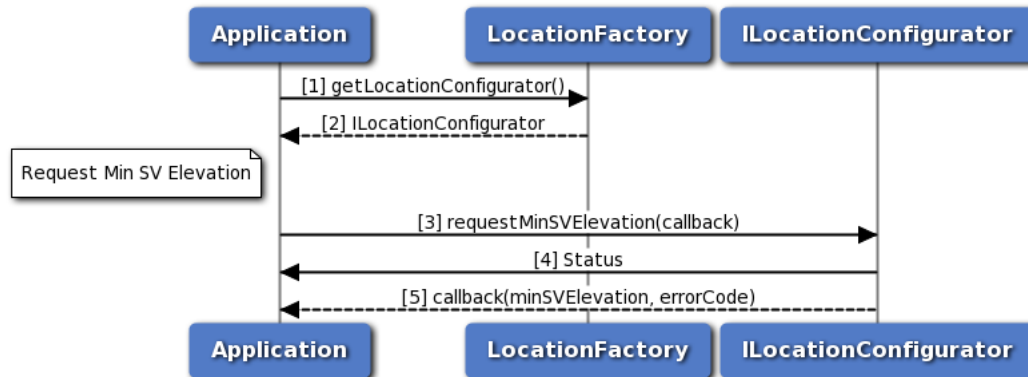


Figure 3-36 Call flow to request min sv elevation

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

3.4.19 Call flow to request robust location

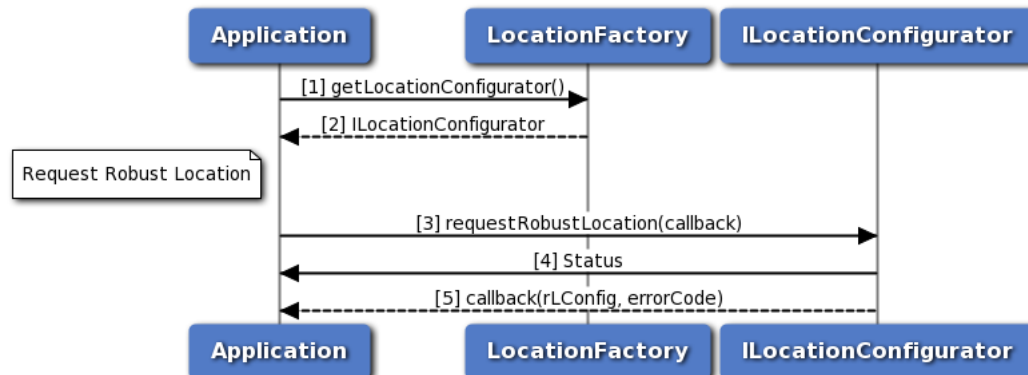


Figure 3-37 Call flow to request robust location

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

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

3.4.20 Call flow to configure dead reckoning engine

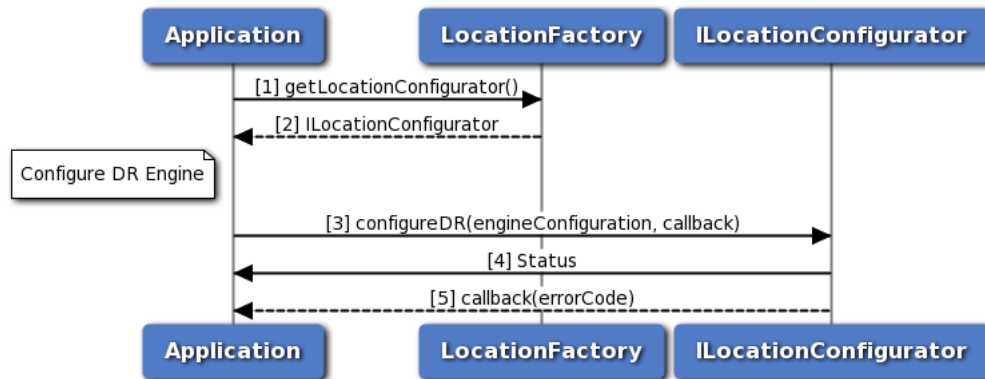


Figure 3-38 Call flow to configure dead reckoning engine

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

3.4.21 Call flow to provide user consent for terrestrial positioning

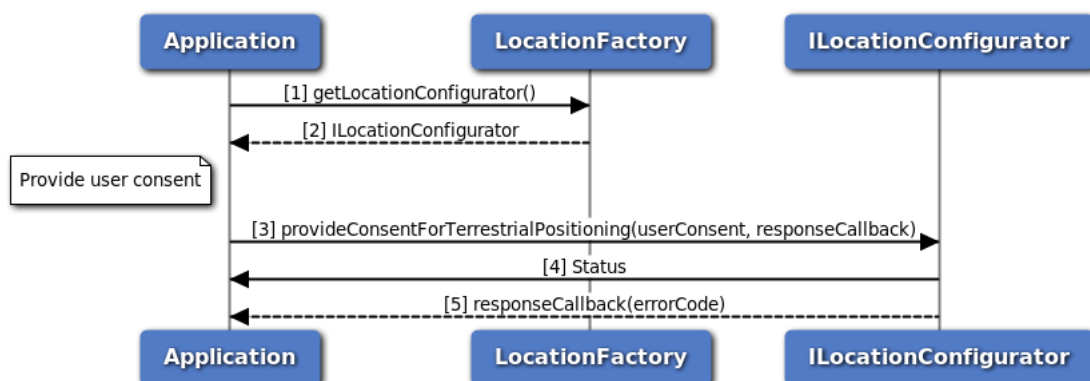


Figure 3-39 Call flow to provide user consent for terrestrial positioning

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

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

3.4.22 Call flow to represent Xtra Feature

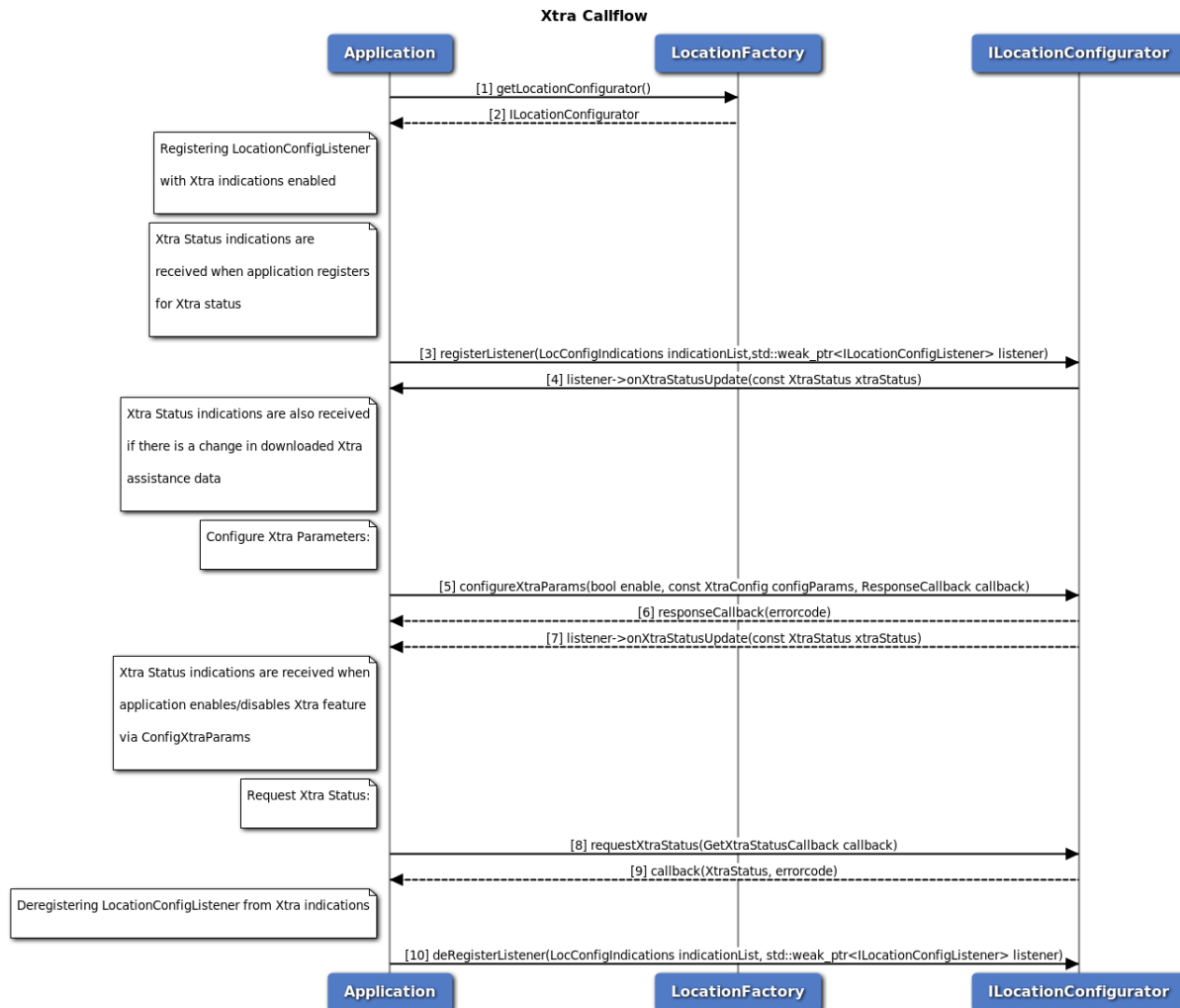


Figure 3-40 Call flow to represent Xtra Feature

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

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

3.5 Data

3.5.1 Start/Stop for data connection manager call flow

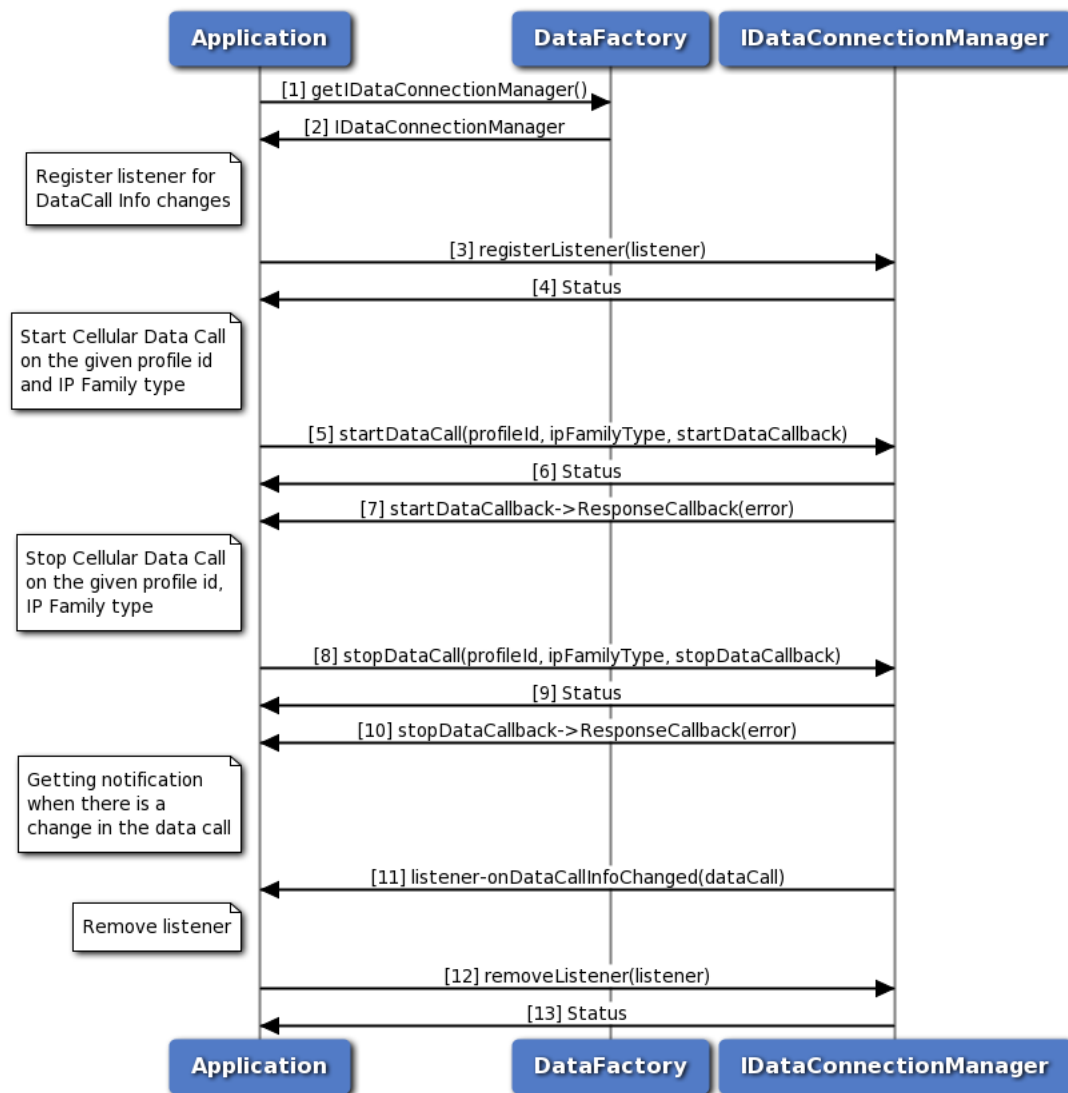


Figure 3-41 Start/Stop for data connection manager call flow

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

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

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

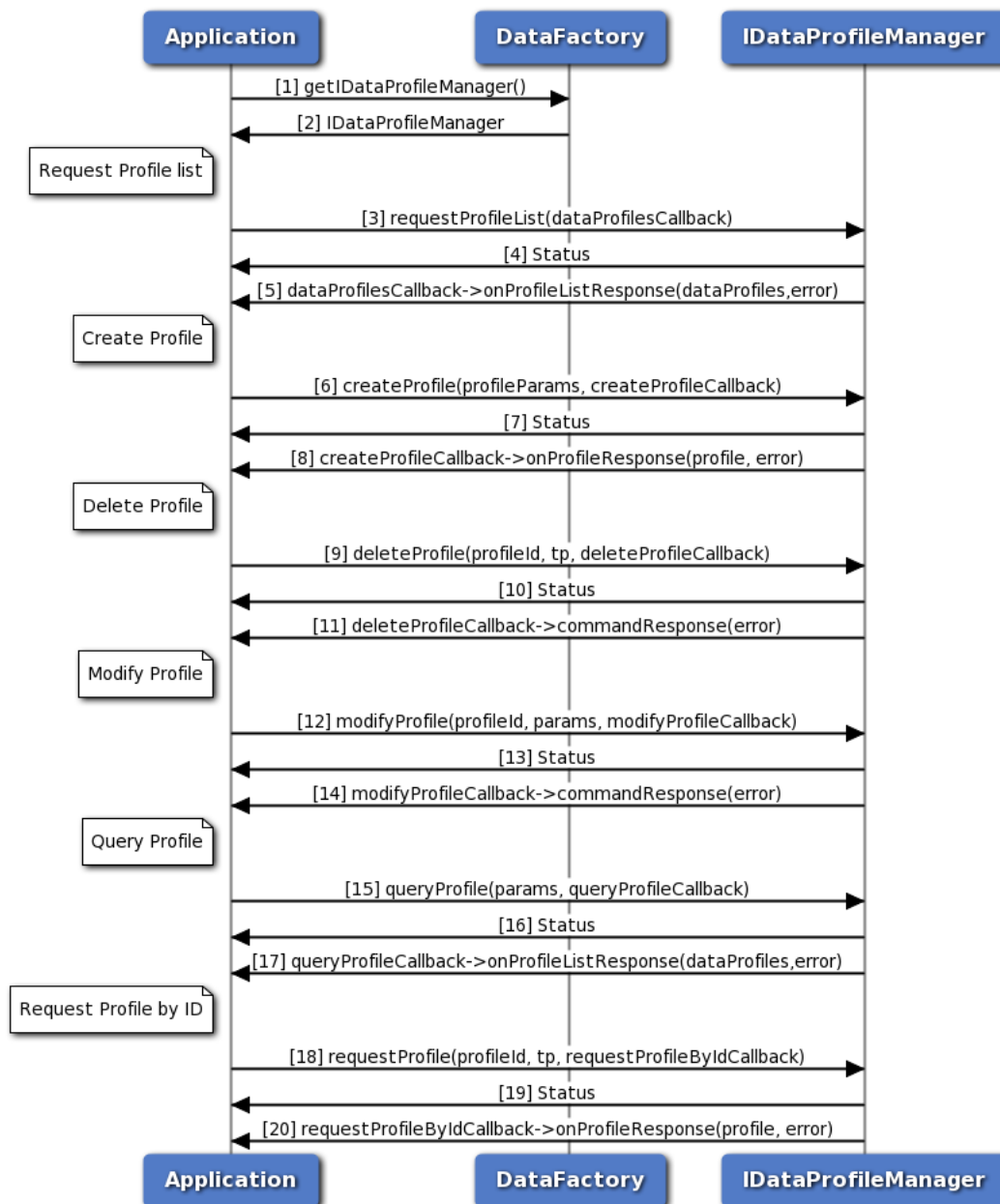


Figure 3-42 Request/Create/Delete/Modify for data profile manager call flow

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

dataProfilesCallback.

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

3.5.3 Data Filter Manager Call Flow

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

3.5.3.1 Call flow to Set/Get data filter mode

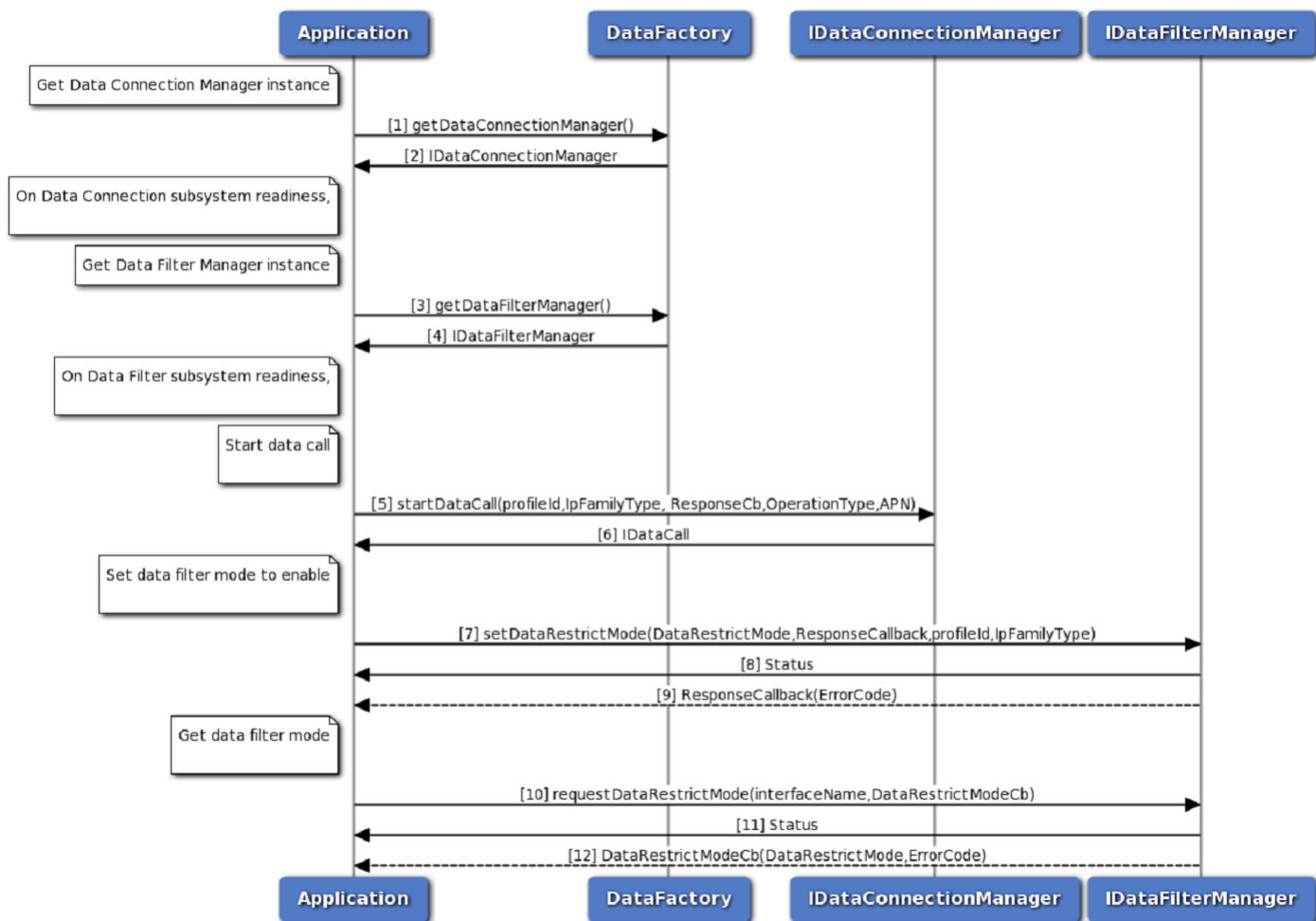


Figure 3-43 Get/Set data filter mode call flow

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

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

3.5.3.2 Call flow to Add data restrict filter

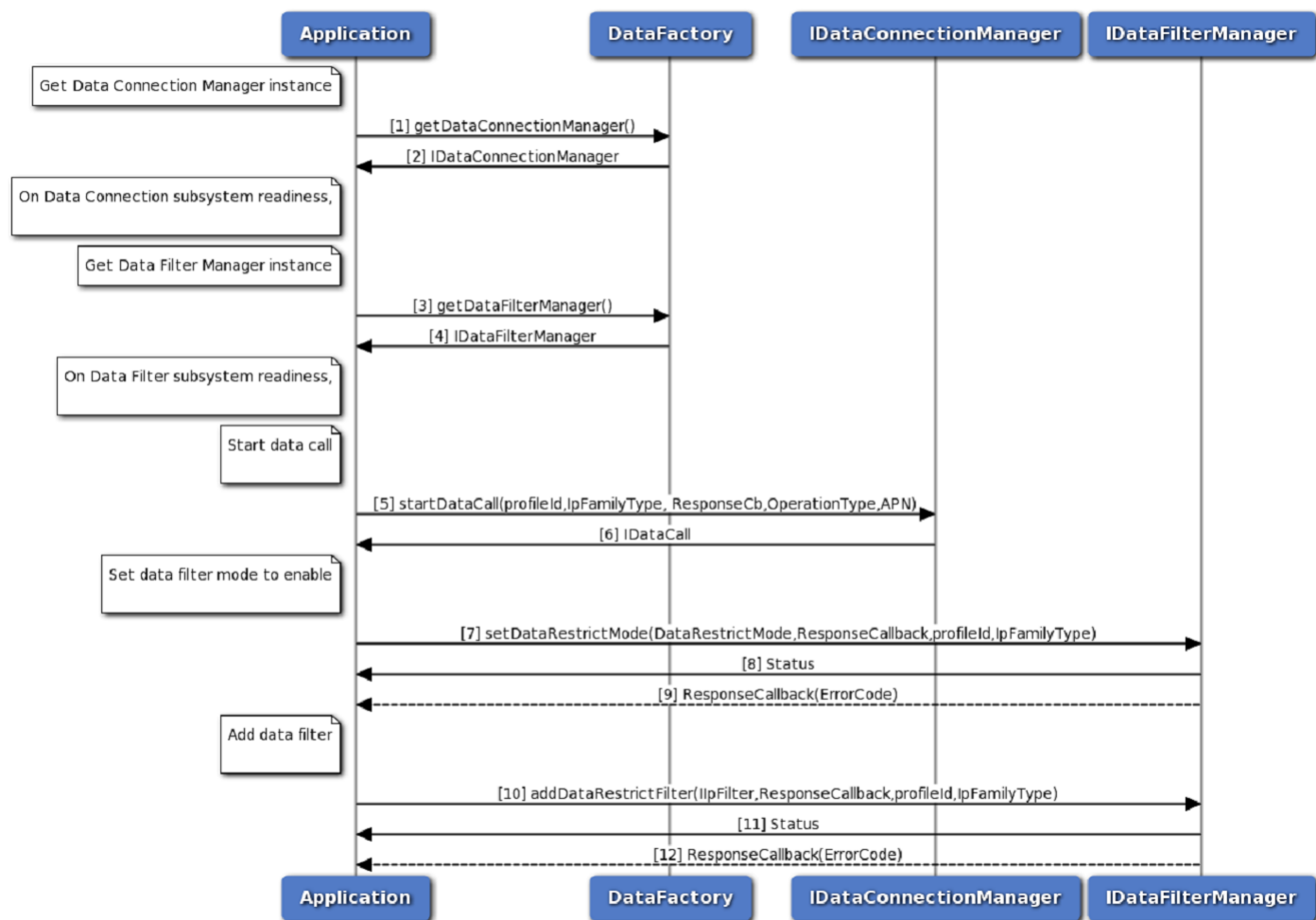


Figure 3-44 Add data restrict filter call flow

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

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

3.5.3.3 Call flow to Remove data restrict filter

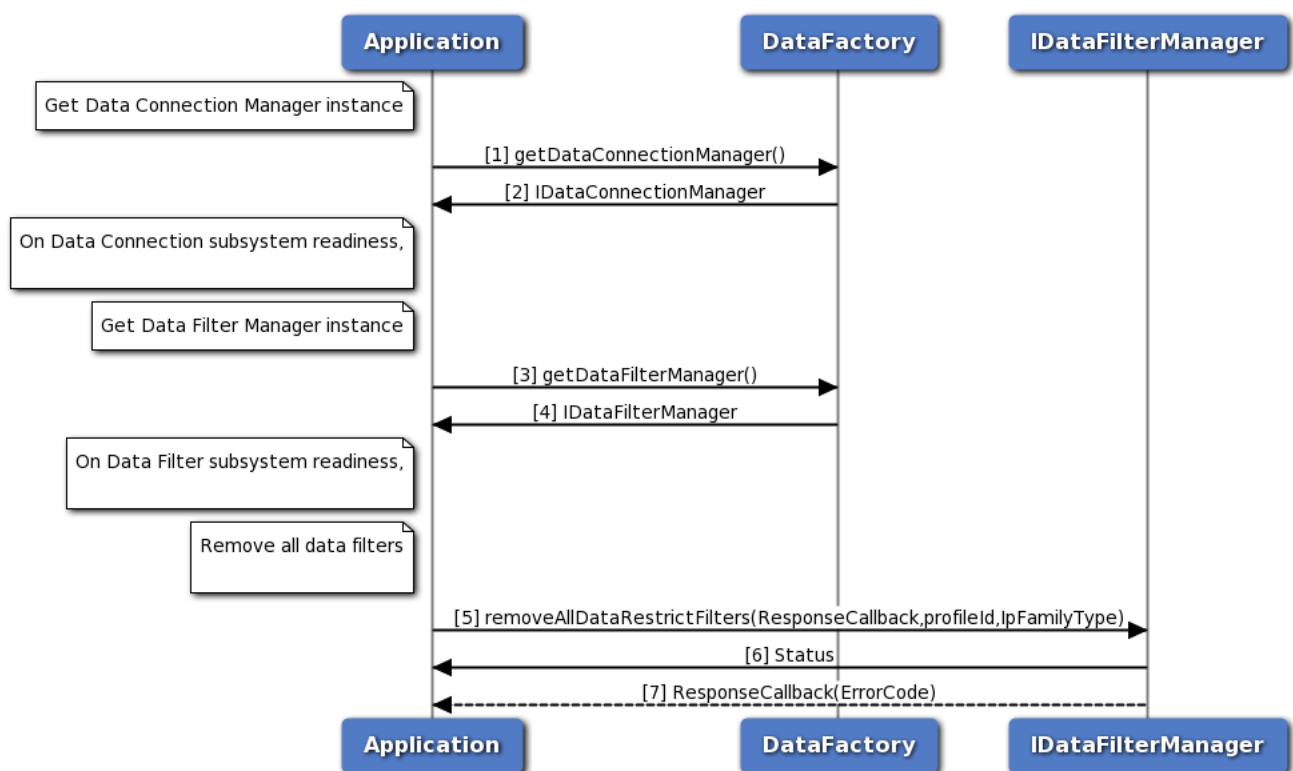


Figure 3-45 Remove data restrict filter call flow

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

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

3.5.4 Data Networking Call Flow

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

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

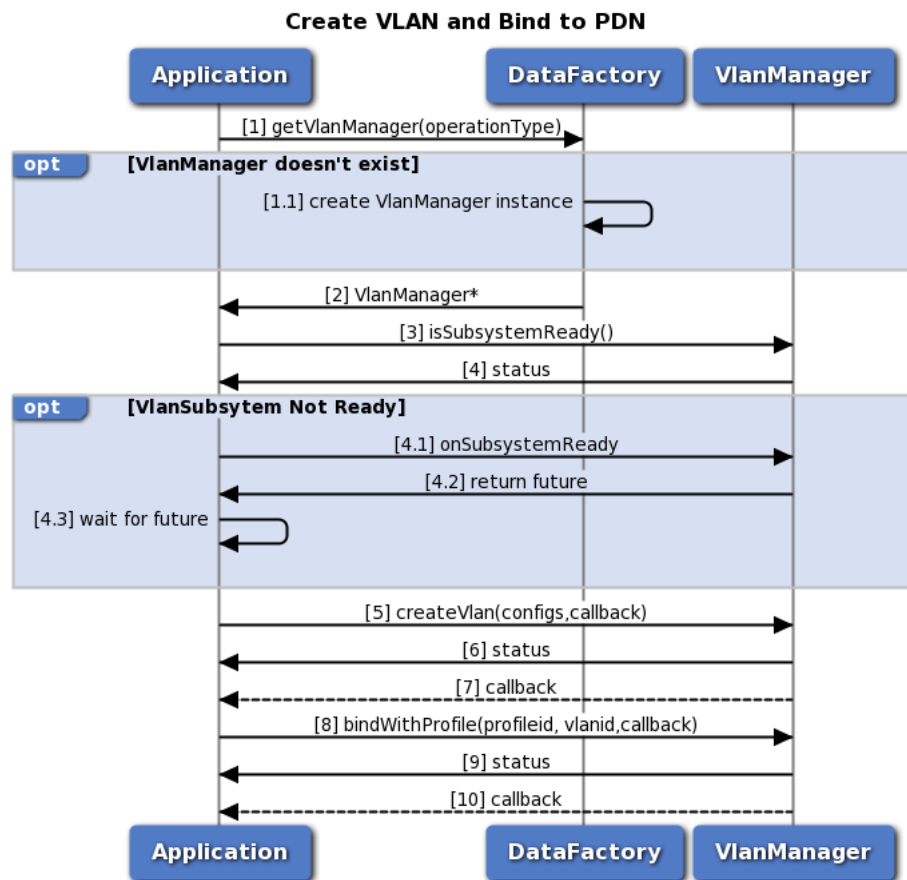


Figure 3-46 Create VLAN and bind it to PDN for data VLAN manager call flow

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

request was sent successfully.

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

3.5.4.2 LAN-LAN VLAN Configuration from EAP usecase call flow

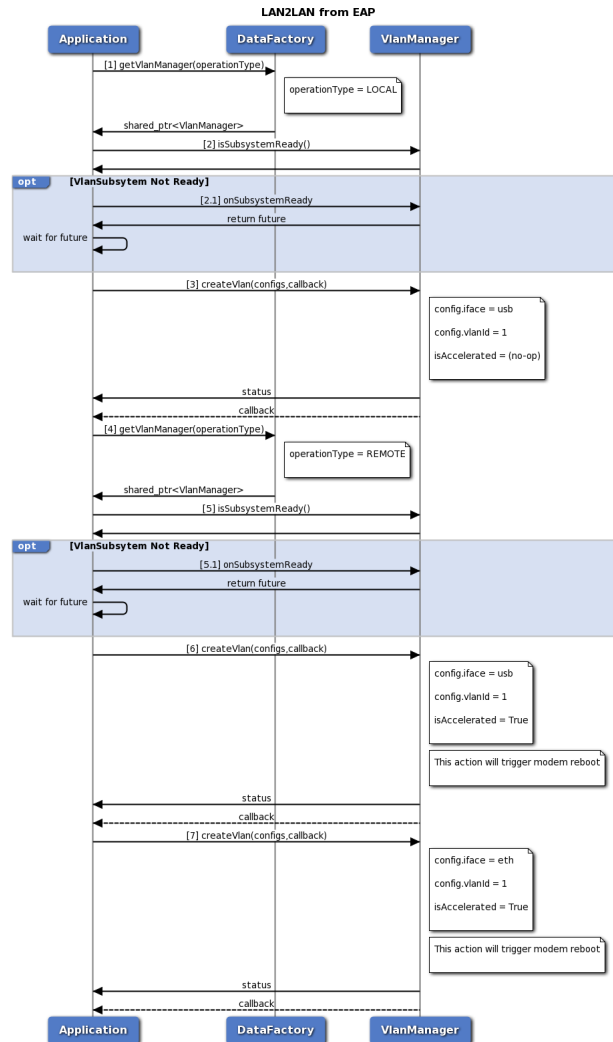


Figure 3-47 LAN-LAN VLAN Configuration Usecase from EAP call flow

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

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

3.5.4.3 LAN-LAN VLAN Configuration from A7 usecase call flow

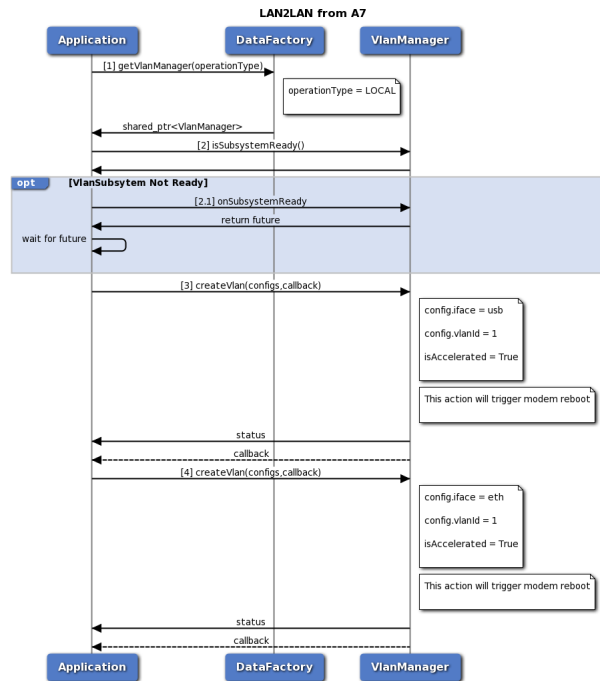


Figure 3-48 LAN-LAN VLAN Configuration Usecase from A7 call flow

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

3.5.4.4 LAN-WAN VLAN Configuration from EAP usecase call flow

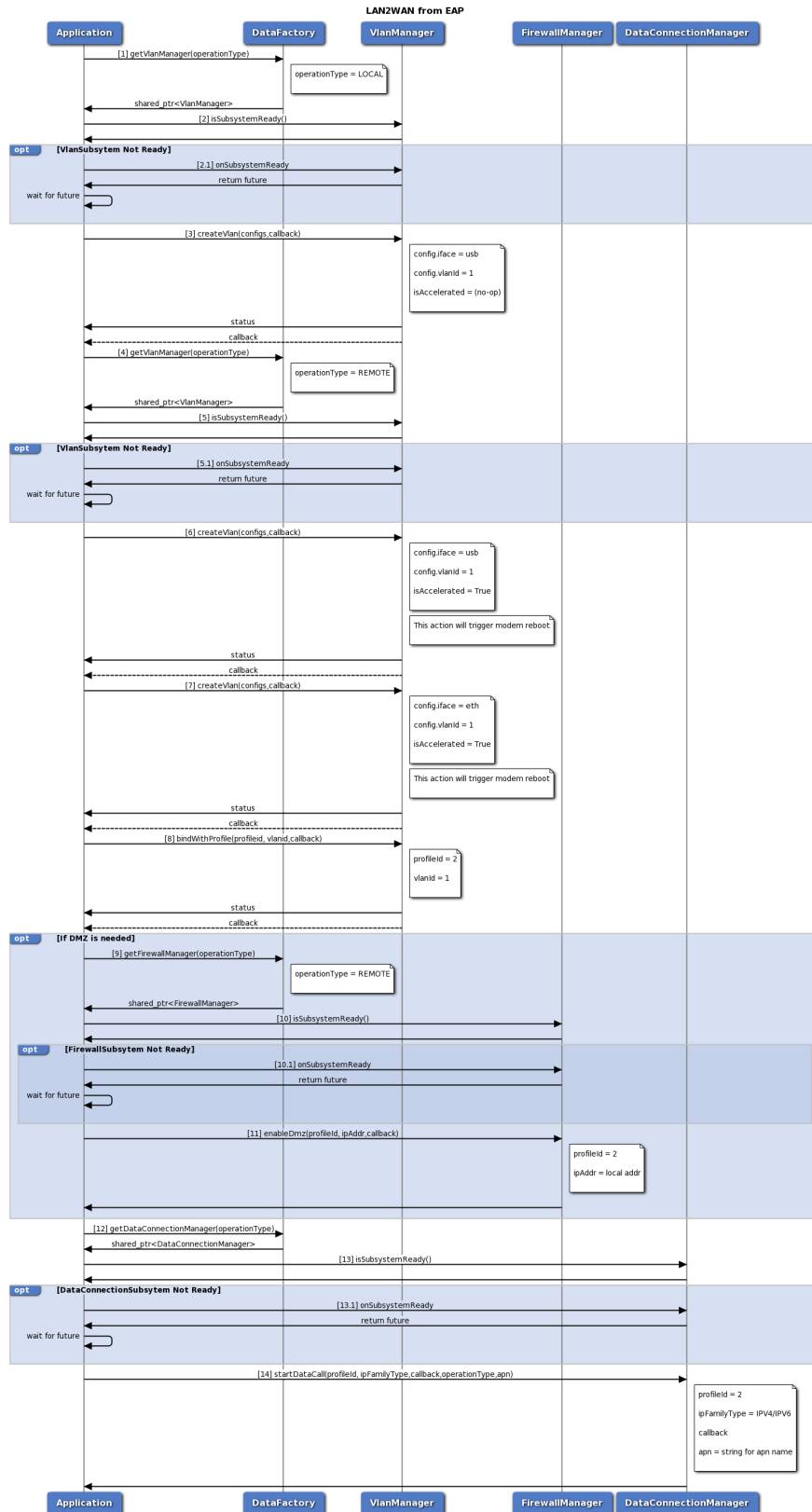


Figure 3-49 LAN-WAN VLAN Configuration Usecase from EAP call flow

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

3.5.4.5 LAN-WAN VLAN Configuration from A7 usecase call flow

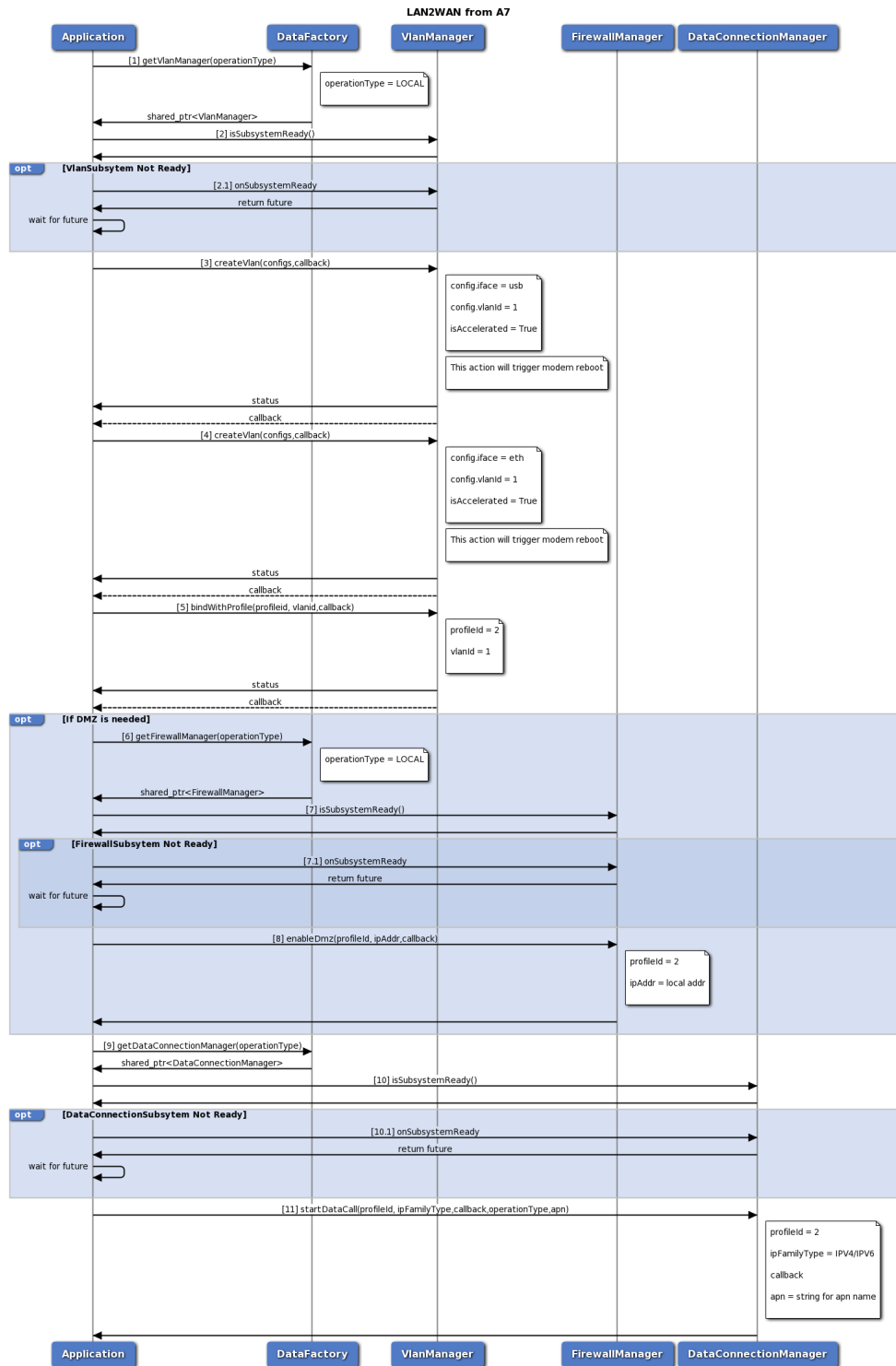


Figure 3-50 LAN-WAN VLAN Configuration Usecase from A7 call flow

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

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

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

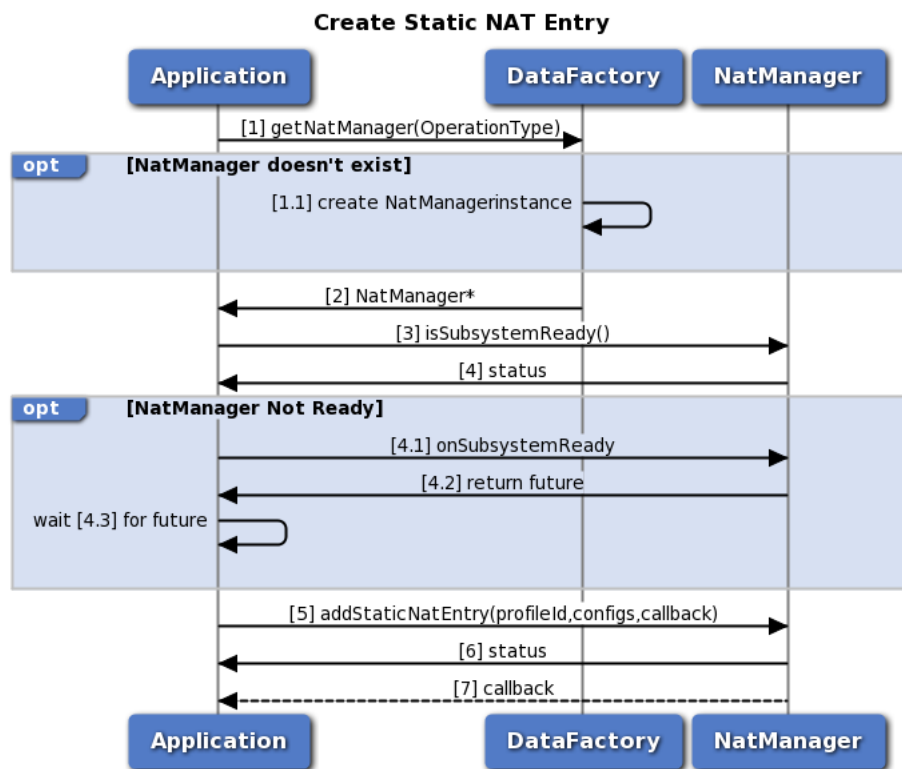


Figure 3-51 Create Static NAT entry for data NAT manager call flow

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

3.5.4.7 Firewall Enablement in data Firewall manager call flow

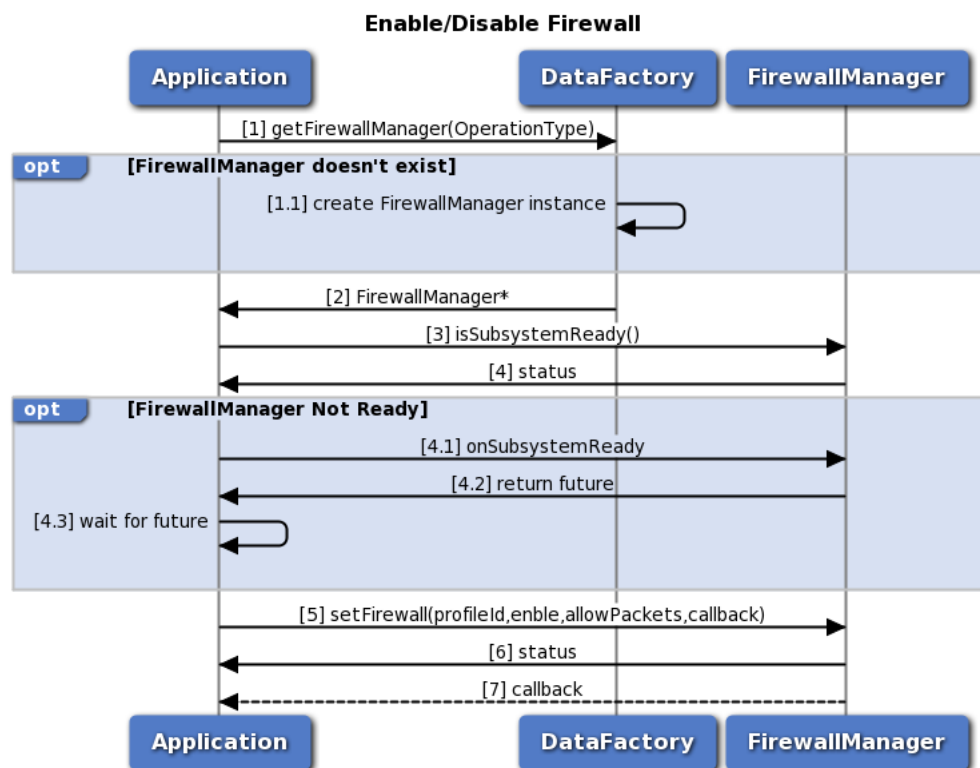


Figure 3-52 Firewall enablement in data Firewall manager call flow

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

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

3.5.4.8 Add Firewall Entry in data Firewall manager call flow

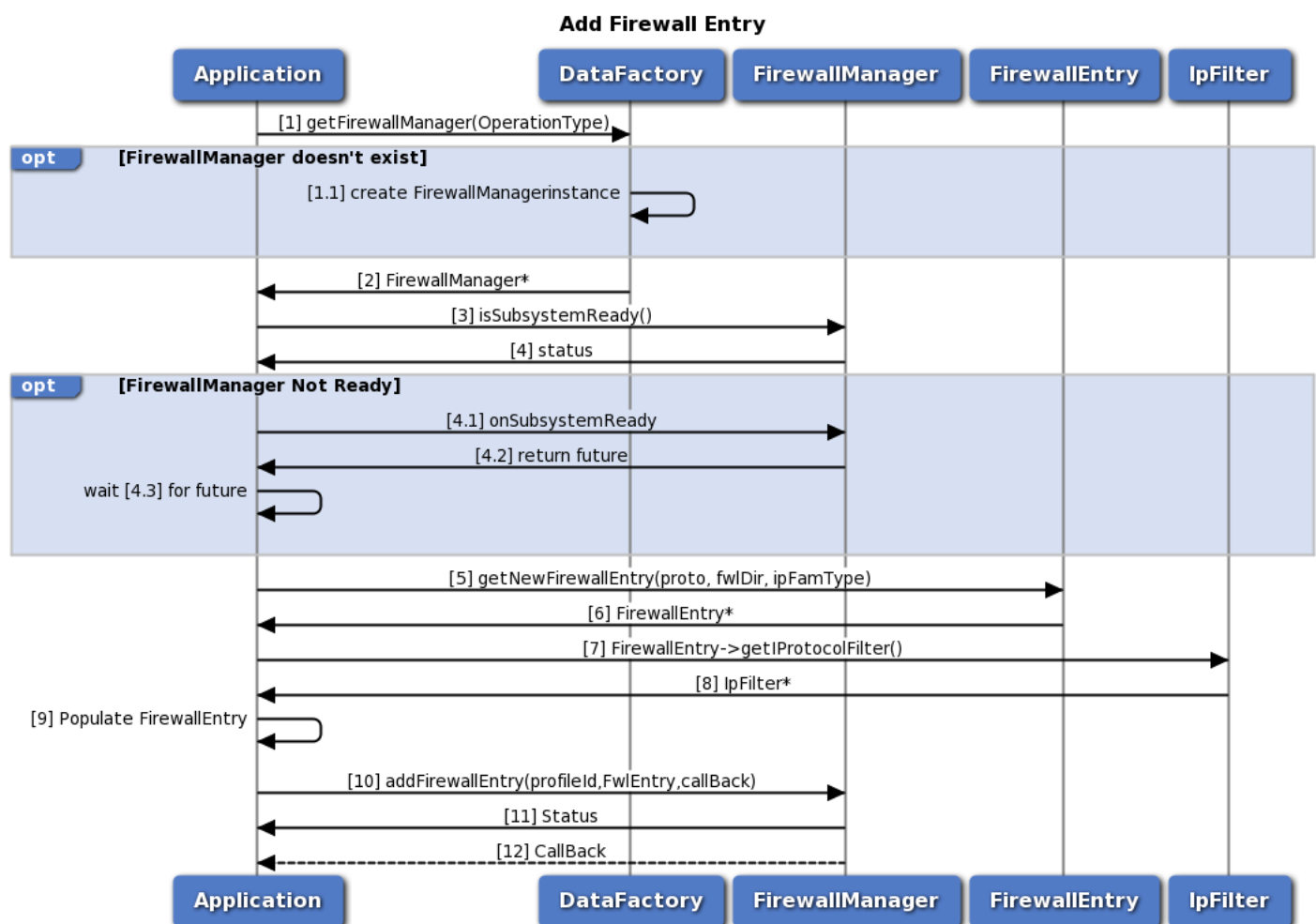


Figure 3-53 Add Firewall entry in data Firewall manager call flow

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

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

3.5.4.9 Set Firewall DMZ in data Firewall manager call flow

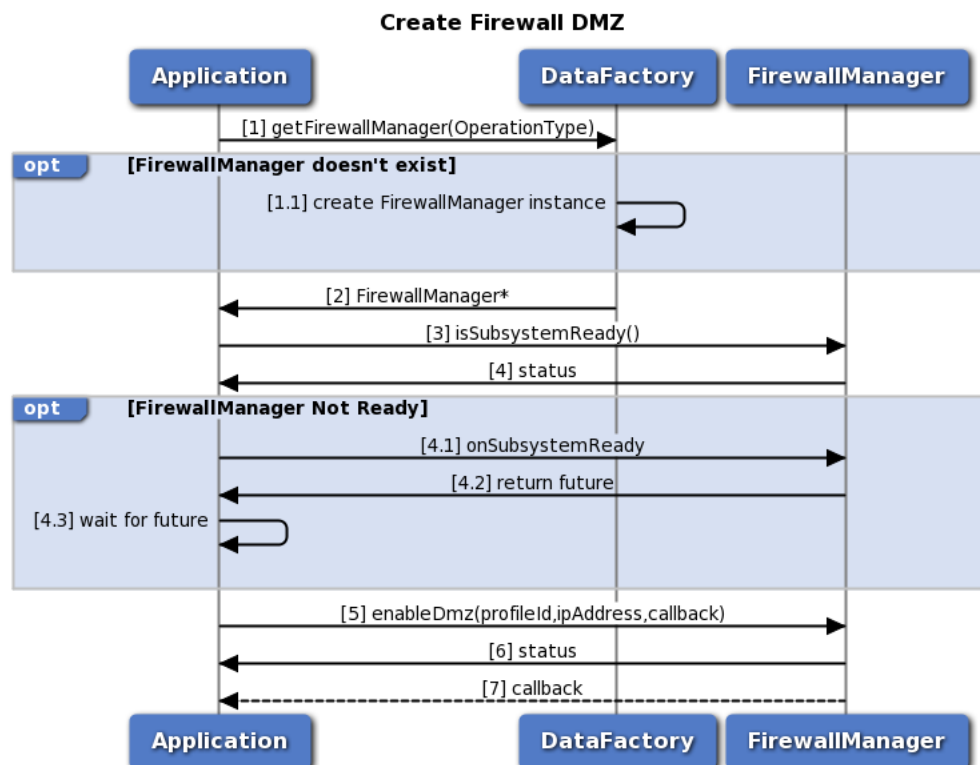


Figure 3-54 Set Firewall DMZ in data Firewall manager call flow

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

3.5.4.10 Socks Enablement in data Socks manager call flow

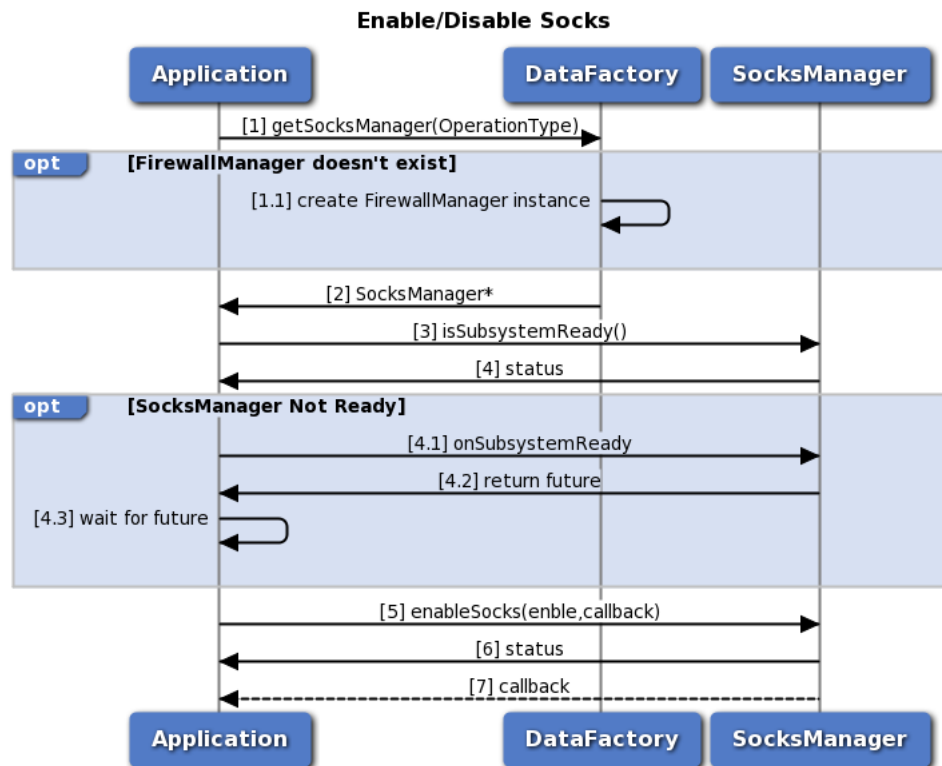


Figure 3-55 Socks enablement in data Socks manager call flow

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

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

3.5.4.11 Call flow to add and enable software bridge

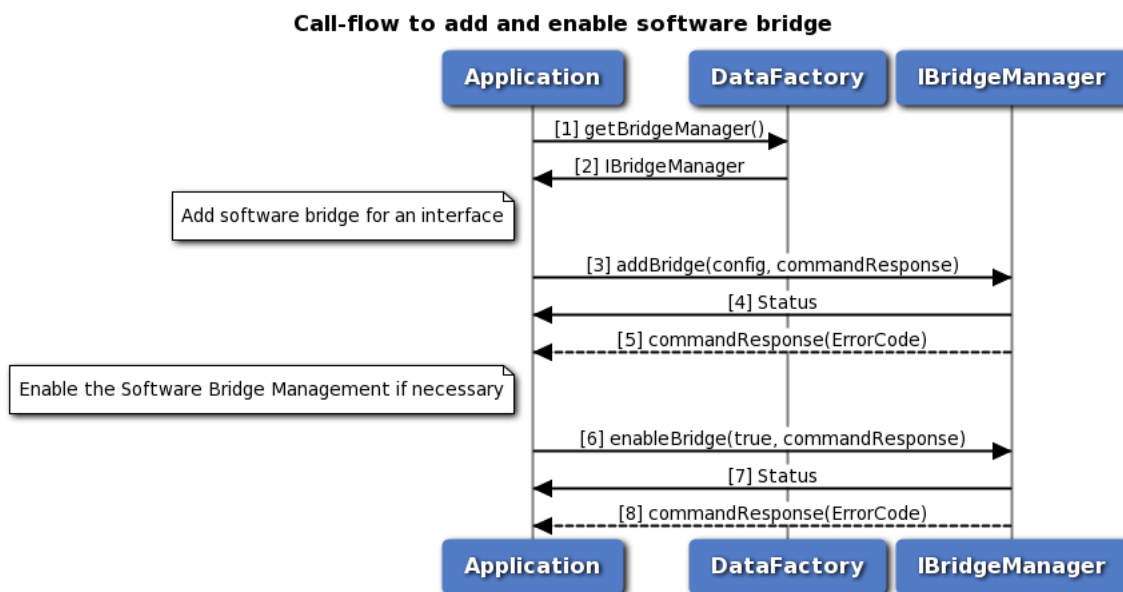


Figure 3-56 Call flow to add and enable a software bridge

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

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

3.5.4.12 Call flow to remove and disable software bridge

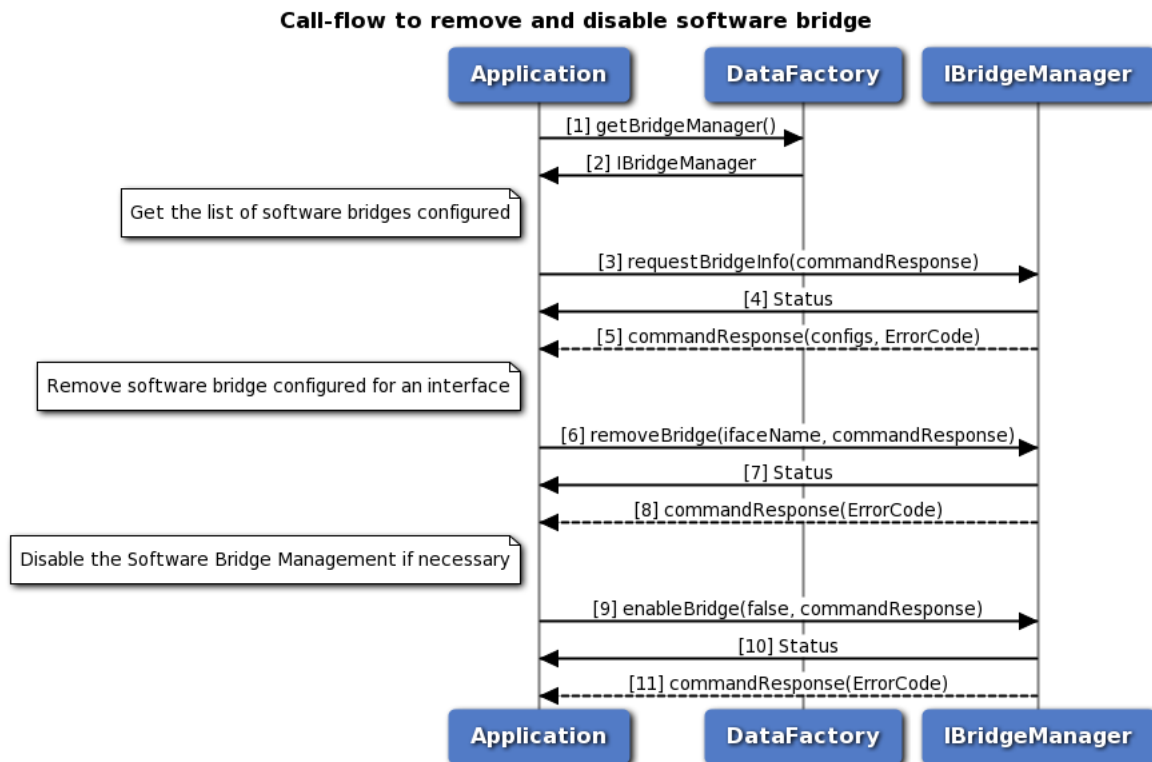


Figure 3-57 Call flow to remove and disable a software bridge

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

all the software bridges configured in the system.

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

3.6 C-V2X

3.6.1 Start/Stop C-V2X Mode

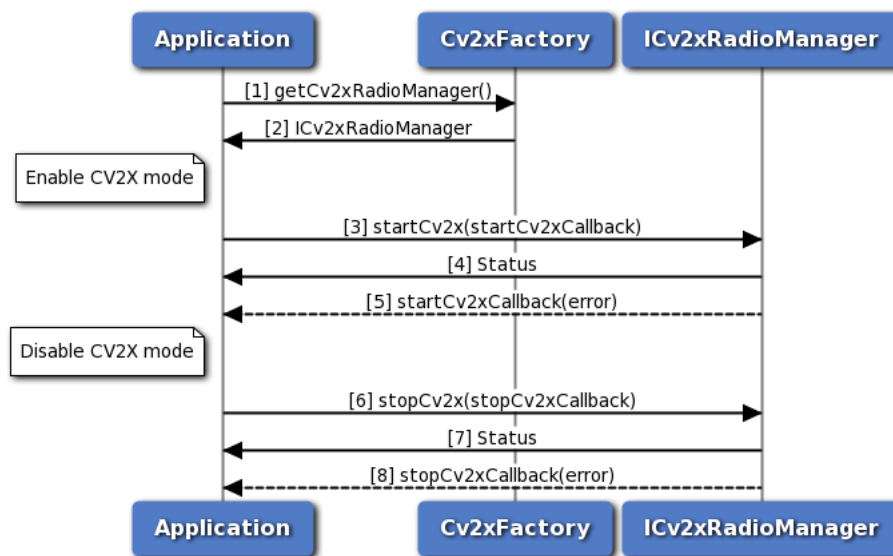


Figure 3-58 Start/Stop C-V2X Mode call flow

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

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

3.6.2 C-V2X Radio Manager API

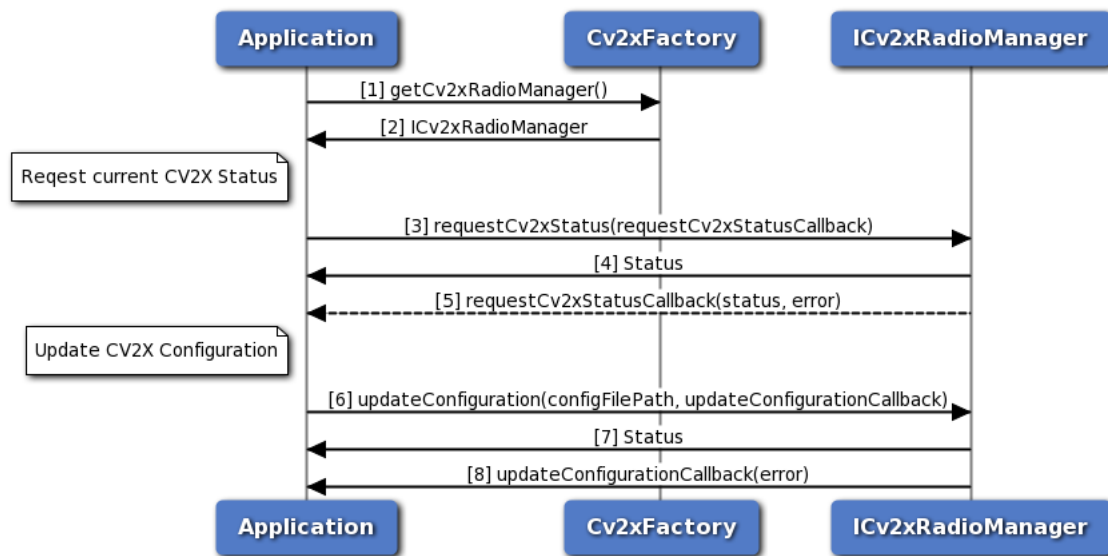


Figure 3-59 C-V2X Radio Manager call flow

API for C-V2X Radio Manager

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

3.6.3 C-V2X Radio Initialization

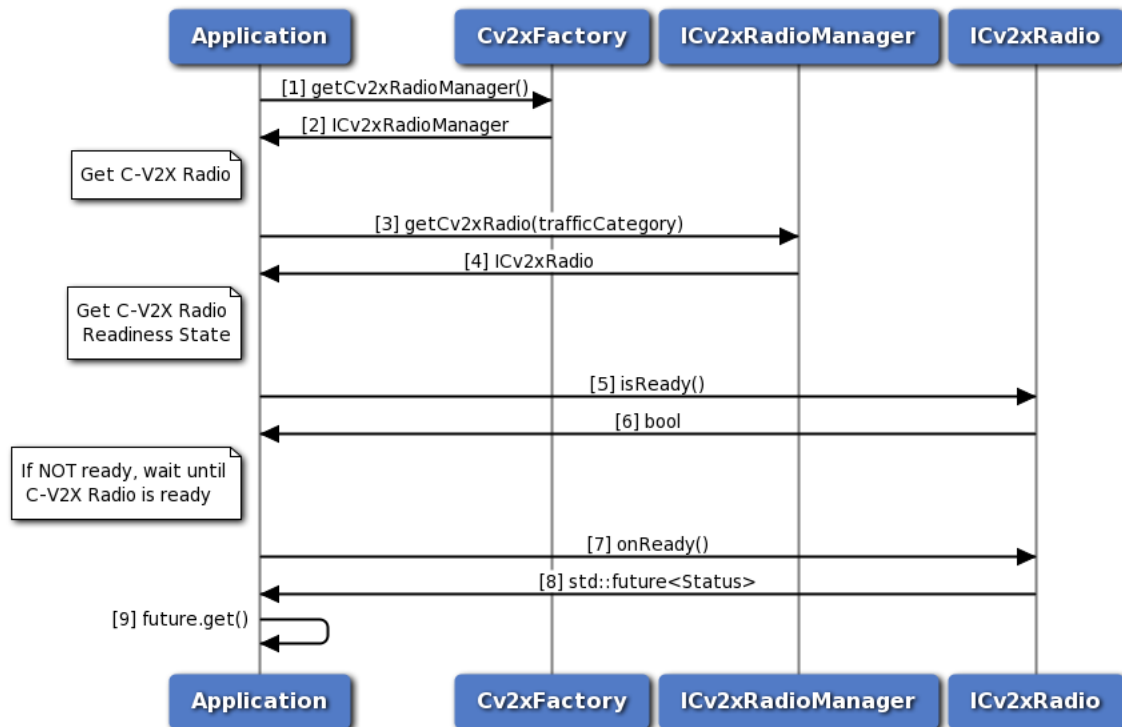


Figure 3-60 C-V2X Radio Initialization call flow

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

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

3.6.4 C-V2X Radio RX subscription

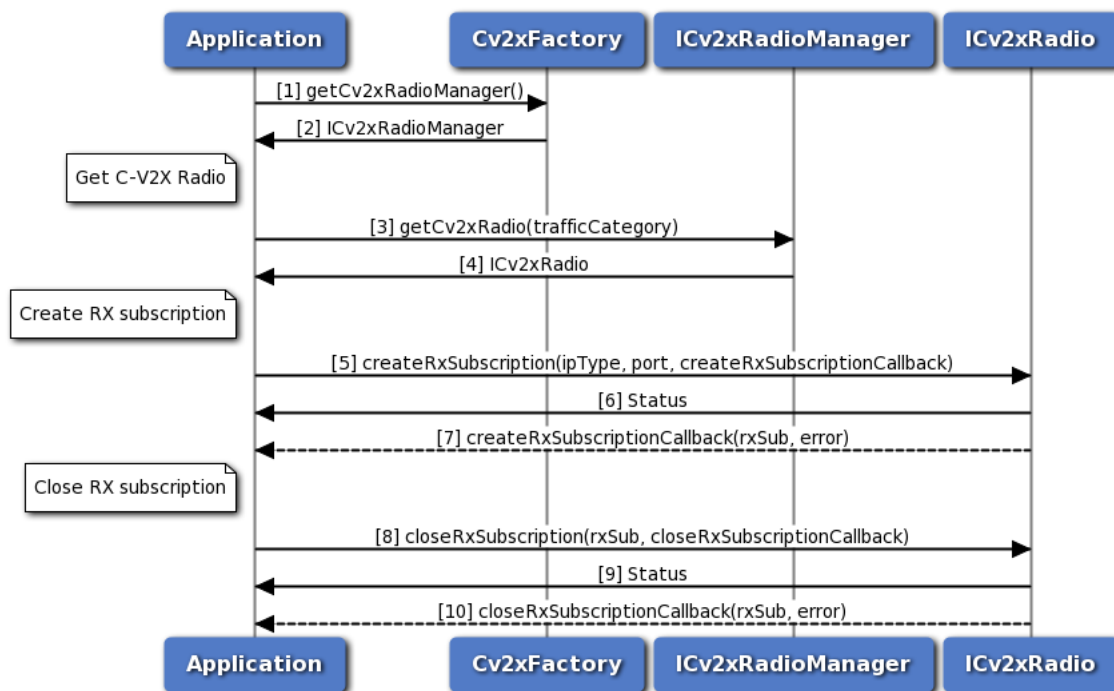


Figure 3-61 C-V2X Radio RX subscription call flow

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

3.6.5 C-V2X Radio TX event-driven flow

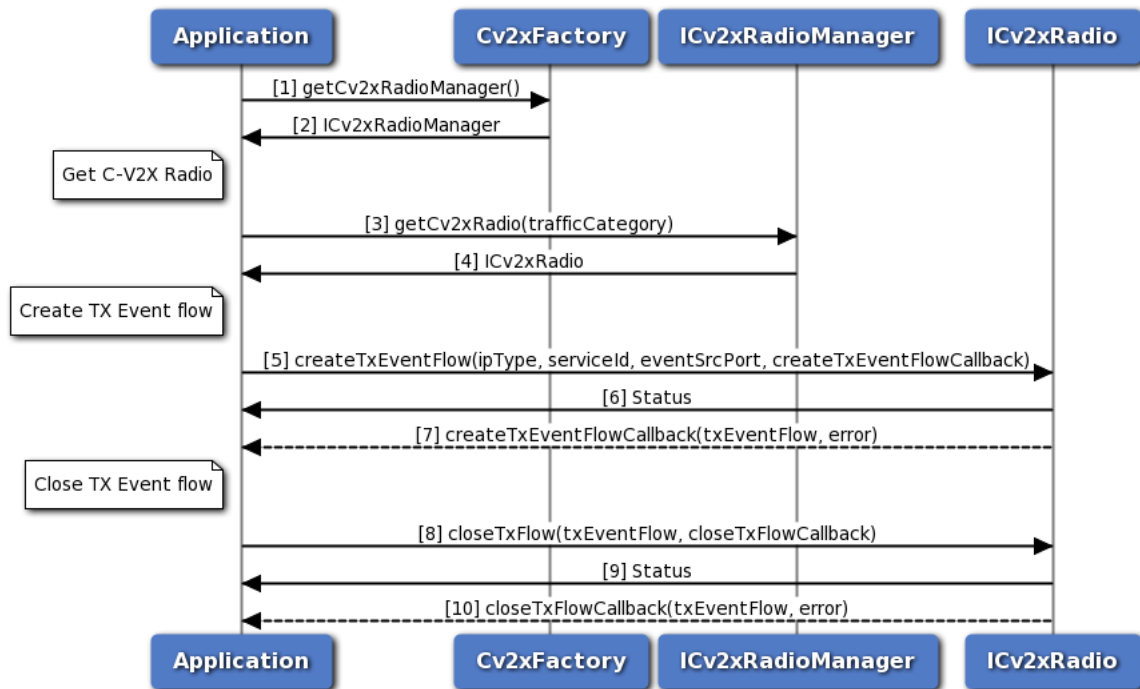


Figure 3-62 C-V2X Radio TX event-driven flow call flow

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

3.6.6 C-V2X Radio TX SPS flow

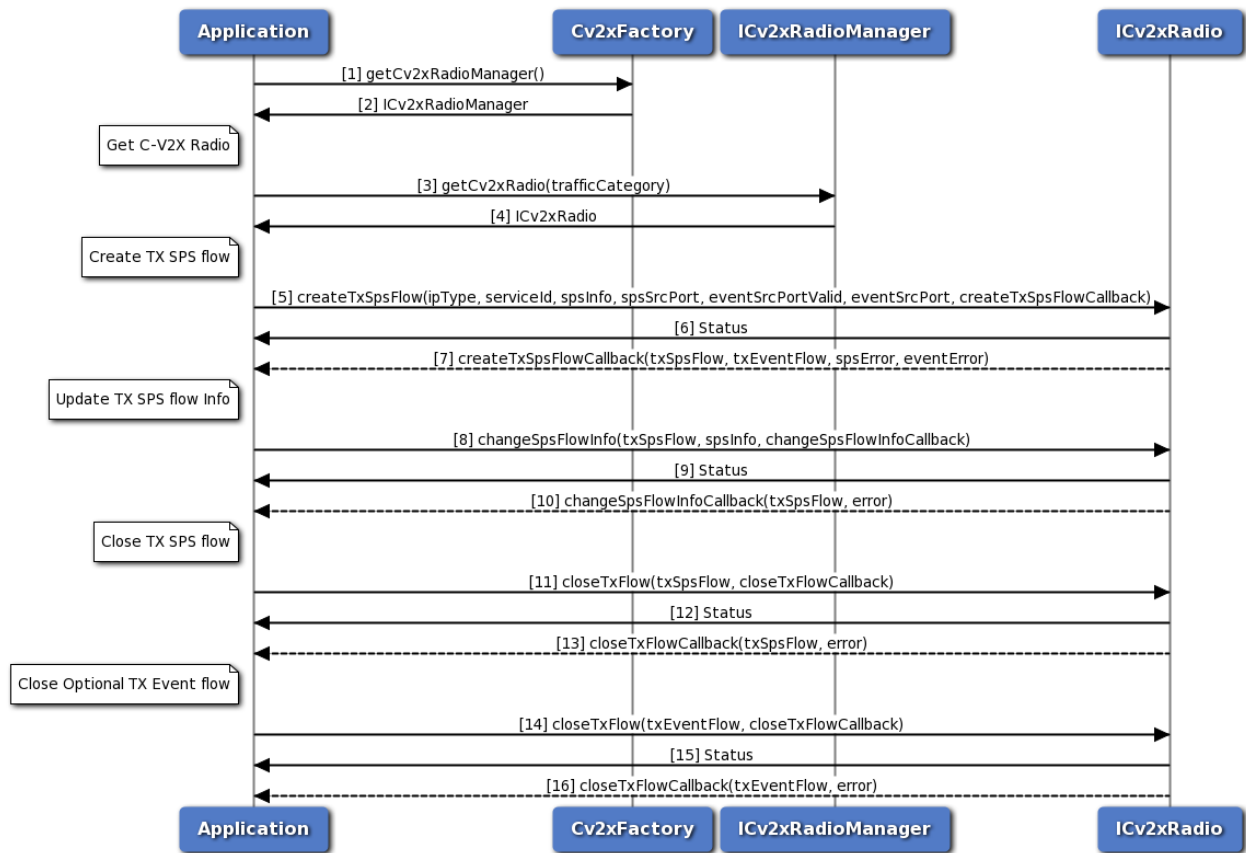


Figure 3-63 C-V2X Radio SPS flow call flow

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

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

3.7 Audio

3.7.1 Audio Manager API call flow

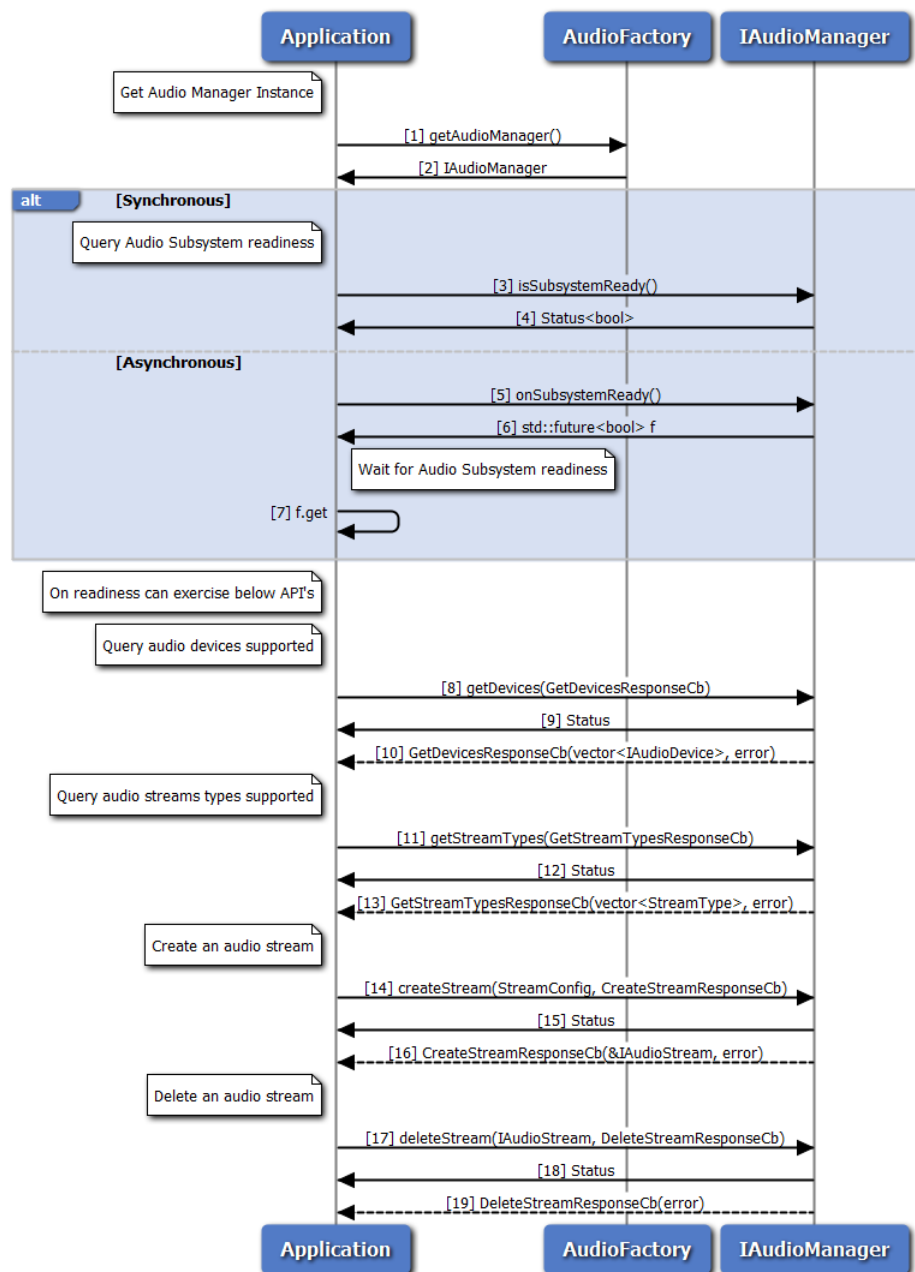


Figure 3-64 Audio Manager API call flow

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

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

3.7.2 Audio Voice Call Start/Stop call flow

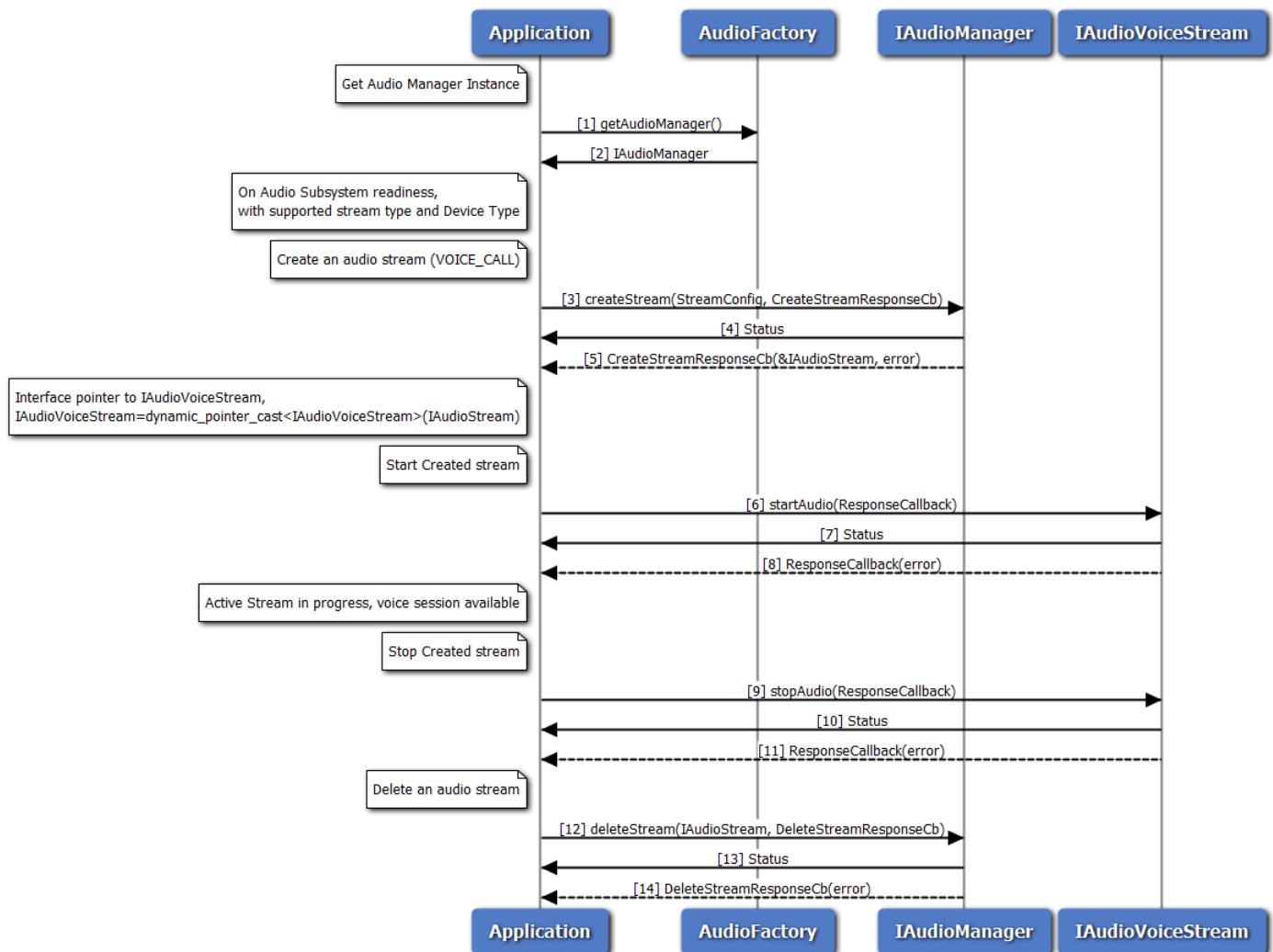


Figure 3-65 Audio Voice Call Start/Stop call flow

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

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

3.7.3 Audio Voice Call Device Switch call flow

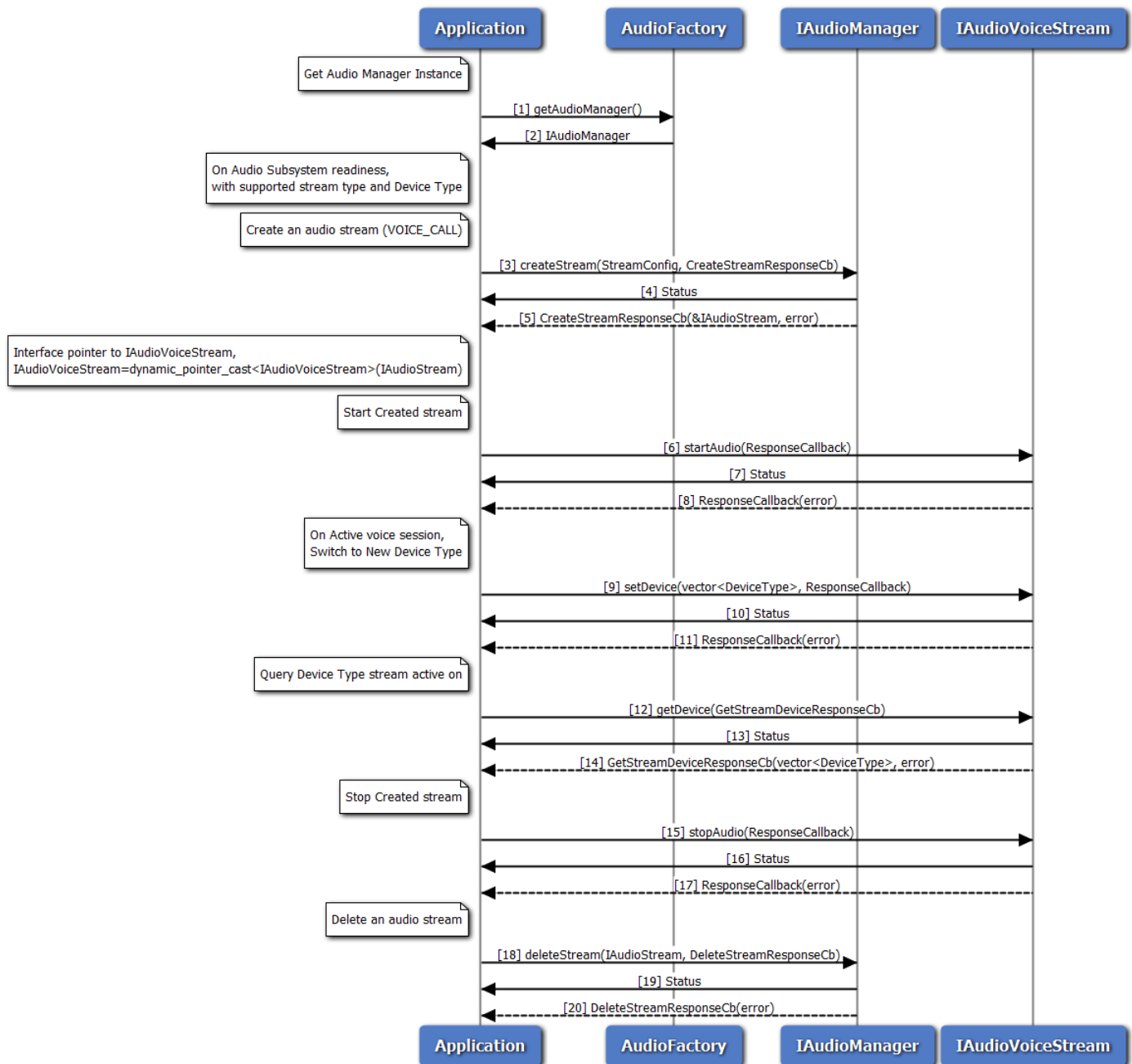


Figure 3-66 Audio Voice Call Device Switch call flow

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

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

3.7.4 Audio Voice Call Volume/Mute control call flow

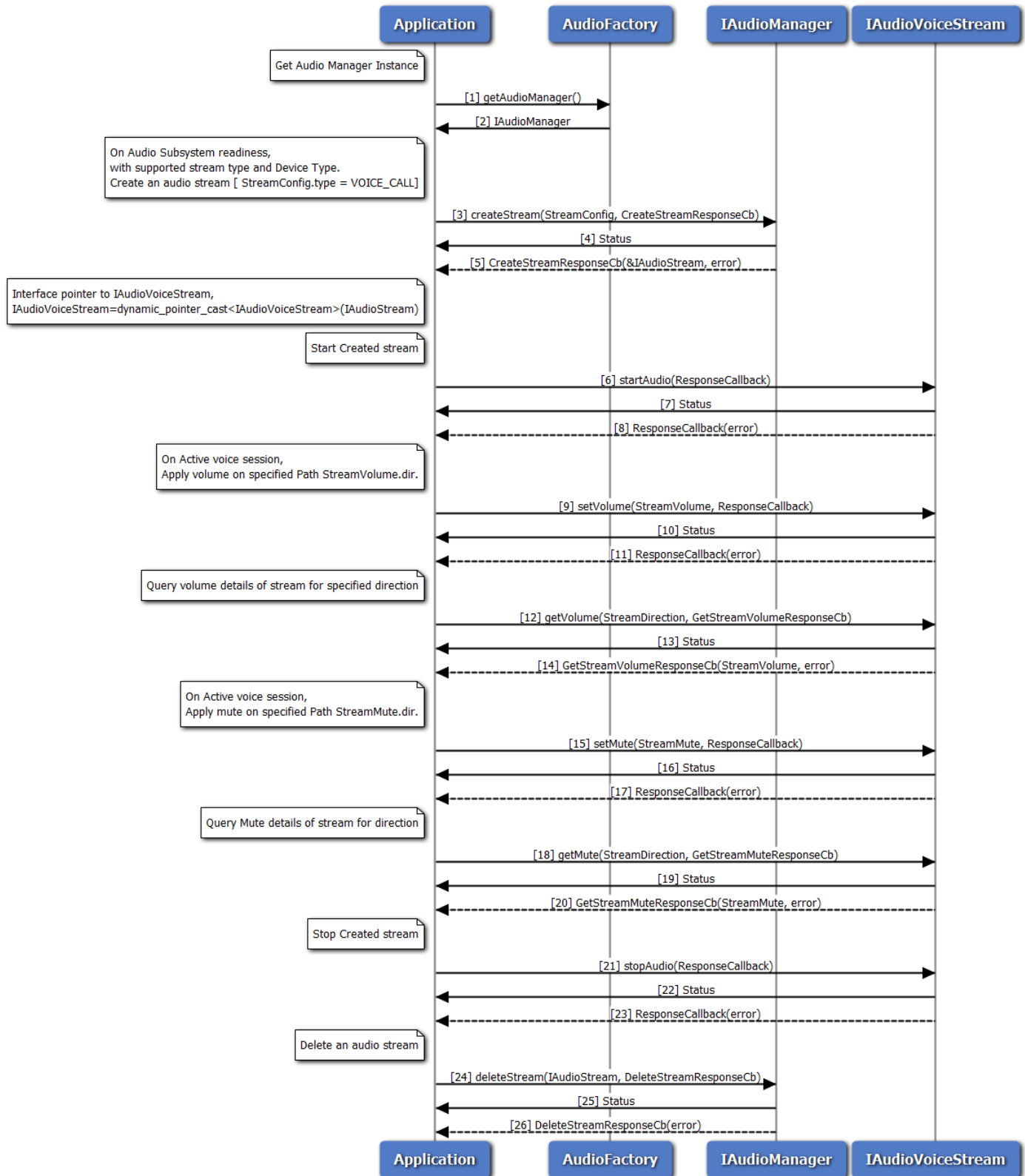


Figure 3-67 Audio Voice Call Volume/Mute control call flow

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

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

3.7.5 Call flow to play DTMF tone

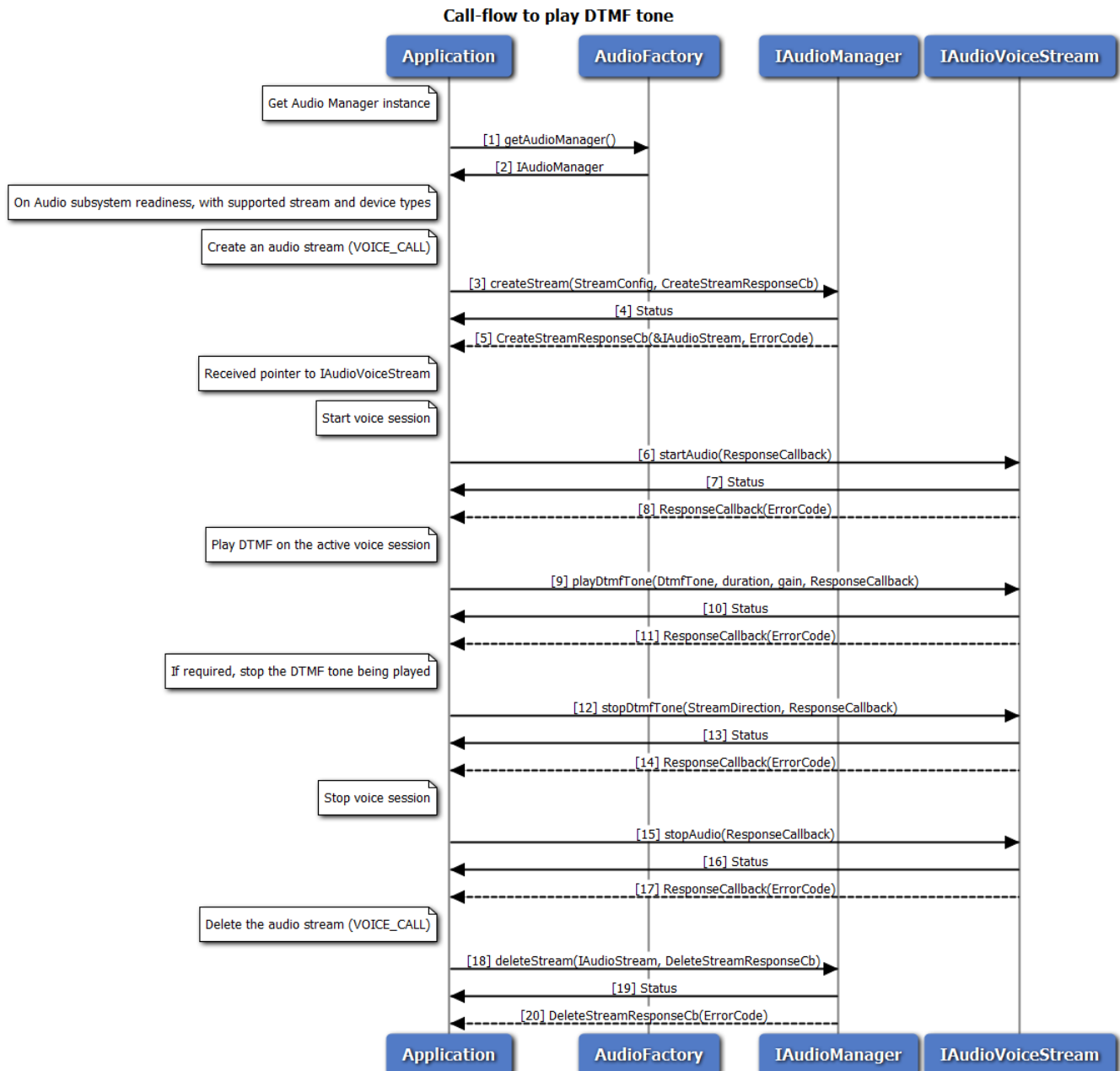


Figure 3-68 Call flow to play DTMF tone

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

3.7.6 Call flow to detect DTMF tones

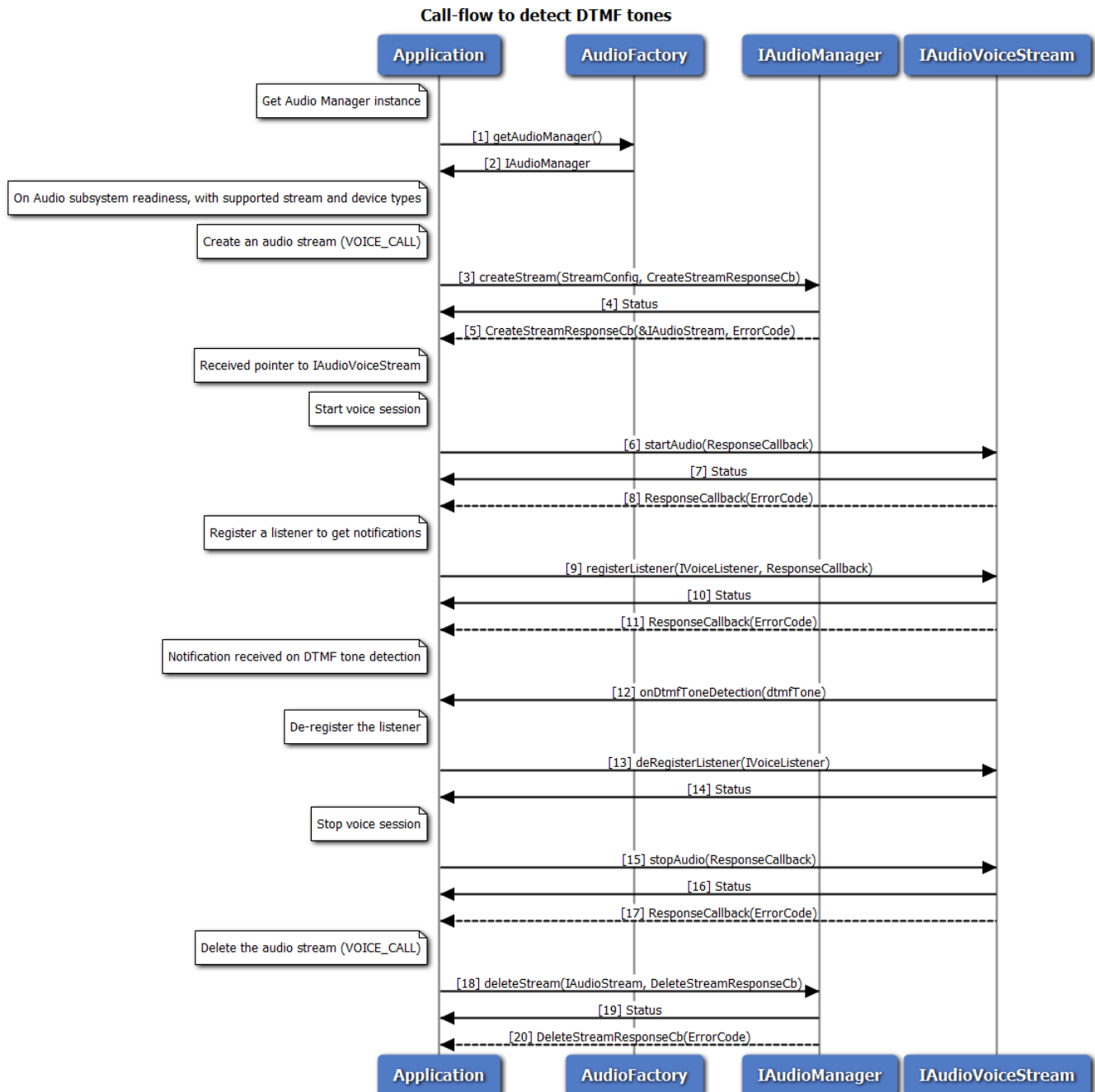


Figure 3-69 Call flow to detect DTMF tone

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

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

3.7.7 Audio Playback call flow

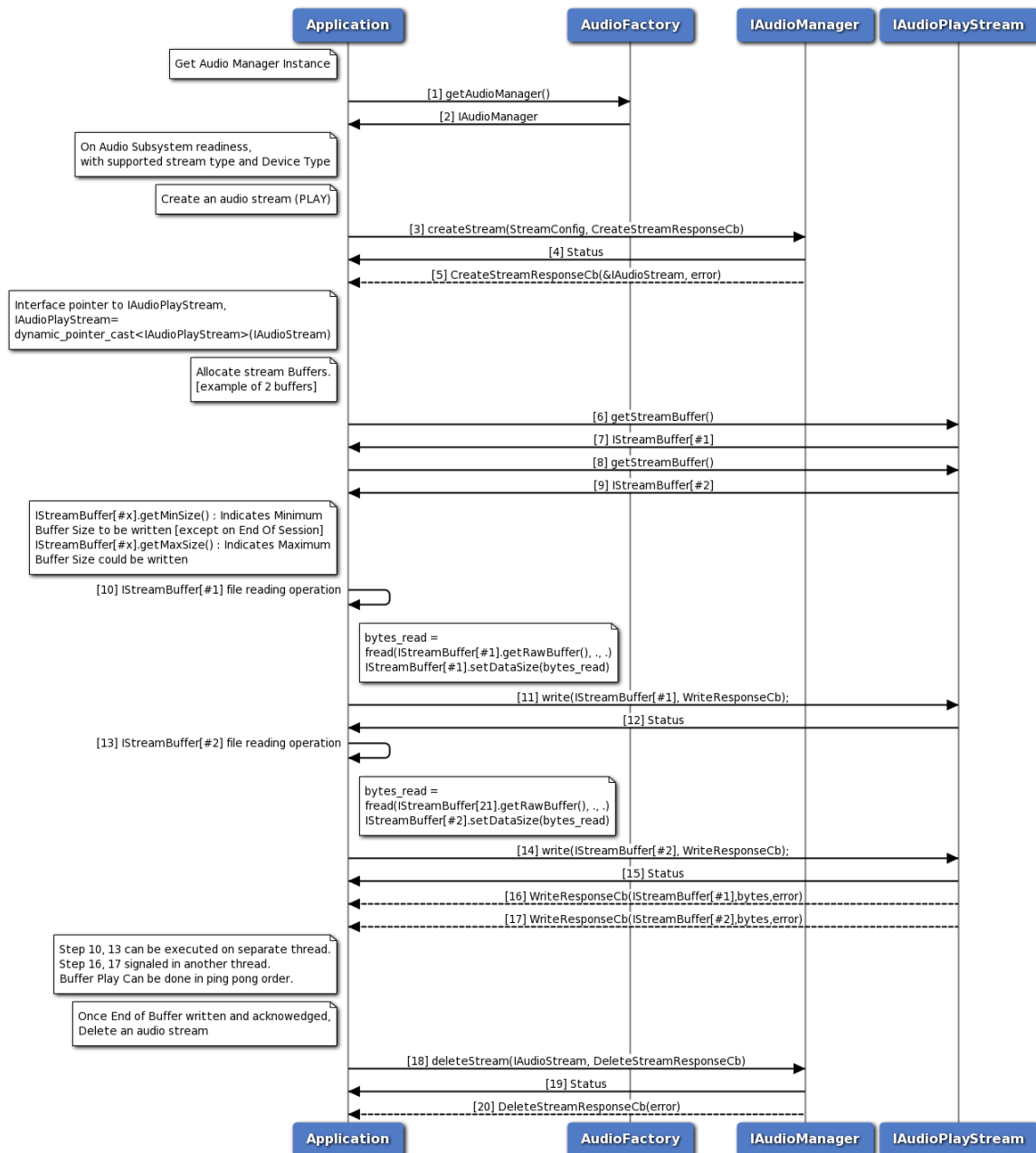


Figure 3-70 Audio Playback call flow

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

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

3.7.8 Audio Capture call flow

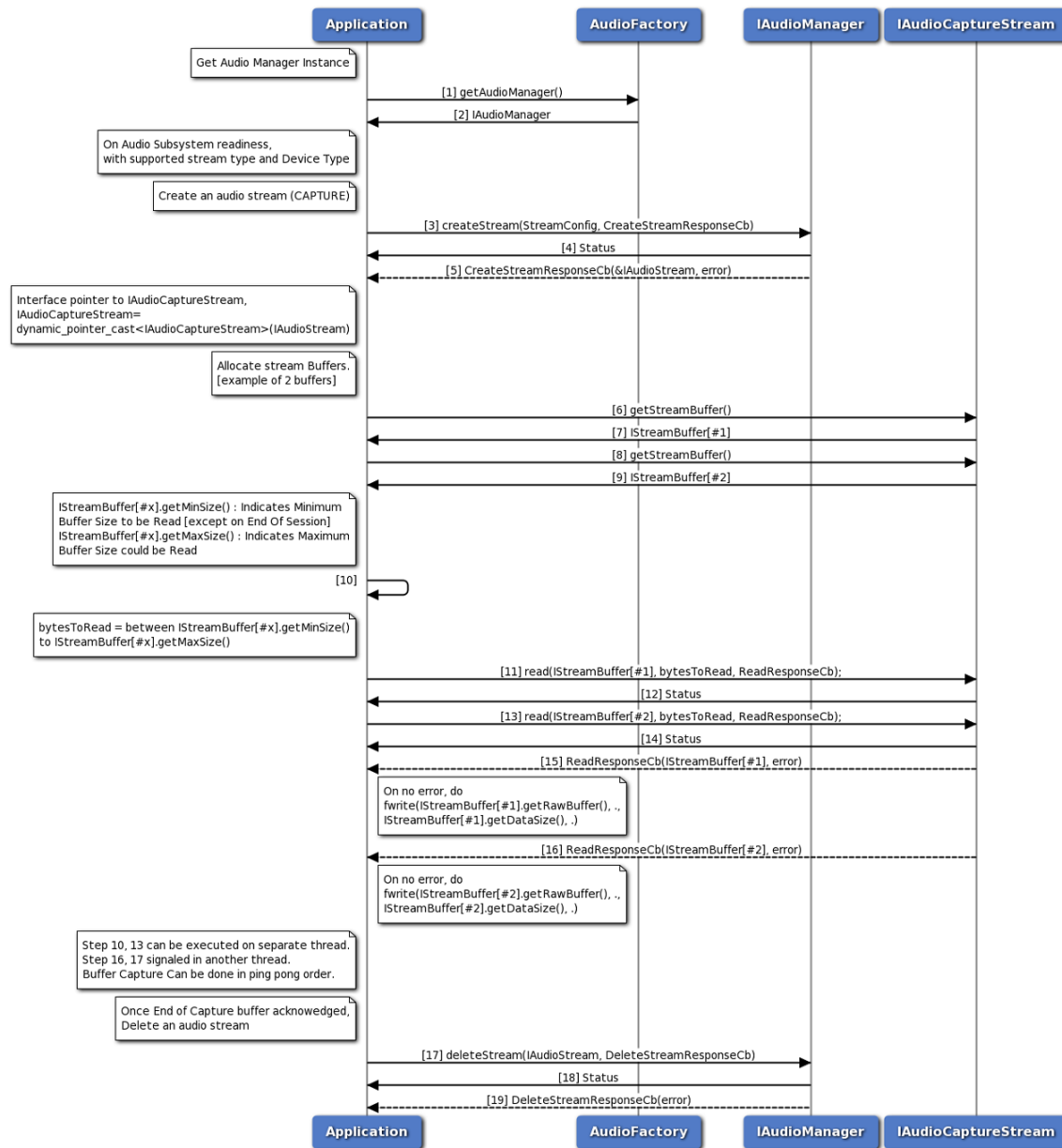


Figure 3-71 Audio Capture call flow

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

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

3.7.9 Audio Tone Generator call flow

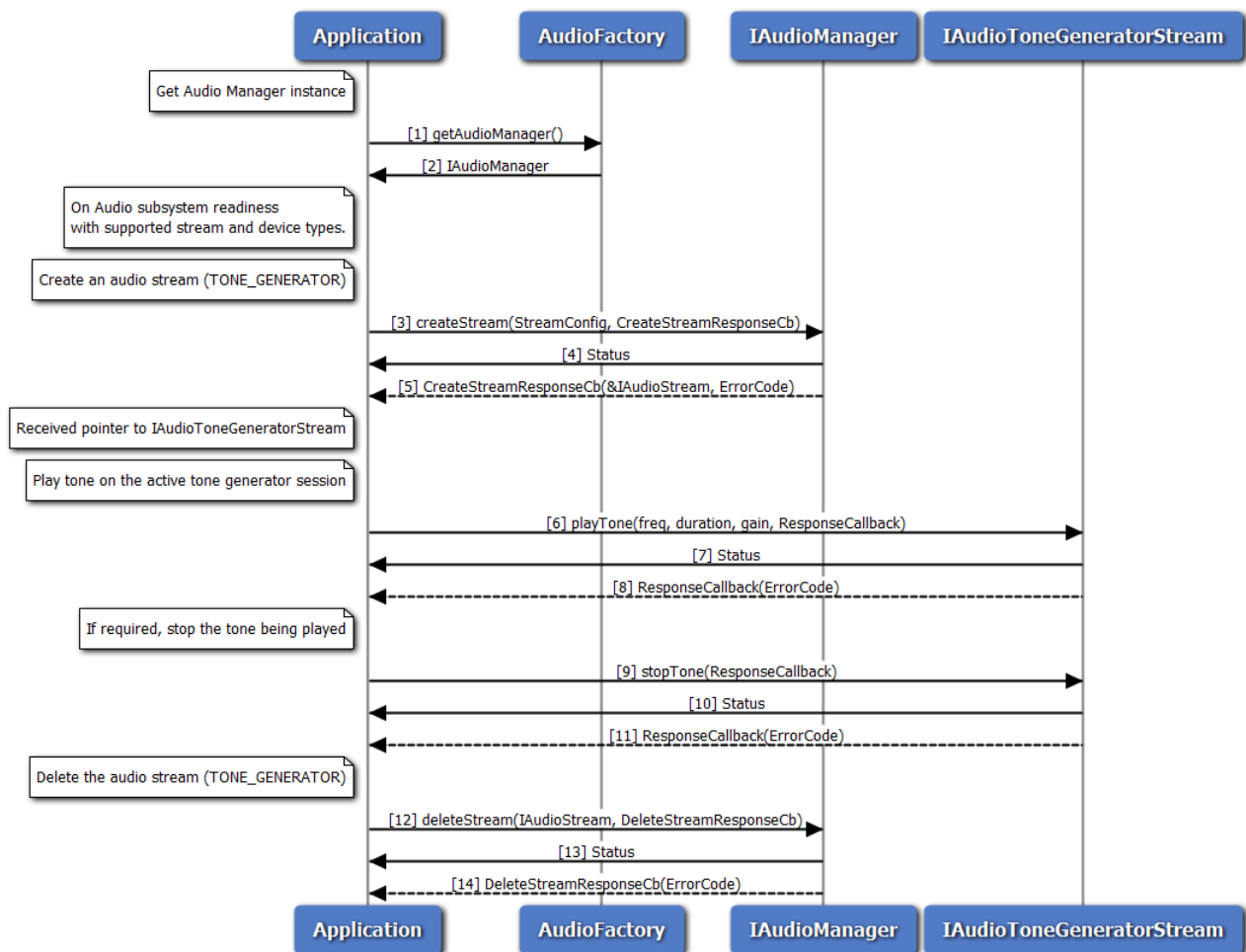


Figure 3-72 Call flow to play/stop tone on a sink device

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

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

3.7.10 Audio Loopback call flow

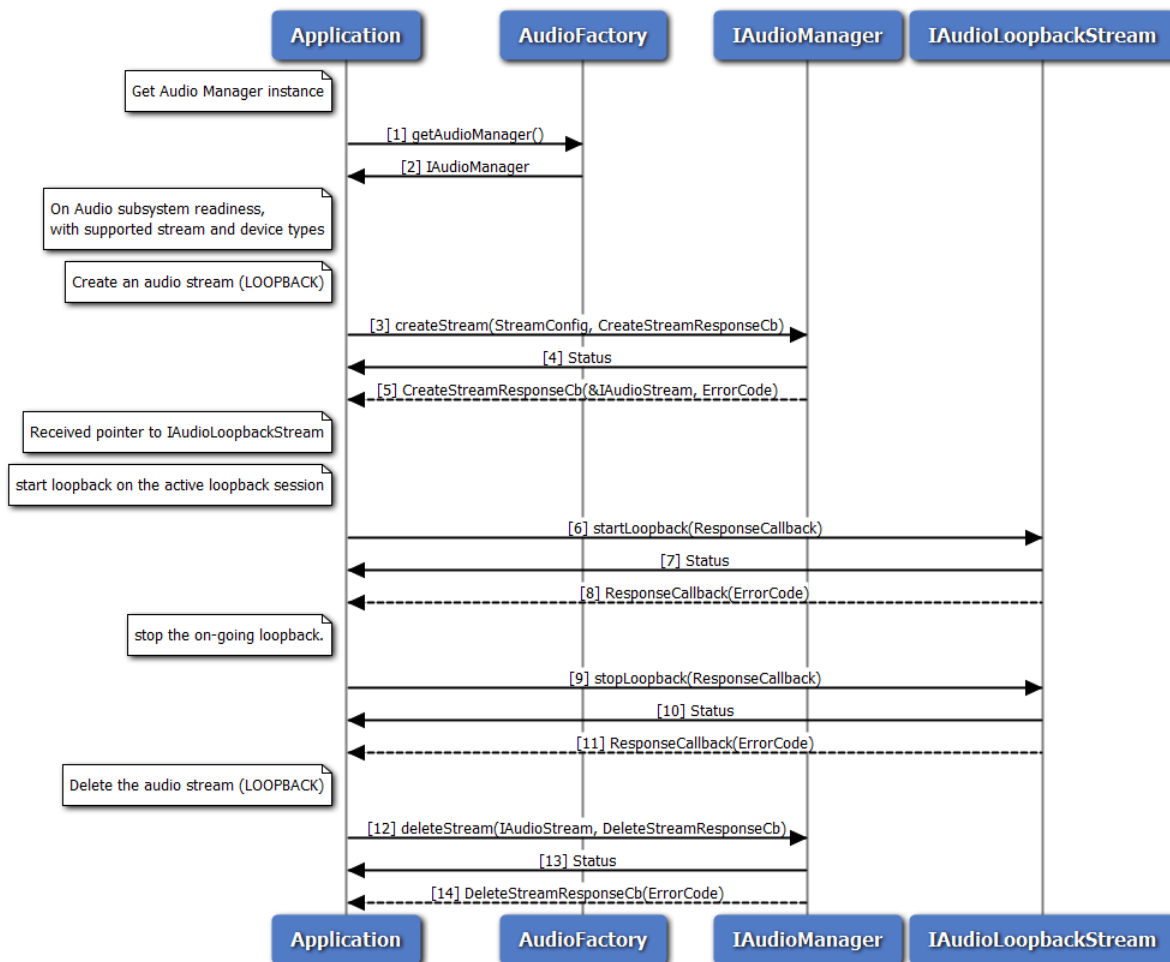


Figure 3-73 Call flow to start/stop loopback between source and sink devices

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

3.7.11 Compressed audio format playback call flow

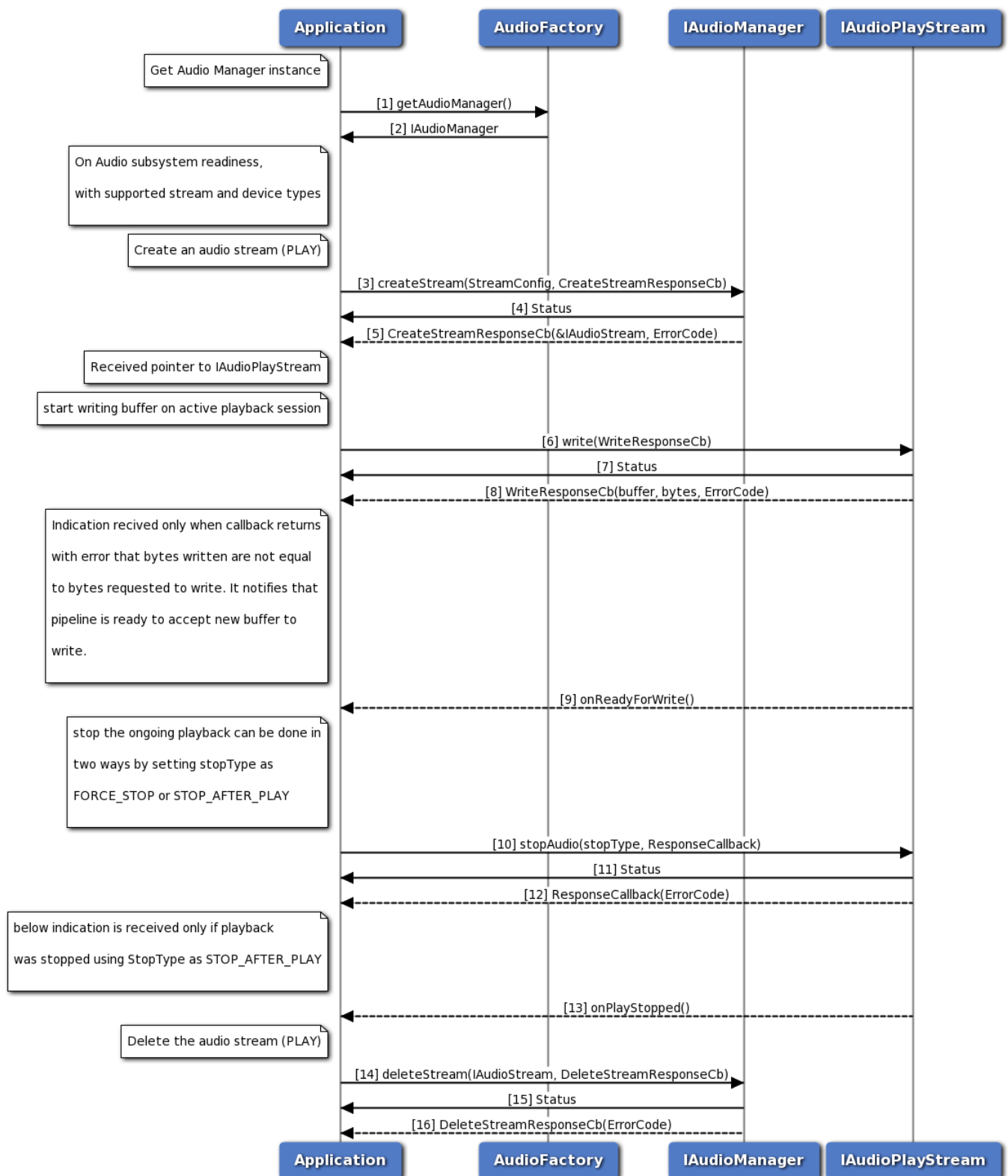


Figure 3-74 Call flow to play Compressed audio format

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

3.7.12 Audio Transcoding Operation Callflow

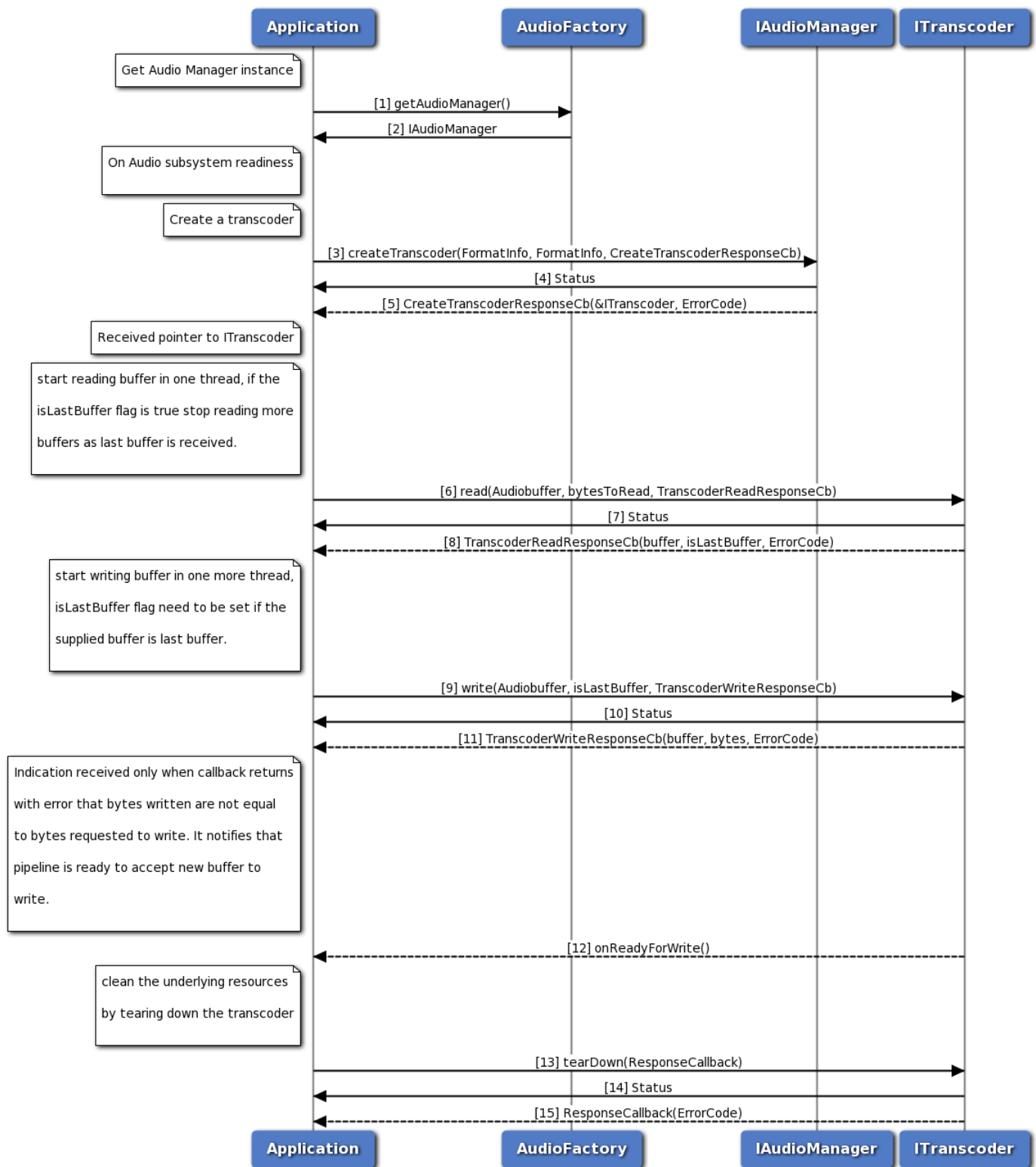


Figure 3-75 Audio Transcoding Operation Callflow

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

3.7.13 Compressed audio format playback on Voice Paths Callflow

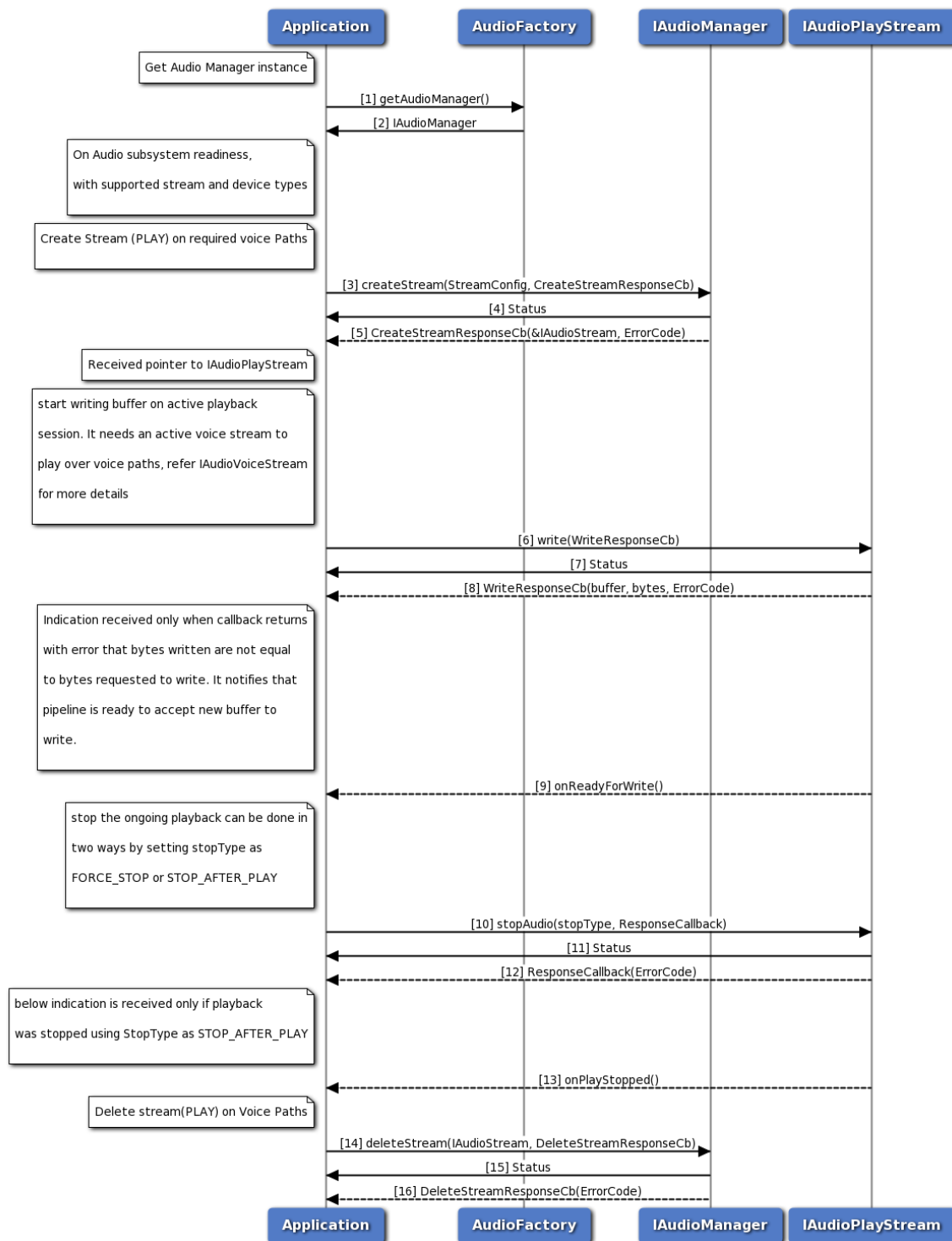


Figure 3-76 Compressed audio format playback on Voice Paths Callflow

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

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

3.8 Thermal manager call flow

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

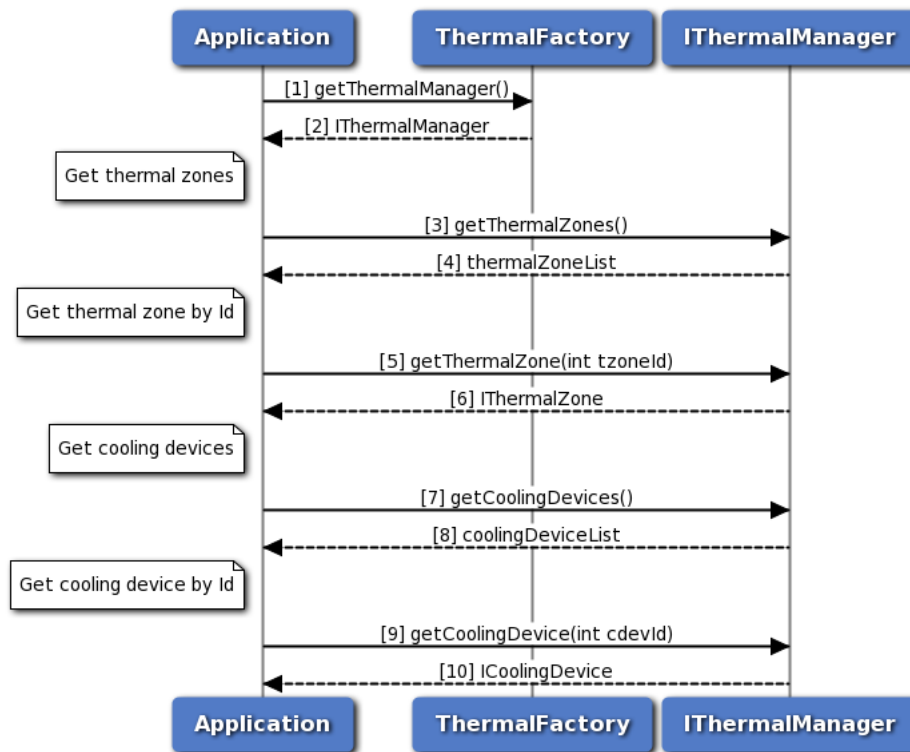


Figure 3-77 Thermal manager call flow

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

3.9 Thermal shutdown management

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

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

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

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

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

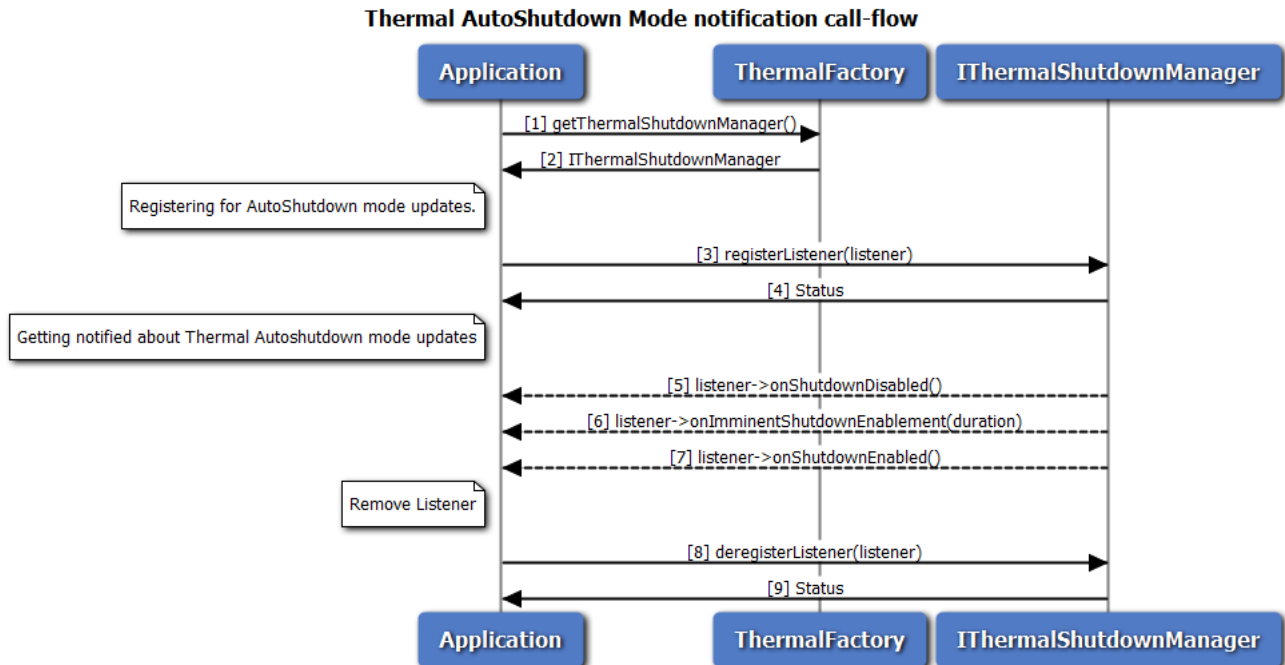


Figure 3-78 Call flow to register/remove listener for Thermal shutdown manager

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

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

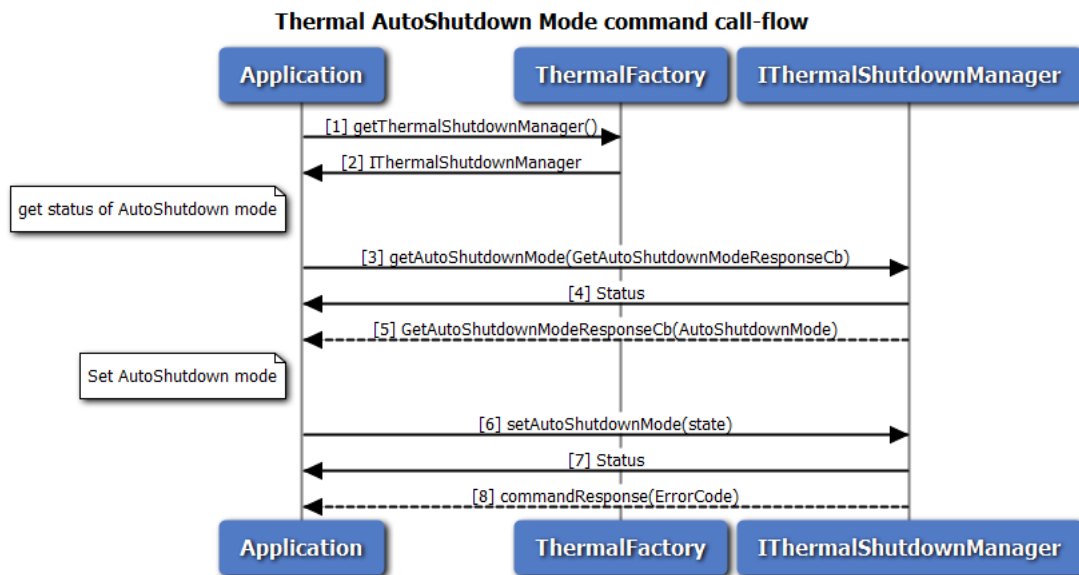


Figure 3-79 Call flow to set/get the Thermal auto-shutdown mode

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

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

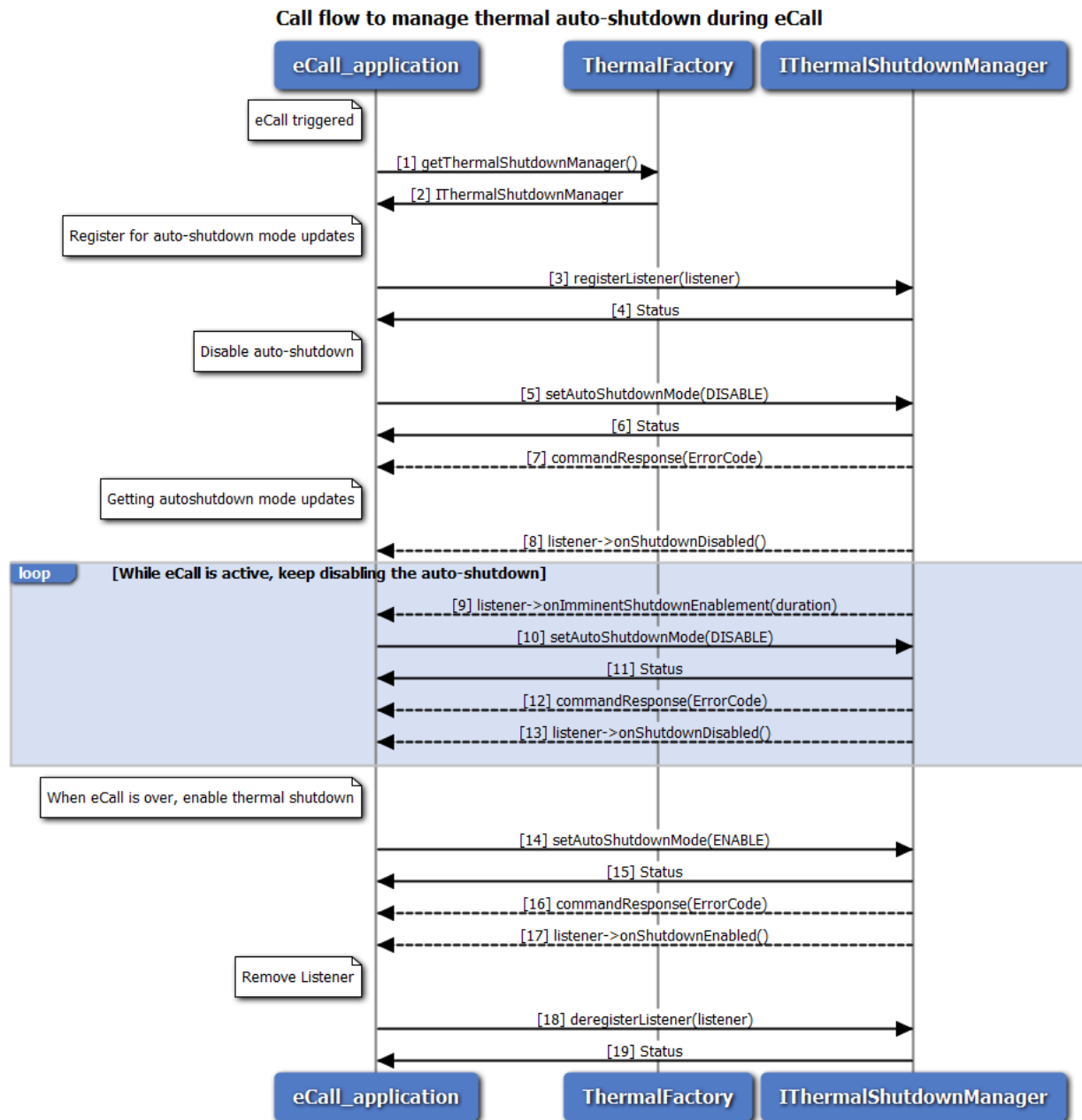


Figure 3-80 Call flow to manage thermal auto-shutdown from an eCall application

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

auto-shutdown during eCall.

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

3.10 TCU Activity Management

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

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

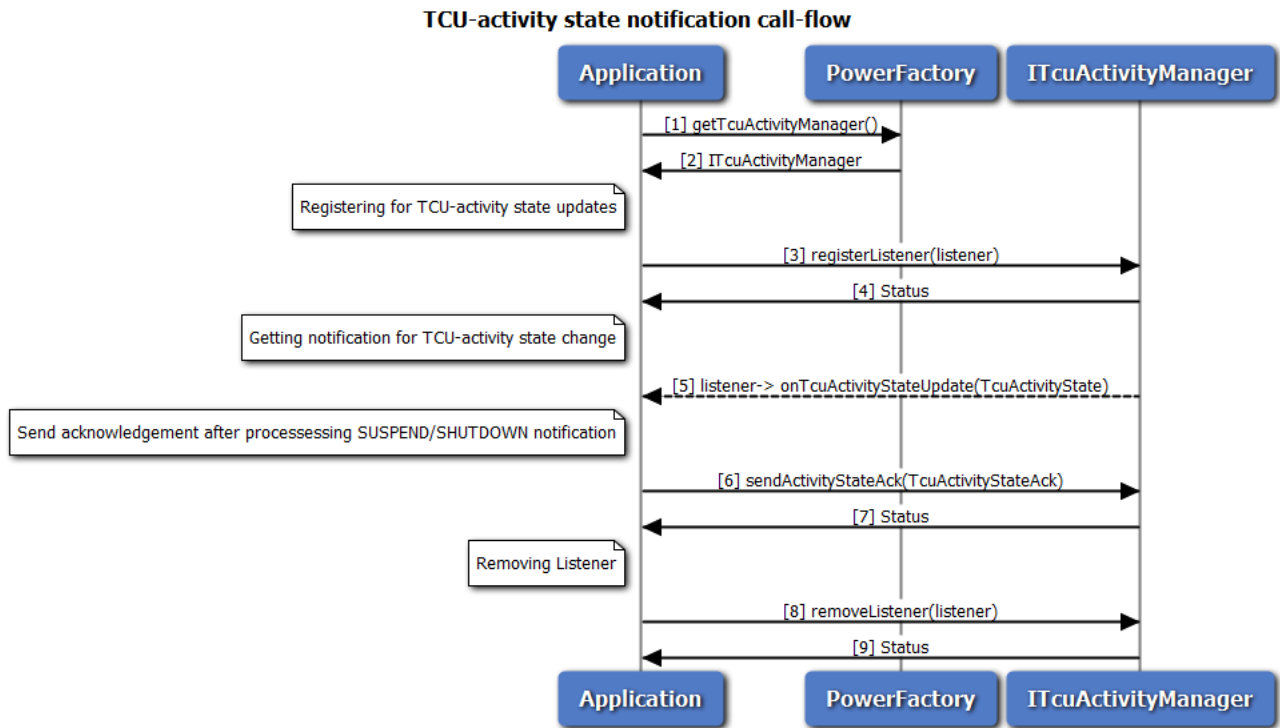


Figure 3-81 Call flow to register/remove listener for TCU-activity manager

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

3.10.2 Call flow to set the TCU-activity state

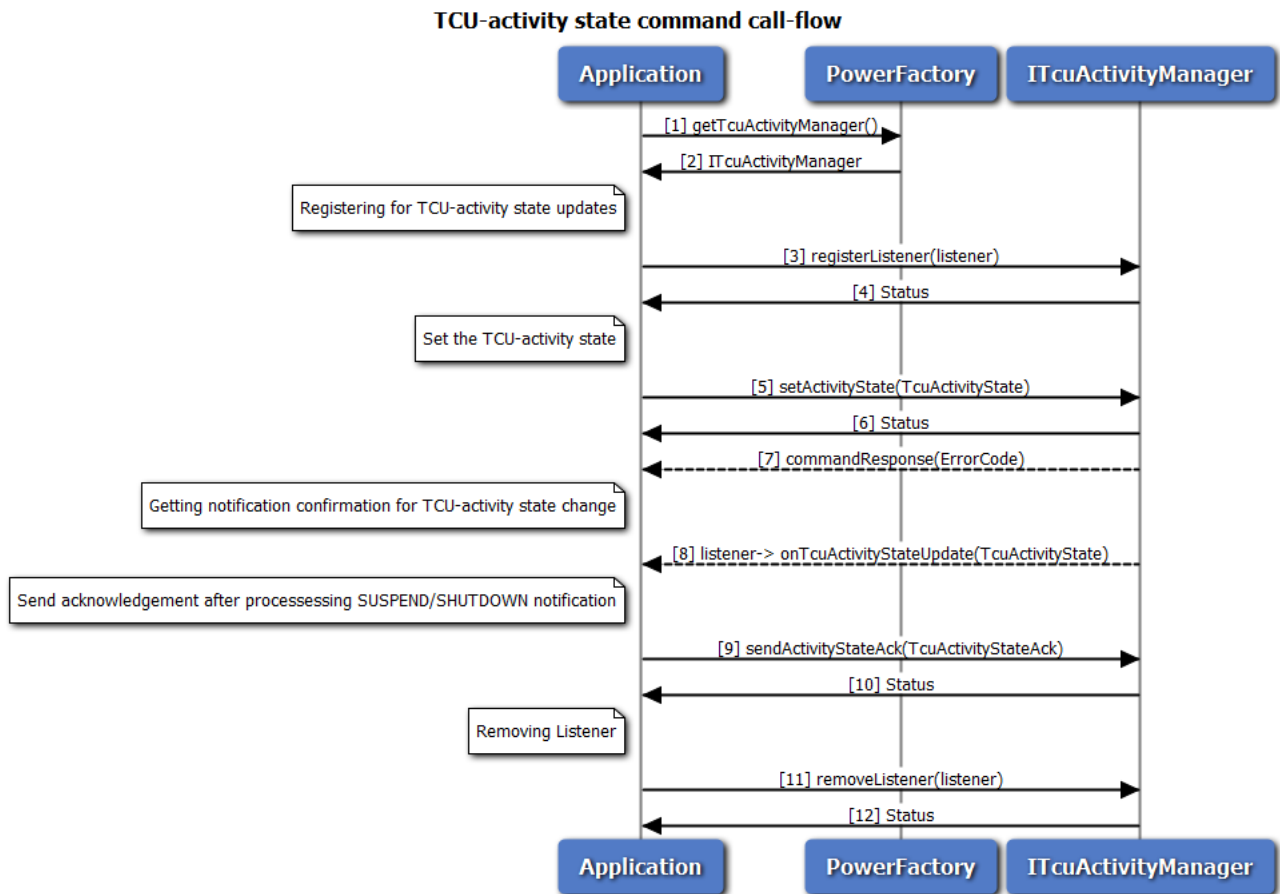


Figure 3-82 Call flow to set the TCU-activity state

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

successfully.

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

3.11 Remote SIM call flow

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

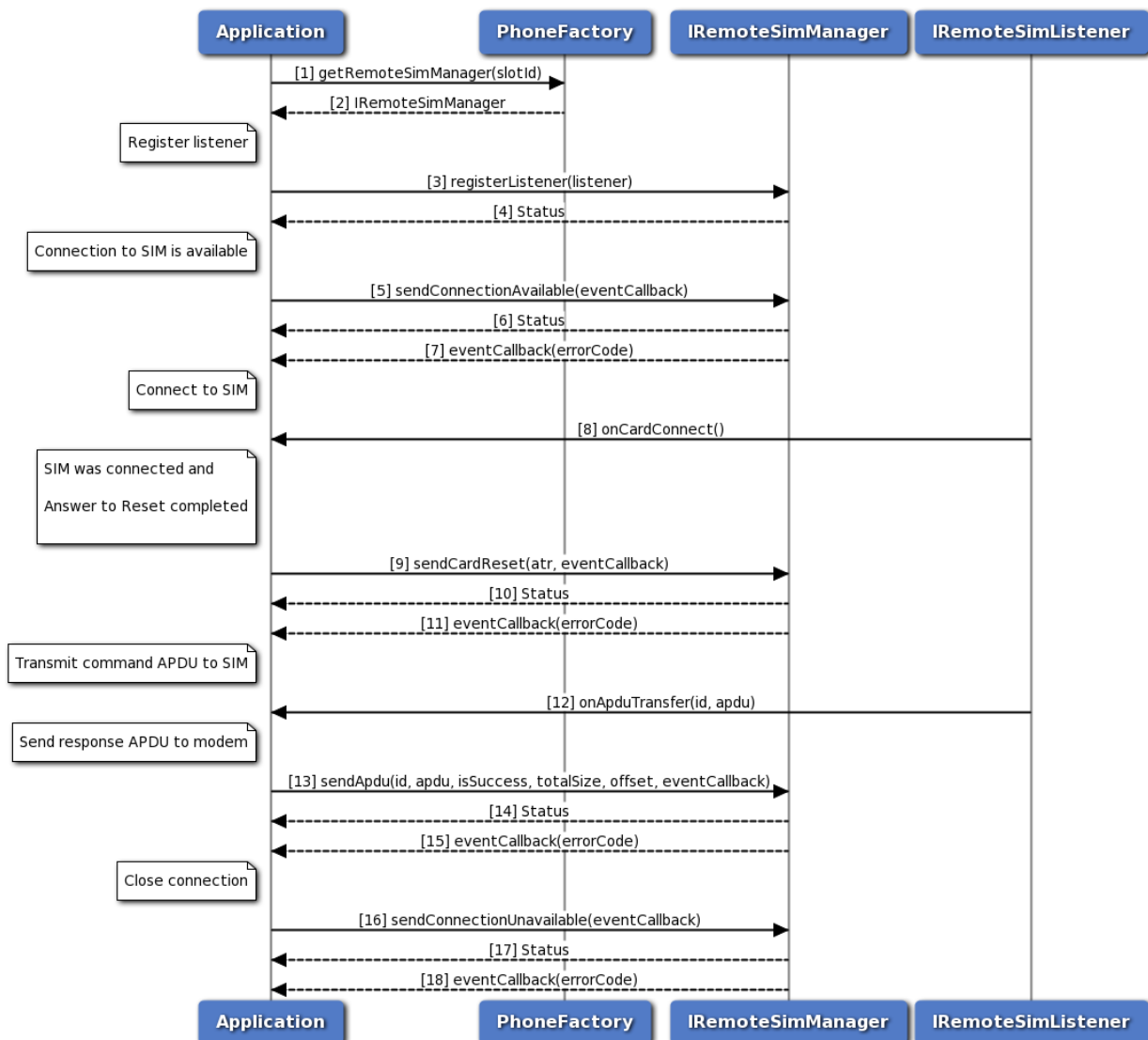


Figure 3-83 Remote SIM call flow

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

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

3.12 Modem Config Call Flow

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

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

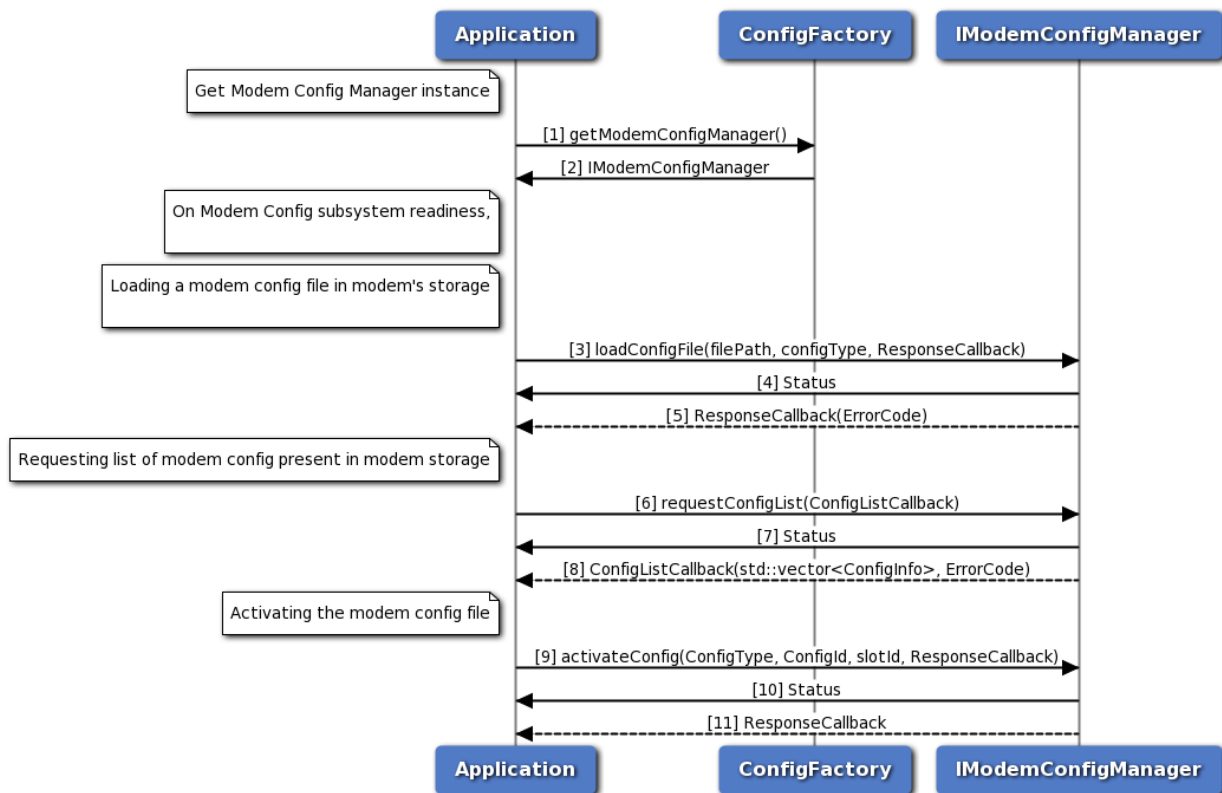


Figure 3-84 Modem Config load and activate call flow

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

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

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

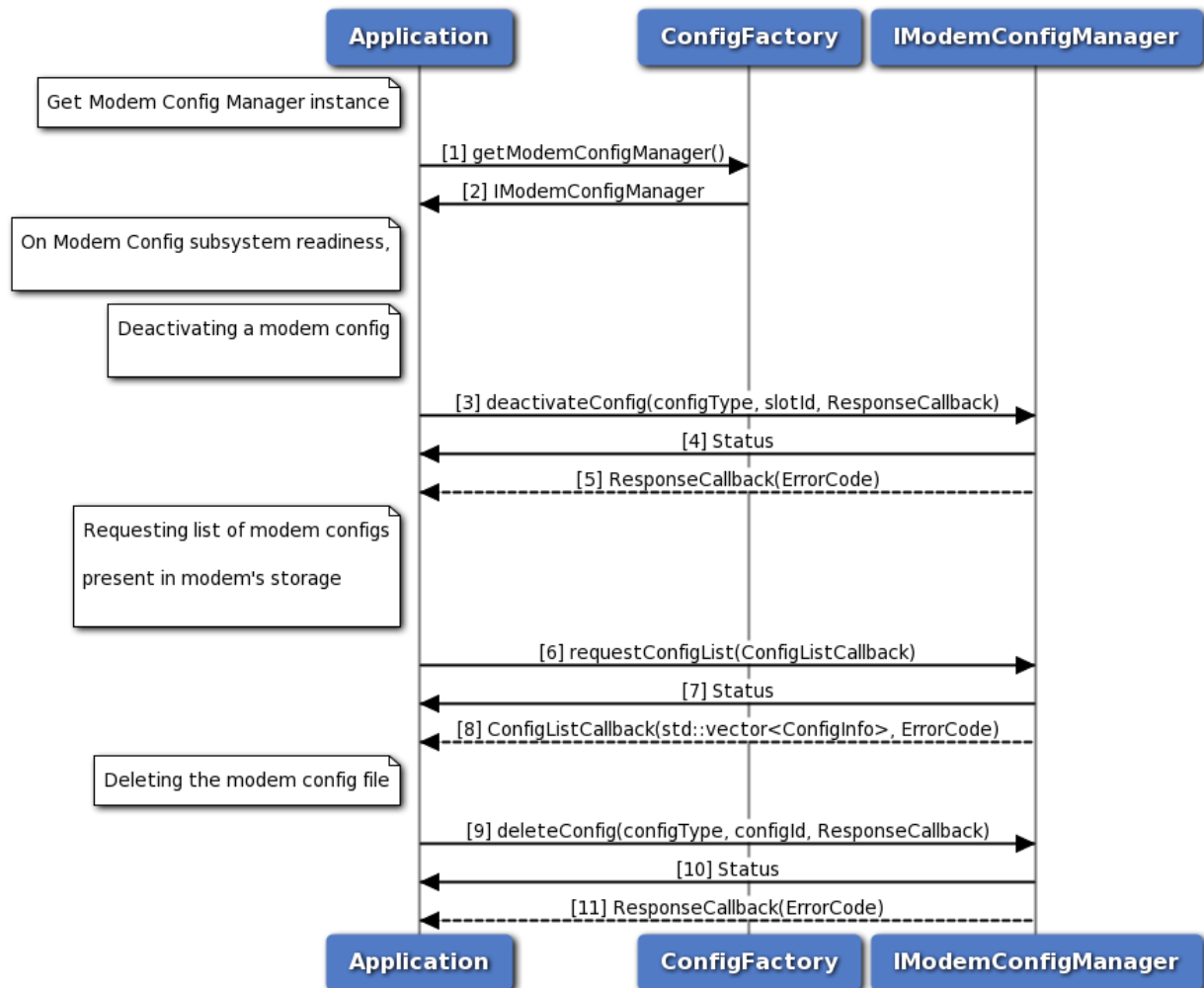


Figure 3-85 Modem Config deactivate and delete Call Flow

1. Application requests modem config manager object from config factory.
2. Config factory returns IModemConfigManager object.
3. Application sends a request to deactivate config file.
4. Application receives synchronous Status which indicates if the request to deactivate config file was sent successfully.
5. Application is notified of the Status of the deactivateConfig 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 delete config file.
10. Application receives synchronous Status which indicates if the request to delete config file was sent successfully.
11. Application is notified of the Status of the deleteConfig request (either SUCCESS or FAILED) via the application-supplied callback.

3.12.3 Call flow to set and get config auto selection mode

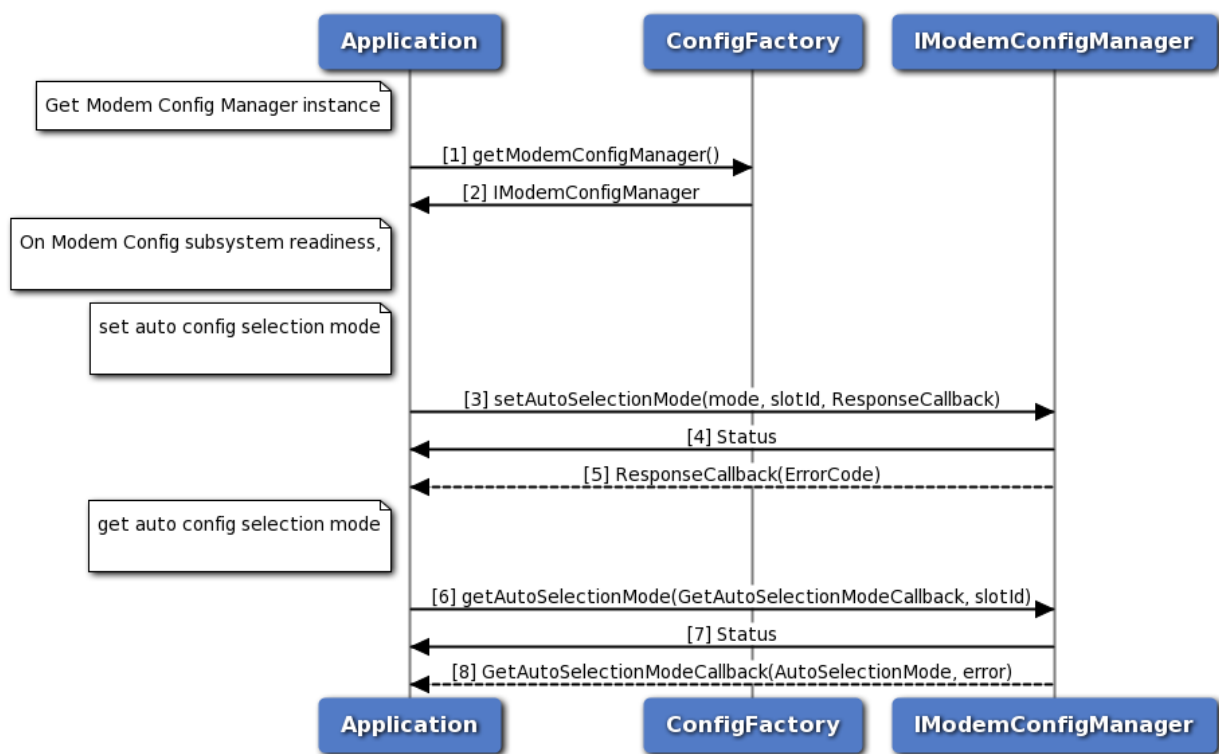


Figure 3-86 Modem Config get and set Auto Selection Mode Call Flow

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

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

3.13 WLAN

3.13.1 Call flow to Modify WLAN Configuration

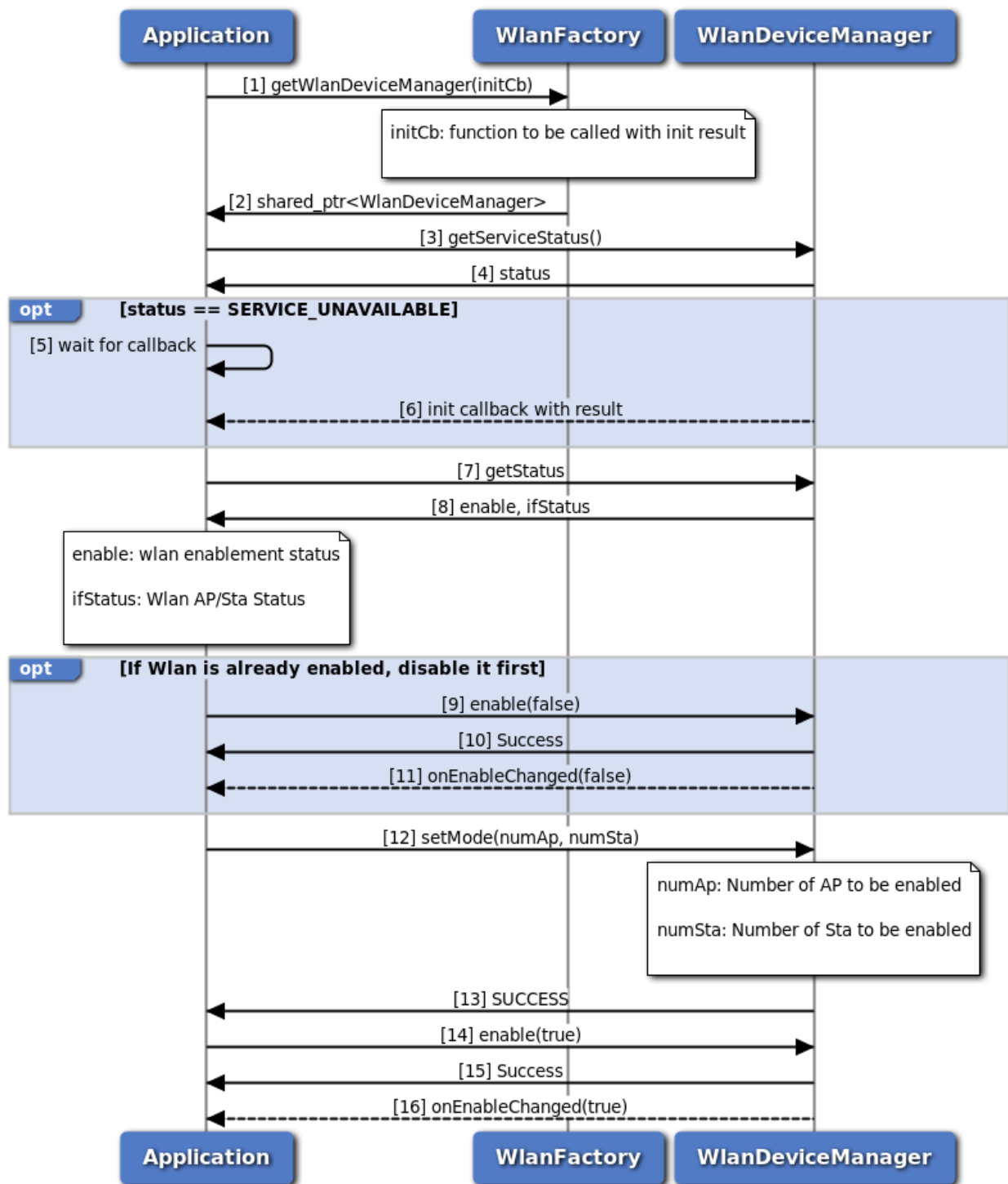


Figure 3-87 Modify WLAN Configuration Call Flow

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

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

3.13.2 Call flow to Modify WLAN Station Configuration

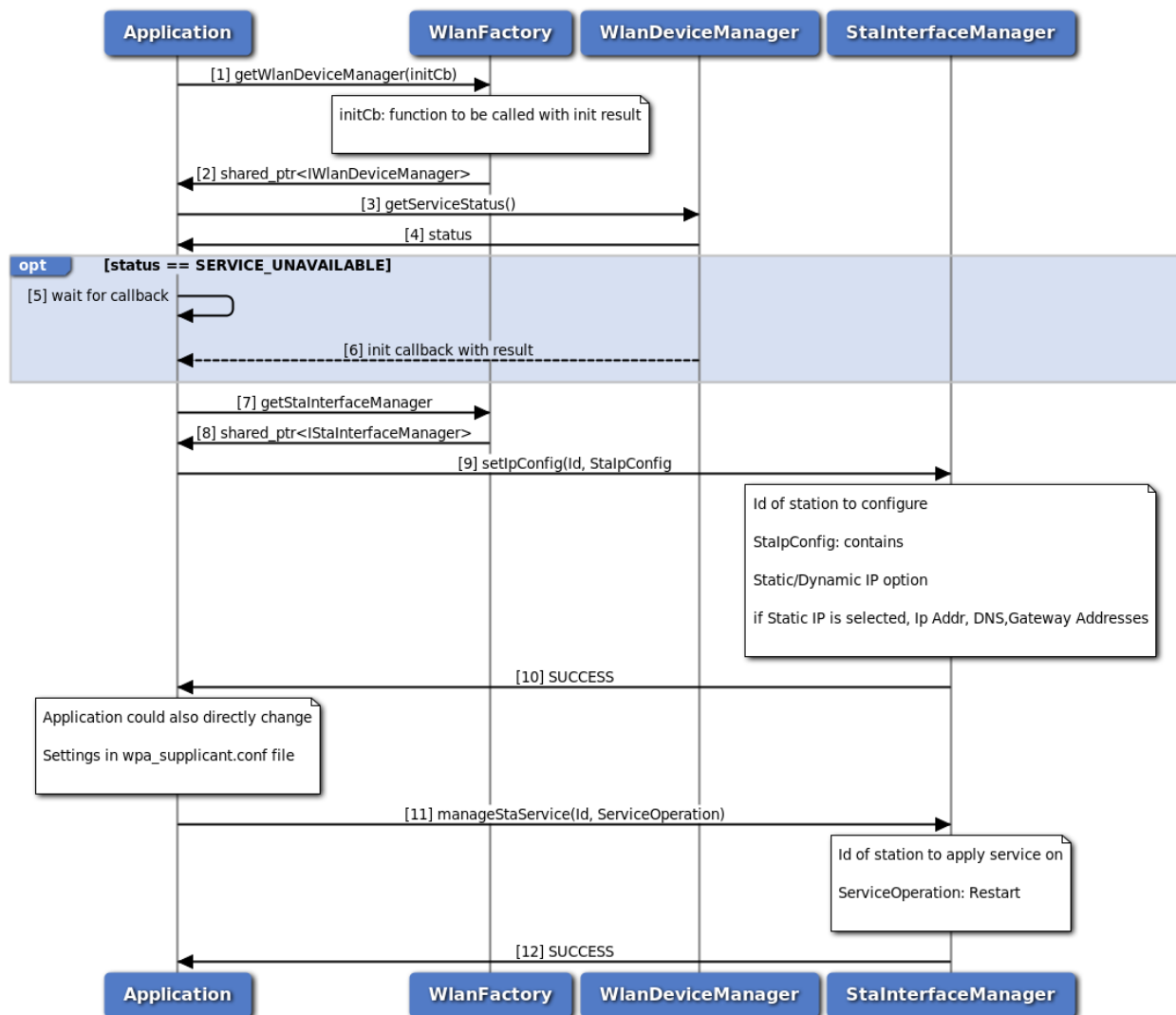


Figure 3-88 Modify WLAN Station Configuration Call Flow

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

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

3.13.3 Call flow to Modify WLAN Access Point Configuration

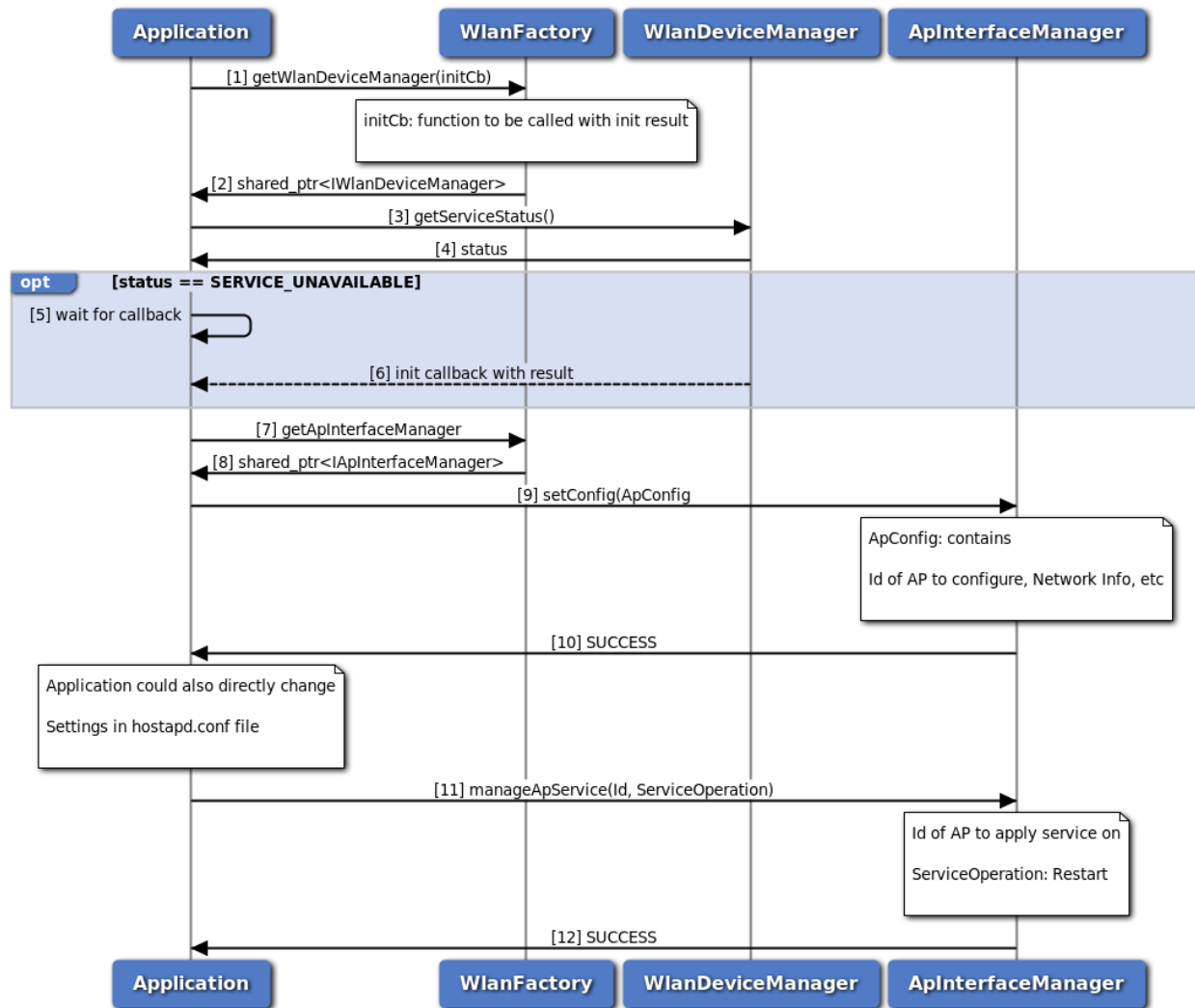


Figure 3-89 Modify WLAN Access Point Configuration Call Flow

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

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

4 Deprecated List

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

use onStatusChanged(Cv2xStatusEx status)

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

Use requestCapabilities() API

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

use onSpsSchedulingChanged

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

use onStatusChanged in Cv2xListener

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

use onStatusChanged in Cv2xListener

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

use requestCv2xStatus(RequestCv2xCalbackEx)

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

Use ICv2xConfig instead

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

use RequestCv2xStatusCallbackEx

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

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

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

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

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

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

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

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

Global **telux::data::net::IFirewallManager::addFirewallEntry** (int profileId, std::shared_ptr< IFirewallEntry > entry, **telux::common::ResponseCallback** callback=nullptr)=0

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

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

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

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

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

Global **telux::data::net::IFirewallManager::removeFirewallEntry** (int profileId, uint32_t handle, **telux::common::ResponseCallback** callback=nullptr)=0

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

Global **telux::data::net::IFirewallManager::requestDmzEntry** (int profileId, DmzEntriesCb dmzCb)=0

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

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

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

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

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

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

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

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

Use bindToBackhaul() API below to bind VLAN to backhaul

Global **telux::data::net::IVlanManager::queryVlanMappingList** (VlanMappingResponseCb callback)=0

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

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

Use unbindFromBackhaul() API below to unbind VLAN to backhaul

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

constellation type is not supported.

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

constellation type is not supported.

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

This API not being supported

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

This API not being supported

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

Use another onECallMsdTransmissionStatus() API with argument ECallMsdTransmissionStatus

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

Use IPhoneManager::requestOperatingMode() API instead

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

Use requestVoiceServiceState() API

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

Use IPhoneManager::setOperatingMode() API instead

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

Use onOperatingModeChanged() API instead

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

Use onVoiceServiceStateChanged() listener

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

Use requestSapState() API below to get SAP state

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

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

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

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

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

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

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

This API not being supported

As of version 1.54.0 this API is no longer supported.

Global `telux::tel::LteSignalStrengthInfo::getTimingAdvance () const`

This API not being supported

Global `telux::tel::ServiceState`

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

Global `telux::tel::SmsMessage::getPdu () const`

Use API `SmsMessage::getRawPdu`

Global `telux::tel::WcdmaSignalStrengthInfo::getBitErrorRate () const`

This API not being supported

Global `telux::wlan::IapInterfaceManager::getConnectedDevicesStats (std::vector< DeviceStats > &clientsStats)=0`

This API is no longer supported.

5 Interfaces

5.1 Telematics SDK APIs

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

5.2 Phone Factory

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

5.2.1 Data Structure Documentation

5.2.1.1 class telux::tel::PhoneFactory

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

Public member functions

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

Static Public Member Functions

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

5.2.1.1.1 Member Function Documentation

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

Get Phone Factory instance.

5.2.1.1.1.2 [std::shared_ptr<IPhoneManager>](#) [telux::tel::PhoneFactory::getPhoneManager](#) ()

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

Returns

Pointer of [IPhoneManager](#) object.

5.2.1.1.1.3 [std::shared_ptr<ISmsManager>](#) [telux::tel::PhoneFactory::getSmsManager](#) (int *phoneId* = *DEFAULT_PHONE_ID*, [telux::common::InitResponseCb](#) *callback* = *nullptr*)

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

Parameters

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

Returns

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

5.2.1.1.1.4 [std::shared_ptr<ICallManager>](#) [telux::tel::PhoneFactory::getCallManager](#) ([telux::common::InitResponseCb](#) *callback* = *nullptr*)

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

Parameters

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

Returns

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

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

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

Returns

Pointer of [ICardManager](#) object.

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

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

Parameters

in	slotId	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	slotId	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	slotId	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	slotId	Unique identifier for the SIM slot

Returns

Pointer of [IRemoteSimManager](#) object.

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

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

Returns

Pointer of [IMultiSimManager](#) object.

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

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

Returns

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

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

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

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

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

Parameters

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

Returns

Pointer of [IImSettingsManager](#) object.

Note

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

5.2.1.1.1.16 `std::shared_ptr<IImServingSystemManager> telux::tel::PhoneFactory::getImServingSystemManager ( SlotId slotId, telux::common::InitResponseCb callback = nullptr )`

Get IMS Serving System Manager instance to query IMS registration status

Returns

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

Note

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

5.3 Phone

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

5.3.1 Data Structure Documentation

5.3.1.1 class telux::tel::GsmCellIdentity

[GsmCellIdentity](#) class provides methods to get mobile country code, mobile network code, location area code, cell identity, absolute RF channel number and base station identity code.

Public member functions

- [GsmCellIdentity](#) (int mcc, int mnc, int lac, int cid, int arfcn, int bsic)
- const int [getMcc](#) ()
- const int [getMnc](#) ()
- const int [getLac](#) ()
- const int [getIdentity](#) ()
- const int [getArfcn](#) ()
- const int [getBaseStationIdentityCode](#) ()

5.3.1.1.1 Constructors and Destructors

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

5.3.1.1.2 Member Function Documentation

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

Get the Mobile Country Code.

Returns

Mcc value.

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

Get the Mobile Network Code.

Returns

Mnc value.

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

Get the location area code.

Returns

Location area code.

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

Get the cell identity.

Returns

Cell identity.

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

Get the absolute RF channel number.

Returns

Absolute RF channel number.

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

Get the base station identity code.

Returns

Base station identity code.

5.3.1.2 class telux::tel::CdmaCellIdentity

[CdmaCellIdentity](#) class provides methods to get the network identifier, system identifier, base station identifier, longitude and latitude.

Public member functions

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

5.3.1.2.1 Constructors and Destructors

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

5.3.1.2.2 Member Function Documentation

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

Get the network identifier.

Returns

Network identifier.

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

Get the system identifier.

Returns

System identifier.

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

Get the base station identifier.

Returns

Base station identifier.

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

Get the longitude.

Returns

Longitude.

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

Get the latitude.

Returns

Latitude.

5.3.1.3 class telux::tel::LteCellIdentity

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

Public member functions

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

5.3.1.3.1 Constructors and Destructors

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

5.3.1.3.2 Member Function Documentation

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

Get the Mobile Country Code.

Returns

Mcc value.

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

Get the Mobile Network Code.

Returns

Mnc value.

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

Get the cell identity.

Returns

Cell identity.

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

Get the physical cell identifier.

Returns

Physical cell identifier.

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

Get the tracking area code.

Returns

Tracking area code.

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

Get the absolute RF channel number.

Returns

Absolute RF channel number.

5.3.1.4 `class telux::tel::WcdmaCellIdentity`

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

Public member functions

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

5.3.1.4.1 Constructors and Destructors

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

5.3.1.4.2 Member Function Documentation

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

Get the Mobile Country Code.

Returns

Mcc value.

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

Get the Mobile Network Code.

Returns

Mnc value.

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

Get the location area code.

Returns

Location area code.

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

Get the cell identity.

Returns

Cell identity.

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

Get the primary scrambling code.

Returns

Primary scrambling code.

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

Get the absolute RF channel number.

Returns

Absolute RF channel number.

5.3.1.5 class telux::tel::TdscdmaCellIdentity

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

Public member functions

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

5.3.1.5.1 Constructors and Destructors

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

5.3.1.5.2 Member Function Documentation

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

Get the Mobile Country Code.

Returns

Mcc value.

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

Get the Mobile Network Code.

Returns

Mnc value.

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

Get the location area code

Returns

Location area code.

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

Get the cell identity.

Returns

Cell identity.

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

Get the cell parameters identifier.

Returns

Cell parameters identifier.

5.3.1.6 `class telux::tel::CellInfo`

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

Public member functions

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

Protected Attributes

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

5.3.1.6.1 Member Function Documentation

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

Get the cell type.

Returns

CellType.

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

Checks whether the current cell is registered or not.

Returns

If true cell is registered or vice-versa.

5.3.1.6.2 Field Documentation

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

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

5.3.1.7 class `telux::tel::GsmCellInfo`

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

Public member functions

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

Additional Inherited Members

5.3.1.7.1 Constructors and Destructors

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

[GsmCellInfo](#) constructor.

Parameters

in	<i>registered</i>	- Registration status of the cell.
in	<i>id</i>	- GSM cell identity.
in	<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 the availability of some optional parameters in MSD as per European eCall MSD standard EN 15722.

Public member functions

- [ECallMsdOptionals \(\)](#)

Data Fields

- [ECallOptionalDataType optionalDataType](#)
- bool [optionalDataPresent](#)
- bool [recentVehicleLocationN1Present](#)
- bool [recentVehicleLocationN2Present](#)
- bool [numberOfPassengersPresent](#)

5.3.1.12.1 Constructors and Destructors

5.3.1.12.1.1 `telux::tel::ECallMsdOptionals::ECallMsdOptionals ( )`

5.3.1.12.2 Field Documentation

5.3.1.12.2.1 `ECallOptionalDataType telux::tel::ECallMsdOptionals::optionalDataType`

Type of optional data

5.3.1.12.2.2 `bool telux::tel::ECallMsdOptionals::optionalDataPresent`

Availability of Optional data:

true - Present or false - Absent

5.3.1.12.2.3 `bool telux::tel::ECallMsdOptionals::recentVehicleLocationN1Present`

Availability of Recent Vehicle Location N1 data:

true - Present or false - Absent. In MSD version-3 (as per EN 15722:2020), as recentVehicleLocationN1 is mandatory, this should be set to true by client

5.3.1.12.2.4 `bool telux::tel::ECallMsdOptionals::recentVehicleLocationN2Present`

Availability of Recent Vehicle Location N2 data:

true - Present or false - Absent. In MSD version-3 (as per EN 15722:2020), as recentVehicleLocationN2 is mandatory, this should be set to true by client

5.3.1.12.2.5 bool telux::tel::ECallMsdOptionals::numberOfPassengersPresent

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.

Public member functions

- [ECallMsdControlBits](#) ()

Data Fields

- bool [automaticActivation](#)
- bool [testCall](#)
- bool [positionCanBeTrusted](#)
- [ECallVehicleType](#) [vehicleType](#): 5

5.3.1.13.1 Constructors and Destructors

5.3.1.13.1.1 telux::tel::ECallMsdControlBits::ECallMsdControlBits ()

5.3.1.13.2 Field Documentation

5.3.1.13.2.1 bool telux::tel::ECallMsdControlBits::automaticActivation

auto / manual activation

5.3.1.13.2.2 bool telux::tel::ECallMsdControlBits::testCall

test / emergency call

5.3.1.13.2.3 bool telux::tel::ECallMsdControlBits::positionCanBeTrusted

false if coincidence < 95% of reported pos within +/- 150m

5.3.1.13.2.4 ECallVehicleType telux::tel::ECallMsdControlBits::vehicleType

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.

Public member functions

- [ECallVehicleIdentificationNumber](#) ()

Data Fields

- std::string [isowmi](#)
- std::string [isovds](#)
- std::string [isovisModelyear](#)
- std::string [isovisSeqPlant](#)

5.3.1.14.1 Constructors and Destructors

5.3.1.14.1.1 `telux::tel::ECallVehicleIdentificationNumber::ECallVehicleIdentificationNumber ( )`

5.3.1.14.2 Field Documentation

5.3.1.14.2.1 `std::string telux::tel::ECallVehicleIdentificationNumber::isowmi`

World Manufacturer Index (WMI)

5.3.1.14.2.2 `std::string telux::tel::ECallVehicleIdentificationNumber::isovds`

Vehicle Type Descriptor (VDS)

5.3.1.14.2.3 `std::string telux::tel::ECallVehicleIdentificationNumber::isovisModelyear`

Model year from Vehicle Identifier Section (VIS)

5.3.1.14.2.4 `std::string telux::tel::ECallVehicleIdentificationNumber::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

Public member functions

- [ECallVehiclePropulsionStorageType](#) ()

Data Fields

- bool [gasolineTankPresent](#)
- bool [dieselTankPresent](#)
- bool [compressedNaturalGas](#)
- bool [liquidPropaneGas](#)
- bool [electricEnergyStorage](#)
- bool [hydrogenStorage](#)
- bool [otherStorage](#)

5.3.1.15.1 Constructors and Destructors

5.3.1.15.1.1 `telux::tel::ECallVehiclePropulsionStorageType::ECallVehiclePropulsionStorageType ( )`

5.3.1.15.2 Field Documentation

5.3.1.15.2.1 `bool telux::tel::ECallVehiclePropulsionStorageType::gasolineTankPresent`

Represents the presence of Gasoline Tank in the vehicle.

5.3.1.15.2.2 `bool telux::tel::ECallVehiclePropulsionStorageType::dieselTankPresent`

Represents the presence of Diesel Tank in the vehicle

5.3.1.15.2.3 `bool telux::tel::ECallVehiclePropulsionStorageType::compressedNaturalGas`

Represents the presence of CNG in the vehicle

5.3.1.15.2.4 `bool telux::tel::ECallVehiclePropulsionStorageType::liquidPropaneGas`

Represents the presence of Liquid Propane Gas in the vehicle

5.3.1.15.2.5 `bool telux::tel::ECallVehiclePropulsionStorageType::electricEnergyStorage`

Represents the presence of Electronic Storage in the vehicle

5.3.1.15.2.6 `bool telux::tel::ECallVehiclePropulsionStorageType::hydrogenStorage`

Represents the presence of Hydrogen Storage in the vehicle

5.3.1.15.2.7 `bool telux::tel::ECallVehiclePropulsionStorageType::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.

Public member functions

- [ECallVehicleLocation \(\)](#)

Data Fields

- int32_t [positionLatitude](#)
- int32_t [positionLongitude](#)

5.3.1.16.1 Constructors and Destructors

5.3.1.16.1.1 telux::tel::ECallVehicleLocation::ECallVehicleLocation ()

5.3.1.16.2 Field Documentation

5.3.1.16.2.1 int32_t telux::tel::ECallVehicleLocation::positionLatitude

latitude in milliarcsec, range is (-2147483648 to 2147483647)

5.3.1.16.2.2 int32_t telux::tel::ECallVehicleLocation::positionLongitude

longitude in milliarcsec, range is (-2147483648 to 2147483647)

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.

Public member functions

- [ECallVehicleLocationDelta \(\)](#)

Data Fields

- int16_t [latitudeDelta](#)
- int16_t [longitudeDelta](#)

5.3.1.17.1 Constructors and Destructors

5.3.1.17.1.1 telux::tel::ECallVehicleLocationDelta::ECallVehicleLocationDelta ()

5.3.1.17.2 Field Documentation

5.3.1.17.2.1 int16_t telux::tel::ECallVehicleLocationDelta::latitudeDelta

(1 Unit = 100 milliarcseconds, range: -512 to 511)

5.3.1.17.2.2 int16_t telux::tel::ECallVehicleLocationDelta::longitudeDelta

(1 Unit = 100 milliarcseconds, range: -512 to 511)

5.3.1.18 struct telux::tel::ECallDeltaV

Defines delta-v parameters as per Euro NCAP Technical Bulletin TB 040.

Data fields

Type	Field	Description
uint8_t	rangeLimit	Upper limit of the detection range for delta-v. The range is unsigned integer[100 to 255].
int16_t	deltaVX	Difference in velocity just before and just after (start of the) triggering incident measured over the X-axis of the vehicle coordinate system. The range is signed integer[-255 to 255].
int16_t	deltaVY	Difference in velocity just before and just after (start of the) triggering incident measured over the Y-axis of the vehicle coordinate system. The range is signed integer[-255 to 255].

5.3.1.19 struct telux::tel::ECallOptionalEuroNcapData

Optional additional data information as per Euro NCAP Technical Bulletin TB 040.

Data fields

Type	Field	Description
ECallLocation-OfImpact	locationOf-Impact	The impact location of the triggering incident.
bool	rollOver-Detected-Present	Availability of rollover detected: true - Present or false - Absent
bool	rollOver-Detected	(Optional) Omitted if vehicle is not able to detect a rollover, else true or false.

Type	Field	Description
ECallDeltaV	deltaV	Difference between velocity just after and just before impact (delta-v).

5.3.1.20 struct telux::tel::ECallOptionalPdu

Optional additional data information for the emergency rescue service.

Data fields

Type	Field	Description
ECallDefault-Options	eCallDefault-Options	Optional information. This field is unused and deprecated. Use the other fields below, instead.
string	oid	Relative object identifier(OID) as per European standard i.e. EN 15722
vector< uint8_t >	data	Optional additional data content.

5.3.1.21 struct telux::tel::ECallMsdData

Data structure to hold all details required to construct an MSD. Supports MSD version-2(as per EN 15722:2015) and MSD version-3(as per EN 15722:2020)

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 eCall and to be incremented with every 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

Type	Field	Description
ECallVehicle-Propulsion-StorageType	vehicle-Propulsion-Storage	<code>VehiclePropulsionStorageType</code> structure as per European standard i.e. EN 15722
<code>uint32_t</code>	timestamp	Seconds elapsed since midnight 01.01.1970 UTC
ECallVehicle-Location	vehicleLocation	<code>VehicleLocation</code> structure as per European standard. i.e. EN 15722
<code>uint8_t</code>	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. Optional field for MSD version-2
ECallVehicle-LocationDelta	recentVehicle-LocationN2	Change in latitude and longitude compared to the last but one MSD transmission. Optional field for MSD version-2
<code>uint8_t</code>	numberOf-Passengers	Number of occupants in the vehicle. Optional field for MSD version-2 and version-3
ECallOptional-Pdu	optionalPdu	Optional information for the emergency rescue service (103 bytes, ASN.1 encoded); may also point to an address, where this information is located Optional additional data information for the emergency rescue service.
<code>uint8_t</code>	msdVersion	MSD format version that is being used

5.3.1.22 struct telux::tel::ECallModelInfo

Represents eCall operating mode information

Data fields

Type	Field	Description
ECallMode	mode	Represents eCall operating mode
ECallMode-Reason	reason	Represents eCall operating mode change reason

5.3.1.23 struct telux::tel::ECallHlapTimerStatus

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

Data fields

Type	Field	Description
HlapTimer-Status	t2	T2 Timer status
HlapTimer-Status	t5	T5 Timer status
HlapTimer-Status	t6	T6 Timer status
HlapTimer-Status	t7	T7 Timer status
HlapTimer-Status	t9	T9 Timer status

5.3.1.24 struct telux::tel::ECallHlapTimerEvents

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

Data fields

Type	Field	Description
HlapTimer-Event	t2	T2 Timer event
HlapTimer-Event	t5	T5 Timer event
HlapTimer-Event	t6	T6 Timer event
HlapTimer-Event	t7	T7 Timer event
HlapTimer-Event	t9	T9 Timer event

5.3.1.25 struct telux::tel::EcallConfig

Represents various configuration parameters related to automotive emergency call

Data fields

Type	Field	Description
EcallConfig-Validity	configValidity-Mask	Indicates the valid configuration parameters in the structure. A bit set to 1 denotes that the corresponding configuration parameter is valid
bool	muteRxAudio	
EcallNumType	numType	
string	overriddenNum	

Type	Field	Description
bool	useCannedMsd	
uint32_t	gnssUpdate-Interval	
uint32_t	t2Timer	
uint32_t	t7Timer	
uint32_t	t9Timer	
uint8_t	msdVersion	

5.3.1.26 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 ()`

5.3.1.26.1 Constructors and Destructors

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

5.3.1.26.2 Member Function Documentation

5.3.1.26.2.1 `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.

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

Get Radio state of device.

Returns

[RadioState](#)

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

5.3.1.26.2.3 `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 telux::tel::VoiceRadioTechResponseCb

Returns

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

5.3.1.26.2.4 `virtual ServiceState telux::tel::IPhone::getServiceState ( ) [pure virtual]`

Get service state of the phone.

Returns

[ServiceState](#)

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

5.3.1.26.2.5 `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 telux::tel::IVoiceServiceStateCallback .

Returns

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

5.3.1.26.2.6 `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

5.3.1.26.2.7 `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 tel::CellInfoCallback

Returns

Status of requestCellInfo i.e. success or suitable error

5.3.1.26.2.8 **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

5.3.1.26.2.9 **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.

5.3.1.26.2.10 **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>	- ECallMode
in	<i>callback</i>	- Callback function to get the response for set eCall operating mode request.

Returns

Status of setECallOperatingMode i.e. success or suitable error

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

Get the eCall operating mode

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

5.3.1.27 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](#) ()

5.3.1.27.1 Constructors and Destructors

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

5.3.1.27.2 Member Function Documentation

5.3.1.27.2.1 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 SUCCESS RADIO_NOT_AVAILABLE

5.3.1.28 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](#) ()

5.3.1.28.1 Constructors and Destructors

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

5.3.1.28.2 Member Function Documentation

5.3.1.28.2.1 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 telux::tel::VoiceServiceInfo
in	<i>error</i>	Return code for whether the operation succeeded or failed • telux::common::ErrorCode::SUCCESS • telux::common::ErrorCode::RADIO_NOT_AVAILABLE • telux::common::ErrorCode::GENERIC_FAILURE

5.3.1.29 struct [telux::tel::SimRatCapability](#)

Structure contains slotID and RAT capabilities corresponding to slot.

Data fields

Type	Field	Description
int	slotId	
RAT-Capabilities-Mask	capabilities	

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

5.3.1.31 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](#) ()

5.3.1.31.1 Constructors and Destructors

5.3.1.31.1.1 virtual telux::tel::IPhoneListener::~IPhoneListener () [virtual]

5.3.1.31.2 Member Function Documentation

5.3.1.31.2.1 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 ServiceState

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

5.3.1.31.2.2 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

5.3.1.31.2.3 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

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

This function is called when radio state changes on phone

in	<i>phone</i>	Unique id of the phone on which radio state changed
in	<i>radioState</i>	Radio state of the phone RadioState

Deprecated Use [onOperatingModeChanged\(\)](#) API instead

5.3.1.31.2.5 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 telux::tel::RadioTechnology

5.3.1.31.2.6 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 telux::tel::VoiceServiceInfo

5.3.1.31.2.7 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 OperatingMode .

5.3.1.31.2.8 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 ECallModeInfo

5.3.1.32 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` ()

5.3.1.32.1 Constructors and Destructors

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

5.3.1.32.2 Member Function Documentation

5.3.1.32.2.1 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).

5.3.1.32.2.2 `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.

5.3.1.32.2.3 `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.

5.3.1.32.2.4 `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.

5.3.1.32.2.5 `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.

5.3.1.32.2.6 `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.

5.3.1.32.2.7 `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.

5.3.1.32.2.8 `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.

5.3.1.32.2.9 `virtual telux::common::Status telux::tel::IPhoneManager::setOperatingMode ( OperatingMode operatingMode, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

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

in	<i>operatingMode</i>	Operating Mode to be set
in	<i>callback</i>	Optional callback pointer to get the response of set operatingmode request. In callback following error is returned. • telux::common::ErrorCode::INVALID_TRANSITION • telux::common::ErrorCode::INVALID_ARGUMENTS • telux::common::ErrorCode::DEVICE_IN_USE • telux::common::ErrorCode::NO_MEMORY

Returns

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

5.3.1.32.2.10 `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.

5.3.1.32.2.11 `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.

5.3.1.33 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` ()

5.3.1.33.1 Constructors and Destructors

5.3.1.33.1.1 virtual `telux::tel::ICellularCapabilityCallback::~~ICellularCapabilityCallback` ()
[virtual]

5.3.1.33.2 Member Function Documentation

5.3.1.33.2.1 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 • <code>telux::common::ErrorCode::SUCCESS</code> • <code>telux::common::ErrorCode::INTERNAL</code> • <code>telux::common::ErrorCode::NO_MEMORY</code>

5.3.1.34 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` ()

5.3.1.34.1 Constructors and Destructors

5.3.1.34.1.1 virtual `telux::tel::IOperatingModeCallback::~~IOperatingModeCallback` () [virtual]

5.3.1.34.2 Member Function Documentation

5.3.1.34.2.1 `virtual void telux::tel::IOperatingModeCallback::operatingModeResponse ( OperatingMode operatingMode, telux::common::ErrorCode error ) [virtual]`

This function is called with the response to requestOperatingMode API.

Parameters

in	<i>operatingMode</i>	OperatingMode
in	<i>error</i>	Return code for whether the operation succeeded or failed - telux::common::ErrorCode::SUCCESS - telux::common::ErrorCode::INTERNAL_ERR - telux::common::ErrorCode::NO_MEMORY

5.3.1.35 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< [Nr5gSignalStrengthInfo](#) > nr5gSignalStrengthInfo)
- std::shared_ptr< [LteSignalStrengthInfo](#) > [getLteSignalStrength](#) ()
- std::shared_ptr< [GsmSignalStrengthInfo](#) > [getGsmSignalStrength](#) ()
- std::shared_ptr< [CdmaSignalStrengthInfo](#) > [getCdmaSignalStrength](#) ()
- std::shared_ptr< [WcdmaSignalStrengthInfo](#) > [getWcdmaSignalStrength](#) ()
- std::shared_ptr< [TdscdmaSignalStrengthInfo](#) > [getTdscdmaSignalStrength](#) ()
- std::shared_ptr< [Nr5gSignalStrengthInfo](#) > [getNr5gSignalStrength](#) ()

5.3.1.35.1 Constructors and Destructors

5.3.1.35.1.1 `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, std::shared_ptr< Nr5gSignalStrengthInfo > nr5gSignalStrengthInfo)`

5.3.1.35.2 Member Function Documentation

5.3.1.35.2.1 `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.

5.3.1.35.2.2 `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.

5.3.1.35.2.3 `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.

5.3.1.35.2.4 `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.

5.3.1.35.2.5 `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.

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

Gives 5G NR signal strength instance.

Returns

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

5.3.1.36 `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

5.3.1.36.1 Constructors and Destructors

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

5.3.1.36.2 Member Function Documentation

5.3.1.36.2.1 `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.

5.3.1.36.2.2 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.

5.3.1.36.2.3 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.

5.3.1.36.2.4 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.

5.3.1.36.2.5 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.

5.3.1.36.2.6 const int telux::tel::LteSignalStrengthInfo::getLteChannelQualityIndicator () const

Get LTE channel quality indicator. (Valid value range [0, 15] and INVALID_SIGNAL_STRENGTH_VALUE i.e. unavailable).

Deprecated This API not being supported

Returns

LteCqi.

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

5.3.1.36.2.7 `const int telux::tel::LteSignalStrengthInfo::getTimingAdvance ( ) const`

Get the timing advance in micro seconds. (Valid value range [0, 0x7FFFFFFE] and INVALID_SIGNAL_STRENGTH_VALUE i.e. unavailable).

Deprecated This API not being supported

Returns

Timing advance value.

5.3.1.37 `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 ( )`

5.3.1.37.1 Constructors and Destructors

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

5.3.1.37.2 Member Function Documentation

5.3.1.37.2.1 `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.

5.3.1.37.2.2 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.

5.3.1.37.2.3 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.

5.3.1.37.2.4 const int telux::tel::GsmSignalStrengthInfo::getGsmBitErrorRate () const

Get the GSM bit error rate. (Valid value range [0, 7] and INVALID_SIGNAL_STRENGTH_VALUE i.e. unavailable).

Deprecated This API not being supported

Returns

GSM bit error rate.

5.3.1.37.2.5 const int telux::tel::GsmSignalStrengthInfo::getTimingAdvance ()

Get the timing advance in bit periods . 1 bit period = 48/13 us (Valid value range [0, 219] and INVALID_SIGNAL_STRENGTH_VALUE i.e. unavailable).

Deprecated This API not being supported

Returns

timing advance.

5.3.1.38 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

5.3.1.38.1 Constructors and Destructors

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

5.3.1.38.2 Member Function Documentation

5.3.1.38.2.1 `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.

5.3.1.38.2.2 `const int telux::tel::CdmaSignalStrengthInfo::getDbm ( ) const`

Get the signal strength in dBm.

Returns

Minimum value of Evdo dBm and Cdma dBm.

5.3.1.38.2.3 `const int telux::tel::CdmaSignalStrengthInfo::getCdmaEcio ( ) const`

Get the CDMA Ec/Io in dB.

Returns

CDMA Ec/Io.

5.3.1.38.2.4 const int telux::tel::CdmaSignalStrengthInfo::getEvdoEcio () const

Get the EVDO Ec/Io in dB.

Returns

EVDO Ec/Io.

5.3.1.38.2.5 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.

5.3.1.39 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

5.3.1.39.1 Constructors and Destructors**5.3.1.39.1.1 telux::tel::WcdmaSignalStrengthInfo::WcdmaSignalStrengthInfo (int *signalStrength*, int *bitErrorRate*)****5.3.1.39.2 Member Function Documentation****5.3.1.39.2.1 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.

5.3.1.39.2.2 `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.

5.3.1.39.2.3 `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.

5.3.1.39.2.4 `const int telux::tel::WcdmaSignalStrengthInfo::getBitErrorRate ( ) const`

Get the WCDMA bit error rate. (Valid value range [0, 7] and INVALID_SIGNAL_STRENGTH_VALUE i.e. unavailable).

Deprecated This API not being supported

Returns

WCDMA bit error rate.

5.3.1.40 `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`

5.3.1.40.1 Constructors and Destructors

5.3.1.40.1.1 `telux::tel::TdscdmaSignalStrengthInfo::TdscdmaSignalStrengthInfo ( int rscp )`

5.3.1.40.2 Member Function Documentation

5.3.1.40.2.1 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.

5.3.1.41 class telux::tel::Nr5gSignalStrengthInfo

5G NR signal strength provides methods to get signal strength and signal-to-noise ratio.

Public member functions

- [Nr5gSignalStrengthInfo](#) (int rsrp, int rsrq, int rssnr)
- const [SignalStrengthLevel](#) getLevel () const
- const int getDbm () const
- const int getReferenceSignalReceiveQuality () const
- const int getReferenceSignalSnr () const

5.3.1.41.1 Constructors and Destructors

5.3.1.41.1.1 telux::tel::Nr5gSignalStrengthInfo::Nr5gSignalStrengthInfo (int *rsrp*, int *rsrq*, int *rssnr*)

5.3.1.41.2 Member Function Documentation

5.3.1.41.2.1 const SignalStrengthLevel telux::tel::Nr5gSignalStrengthInfo::getLevel () const

Get signal level in the range.

Returns

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

5.3.1.41.2.2 const int telux::tel::Nr5gSignalStrengthInfo::getDbm () const

Get the signal strength in dBm. (Valid value range [-140, -44] and INVALID_SIGNAL_STRENGTH_VALUE i.e. unavailable). INVALID_SIGNAL_STRENGTH_VALUE indicates that modem is not in ENDC connected mode.

Returns

5G NR dBm value.

5.3.1.41.2.3 `const int telux::tel::Nr5gSignalStrengthInfo::getReferenceSignalReceiveQuality ( ) const`

Get 5G NR reference signal receive quality in dB. (Valid value range [-20, -3] and INVALID_SIGNAL_STRENGTH_VALUE i.e. unavailable). INVALID_SIGNAL_STRENGTH_VALUE indicates that modem is not in ENDC connected mode.

Returns

5G NR rsrq.

5.3.1.41.2.4 `const int telux::tel::Nr5gSignalStrengthInfo::getReferenceSignalSnr ( ) const`

Get 5G NR reference signal signal-to-noise ratio, multiply by 0.1 to get SNR in dB. (Valid value range [-200, +300] and INVALID_SIGNAL_STRENGTH_VALUE i.e. unavailable). (-200 = -20.0 dB, +300 = 30dB). INVALID_SIGNAL_STRENGTH_VALUE indicates that modem is not in ENDC connected mode.

Returns

5G NR signal-to-noise.

5.3.1.42 `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 ()

5.3.1.42.1 Constructors and Destructors

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

5.3.1.42.2 Member Function Documentation

5.3.1.42.2.1 VoiceServiceState telux::tel::VoiceServiceInfo::getVoiceServiceState ()

Get voice service state.

Returns

VoiceServiceState

5.3.1.42.2.2 VoiceServiceDenialCause telux::tel::VoiceServiceInfo::getVoiceServiceDenialCause ()

Get Voice service denial cause

Returns

VoiceServiceDenialCause

5.3.1.42.2.3 bool telux::tel::VoiceServiceInfo::isEmergency ()

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

5.3.1.42.2.4 bool telux::tel::VoiceServiceInfo::isInService ()

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

5.3.1.42.2.5 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

Defines all the cell info types.

Enumerator

GSM
CDMA
LTE
WCDMA
TDSCDMA

5.3.2.2 enum telux::tel::ECallVariant

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

Emergency Call Type

Enumerator

CALL_TYPE_ECALL eCall (0x0C)

5.3.2.4 enum telux::tel::ECallMsdTransmissionStatus

MSD Transmission Status

Enumerator

SUCCESS In-band MSD transmission is successful

FAILURE In-band MSD transmission failed

MSD_TRANSMISSION_STARTED In-band MSD transmission started

NACK_OUT_OF_ORDER Out of order NACK message detected during in-band MSD transmission

ACK_OUT_OF_ORDER Out of order ACK message detected during in-band MSD transmission

START_RECEIVED SEND-MSD(START) is received and SYNC is locked during in-band MSD transmission

LL_ACK_RECEIVED Link-Layer Acknowledgement(LL-ACK) is received during in-band MSD transmission

OUTBAND_MSD_TRANSMISSION_STARTED Outband MSD transmission started in NG eCall

OUTBAND_MSD_TRANSMISSION_SUCCESS Outband MSD transmission succeeded in NG eCall

OUTBAND_MSD_TRANSMISSION_FAILURE Outband MSD transmission failed in NG eCall

5.3.2.5 enum telux::tel::ECallCategory

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 15722:2020. Some of these values are only supported in certain MSD versions, so ensure to use supported values in an MSD. For example, TRAILERS_CLASS_O is not supported in MSD version-2 (as per A.1 in EN 15722:2015(E)), but supported in MSD version-3 (as per A.1 in EN 15722:2020).

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
TRAILERS_CLASS_O
AGRI_VEHICLES_CLASS_R
AGRI_VEHICLES_CLASS_S
AGRI_VEHICLES_CLASS_T
OFF_ROAD_VEHICLES_G
SPECIAL_PURPOSE_MOTOR_CARAVAN_CLASS_SA
SPECIAL_PURPOSE_ARMOURED_VEHICLE_CLASS_SB
SPECIAL_PURPOSE_AMBULANCE_CLASS_SC
SPECIAL_PURPOSE_HEARCE_CLASS_SD
OTHER_VEHICLE_CLASS

5.3.2.7 enum telux::tel::ECallOptionalDataType

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

Enumerator

ECALL_DEFAULT

5.3.2.8 enum telux::tel::ECallLocationOfImpact

Defines the impact location of the triggering incident as per Euro NCAP Technical Bulletin TB 040.

Enumerator

UNKNOWN Location of impact is unknown.
NONE No triggering impact detected.
FRONT At front of the car.
REAR At rear of the car.
DRIVER_SIDE At the driver side of the car.
NON_DRIVER_SIDE At the other side of the car.
OTHER At an unspecified location.

5.3.2.9 enum telux::tel::ECallMode

Represents eCall operating mode

Enumerator

NORMAL eCall and normal voice calls are allowed
ECALL_ONLY Only eCall is allowed
NONE Invalid mode

5.3.2.10 enum telux::tel::ECallModeReason

Represents eCall operating mode change reason

Enumerator

NORMAL eCall operating mode changed due to normal operation like setting of eCall mode
ERA_GLONASS eCall operating mode changed due to ERA-GLONASS operation

5.3.2.11 enum telux::tel::HlapTimerStatus

Represents the status of an eCall High Level Application Protocol(HLAP) timer that is maintained by the UE state machine.

Enumerator

UNKNOWN Unknown
INACTIVE eCall Timer is Inactive i.e
it has not started or it has stopped/expired
ACTIVE eCall Timer is Active i.e
it has started but not yet stopped/expired

5.3.2.12 enum telux::tel::HlapTimerEvent

Represents an event causing a change in the the status of eCall High Level Application Protocol (HLAP) timer that is maintained by the UE state machine.

The timer STARTED notification is provided when the timer moves from INACTIVE to ACTIVE state. The timer STOPPED notification is provided when the timer moves from ACTIVE to INACTIVE state, after its underlying condition is satisfied. The timer EXPIRED notification is provided when the timer moves from ACTIVE to INACTIVE state, after its underlying condition not satisfied until its timeout. The timer RESUMED notification is provided when the application restarts the timer after events like modem reset or a change of modem operating mode from low power mode to online using [telux::tel::ICallManager::restartECallHlapTimer\(\)](#).

Enumerator

UNKNOWN Unknown
UNCHANGED No change in timer status
STARTED eCall Timer is started
STOPPED eCall Timer is stopped

EXPIRED eCall Timer is expired

RESUMED eCall Timer is resumed. Applicable only for T9 and T10 timers

5.3.2.13 enum telux::tel::ECallNumType

Configuration that represents the type of the number to be dialed when an automotive emergency call is initiated.

Enumerator

DEFAULT

OVERRIDDEN

5.3.2.14 enum telux::tel::RadioState

Defines the radio state

Enumerator

RADIO_STATE_OFF Radio is explicitly powered off

RADIO_STATE_UNAVAILABLE Radio unavailable (eg, resetting or not booted)

RADIO_STATE_ON Radio is on

5.3.2.15 enum telux::tel::ServiceState

Defines the service states

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

Enumerator

EMERGENCY_ONLY Only emergency calls allowed

IN_SERVICE Normal operation, device is registered with a carrier and online

OUT_OF_SERVICE Device is not registered with any carrier

RADIO_OFF Device radio is off - Airplane mode for example

5.3.2.16 enum telux::tel::RadioTechnology

Defines all available radio access technologies

Enumerator

RADIO_TECH_UNKNOWN Network type is unknown

RADIO_TECH_GPRS Network type is GPRS

RADIO_TECH_EDGE Network type is EDGE

RADIO_TECH_UMTS Network type is UMTS

RADIO_TECH_IS95A Network type is IS95A

RADIO_TECH_IS95B Network type is IS95B

RADIO_TECH_1xRTT Network type is 1xRTT

RADIO_TECH_EVDO_0 Network type is EVDO revision 0

RADIO_TECH_EVDO_A Network type is EVDO revision A

RADIO_TECH_HSDPA Network type is HSDPA

RADIO_TECH_HSUPA Network type is HSUPA
RADIO_TECH_HSPA Network type is HSPA
RADIO_TECH_EVDO_B Network type is EVDO revision B
RADIO_TECH_EHRPD Network type is eHRPD
RADIO_TECH_LTE Network type is LTE
RADIO_TECH_HSPAP Network type is HSPA+
RADIO_TECH_GSM Network type is GSM, Only supports voice
RADIO_TECH_TD_SCDMA Network type is TD SCDMA
RADIO_TECH_IWLAN Network type is TD IWLAN
RADIO_TECH_LTE_CA Network type is LTE CA

5.3.2.17 enum telux::tel::RATCapability

Defines all available RAT capabilities for each subscription

Enumerator

AMPS
CDMA
HDR
GSM
WCDMA
LTE
TDS

5.3.2.18 enum telux::tel::VoiceServiceTechnology

Defines all voice support available on device

Enumerator

VOICE_TECH_GW_CSFB
VOICE_TECH_1x_CSFB
VOICE_TECH_VOLTE

5.3.2.19 enum telux::tel::OperatingMode

Defines operating modes of the device.

Enumerator

ONLINE Online mode
AIRPLANE Low Power mode i.e temporarily disabled RF
FACTORY_TEST Special mode for manufacturer use
OFFLINE Device has deactivated RF and partially shutdown
RESETTING Device is in process of power cycling
SHUTTING_DOWN Device is in process of shutting down
PERSISTENT_LOW_POWER Persists low power mode even on reset

5.3.2.20 enum telux::tel::EcbMode

Emergency callback mode

Enumerator

NORMAL Device is not in emergency callback mode(ECBM)

EMERGENCY Device is in emergency callback mode(ECBM)

5.3.2.21 enum telux::tel::SignalStrengthLevel

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

5.3.2.22 enum telux::tel::VoiceServiceState

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

5.3.2.23 enum telux::tel::VoiceServiceDenialCause

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 `getSipErrorCode` ()=0
- virtual int `getPhoneId` ()=0
- virtual bool `isMultiPartyCall` ()=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::I-
CommandResponseCallback > callback = nullptr) [pure virtual]`

Allows the client to answer the call. This is only applicable for CallState::INCOMING and CallState::WAITING calls. If a Waiting call is being answered and the existing call is Active, then existing call will move to Hold state. If the existing call is on Hold already, then it will remain on Hold. The waiting call state transition from Waiting to Active.

Parameters

<i>in</i>	<i>callback</i>	- optional callback pointer to get the response of answer request below are possible error codes for callback response • SUCCESS • RADIO_NOT_AVAILABLE • NO_MEMORY • MODEM_ERR • INTERNAL_ERR • INVALID_STATE • INVALID_CALL_ID • INVALID_ARGUMENTS • OPERATION_NOT_ALLOWED • GENERIC_FAILURE

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::I-
CommandResponseCallback > callback = nullptr) [pure virtual]`

Puts the ongoing call on hold.

Parameters

in	<i>callback</i>	- optional callback pointer to get the response of hold request below are possible error codes for callback response • SUCCESS • RADIO_NOT_AVAILABLE • NO_MEMORY • MODEM_ERR • INTERNAL_ERR • INVALID_STATE • INVALID_CALL_ID • INVALID_ARGUMENTS • OPERATION_NOT_ALLOWED • GENERIC_FAILURE

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::l-
CommandResponseCallback > callback = nullptr) [pure virtual]`

Resumes this call from on-hold state to active state

Parameters

in	<i>callback</i>	- optional callback pointer to get the response of resume request below are possible error codes for callback response • SUCCESS • RADIO_NOT_AVAILABLE • NO_MEMORY • MODEM_ERR • INTERNAL_ERR • INVALID_STATE • INVALID_CALL_ID • INVALID_ARGUMENTS • OPERATION_NOT_ALLOWED • GENERIC_FAILURE

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::l-
CommandResponseCallback > callback = nullptr) [pure virtual]`

Reject the incoming/waiting call. Only applicable for CallState::INCOMING and CallState::WAITING calls.

in	<i>callback</i>	- optional callback pointer to get the response of reject request below are possible error codes for callback response • SUCCESS • RADIO_NOT_AVAILABLE • NO_MEMORY • MODEM_ERR • INTERNAL_ERR • INVALID_STATE • INVALID_CALL_ID • INVALID_ARGUMENTS • OPERATION_NOT_ALLOWED • GENERIC_FAILURE

Returns

Status of reject function i.e. success or suitable error code.

5.4.1.1.2.5 `virtual telux::common::Status telux::tel::lCall::reject (const std::string & rejectSMS,
std::shared_ptr< telux::common::lCommandResponseCallback > callback = nullptr)
[pure virtual]`

Reject the call and send an SMS to caller. Only applicable for CallState::INCOMING and CallState::WAITING calls.

Parameters

in	<i>rejectSMS</i>	SMS string used to send in response to a call rejection.
in	<i>callback</i>	- optional callback pointer to get the response of rejectwithSMS request below are possible error codes for callback response • SUCCESS • RADIO_NOT_AVAILABLE • NO_MEMORY • MODEM_ERR • INTERNAL_ERR • INVALID_STATE • INVALID_CALL_ID • INVALID_ARGUMENTS • OPERATION_NOT_ALLOWED • GENERIC_FAILURE

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::I-CommandResponseCallback > *callback* = nullptr) [pure virtual]**

Hangup the call if the call state is either active, hold, dialing, waiting or alerting.

Parameters

in	<i>callback</i>	- optional callback pointer to get the response of hangup request below are possible error codes for callback response • SUCCESS • RADIO_NOT_AVAILABLE • NO_MEMORY • MODEM_ERR • INTERNAL_ERR • INVALID_STATE • INVALID_CALL_ID • INVALID_ARGUMENTS • OPERATION_NOT_ALLOWED • GENERIC_FAILURE

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.

<i>in</i>	<i>tone</i>	- a single character with one of 12 values: 0-9, *, #.
<i>in</i>	<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

<i>in</i>	<i>callback</i>	- Optional callback pointer to get the result of stopDtmfTone function.

Returns

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

5.4.1.1.2.10 `virtual CallState telux::tel::ICall::getCallState ( ) [pure virtual]`

Get the current state of the call, such as ringing, in progress etc.

Returns

CallState - enumeration representing call State

5.4.1.1.2.11 `virtual int telux::tel::ICall::getCallIndex ( ) [pure virtual]`

Get the unique index of the call assigned by Telephony subsystem

Returns

Call Index

5.4.1.1.2.12 `virtual CallDirection telux::tel::ICall::getCallDirection ( ) [pure virtual]`

Get the direction of the call

Returns

CallDirection - enumeration representing call direction i.e. INCOMING/ OUTGOING

5.4.1.1.2.13 virtual std::string telux::tel::ICall::getRemotePartyNumber () [pure virtual]

Get the dailing number

Returns

Phone Number to which the call was dialed out Empty string in case of INCOMING call direction

5.4.1.1.2.14 virtual CallEndCause telux::tel::ICall::getCallEndCause () [pure virtual]

Get the cause of the termination of the call.

Returns

Enum representing call end cause.

5.4.1.1.2.15 virtual int telux::tel::ICall::getSipErrorCode () [pure virtual]

Get the SIP error code for the termination of the IMS call. Refer RFC3261 Section 21 for error description.

On platforms with access control enabled, the caller needs to have TELUX_TEL_CALL_INFO_READ permission to successfully invoke this API.

Returns

integer representing SIP error code.

Note

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

5.4.1.1.2.16 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.

5.4.1.1.2.17 virtual bool telux::tel::ICall::isMultiPartyCall () [pure virtual]

To check if call is in multi party call(conference) or not

Returns

True if call is in conference otherwise false.

Note

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

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 [onECallMsdtTransmissionStatus](#) (int phoneId, [telux::common::ErrorCode](#) errorCode)
- virtual void [onECallMsdtTransmissionStatus](#) (int phoneId, [telux::tel::ECallMsdtTransmissionStatus](#) msdTransmissionStatus)
- virtual void [OnMsdtUpdateRequest](#) (int phoneId)
- virtual void [onECallHlapTimerEvent](#) (int phoneId, [ECallHlapTimerEvents](#) timersStatus)
- virtual void [onEmergencyNetworkScanFail](#) (int phoneId)
- virtual void [onEcbmChange](#) ([telux::tel::EcbMode](#) mode)
- virtual [~ICallListener](#) ()

5.4.1.2.1 Constructors and Destructors

5.4.1.2.1.1 virtual [telux::tel::ICallListener::~~ICallListener](#) () [[virtual](#)]

5.4.1.2.2 Member Function Documentation

5.4.1.2.2.1 virtual void [telux::tel::ICallListener::onIncomingCall](#) (std::shared_ptr< [ICall](#) > *call*) [[virtual](#)]

This function is called when device receives an incoming/waiting call.

Parameters

in	<i>call</i>	- Pointer to ICall instance

5.4.1.2.2.2 virtual void [telux::tel::ICallListener::onCallInfoChange](#) (std::shared_ptr< [ICall](#) > *call*) [[virtual](#)]

This function is called when there is a change in call attributes

in	<i>call</i>	- Pointer to ICall instance

5.4.1.2.2.3 virtual void telux::tel::ICallListener::onECallMsdTransmissionStatus (int *phoneId*, telux::common::ErrorCode *errorCode*) [virtual]

This function is called when device completes MSD Transmission.

Parameters

in	<i>phoneId</i>	- Unique Id of phone on which MSD Transmission Status is being reported
in	<i>status</i>	- Indicates MSD Transmission status i.e. success or failure

Deprecated Use another [onECallMsdTransmissionStatus\(\)](#) API with argument ECallMsdTransmissionStatus

5.4.1.2.2.4 virtual void telux::tel::ICallListener::onECallMsdTransmissionStatus (int *phoneId*, telux::tel::ECallMsdTransmissionStatus *msdTransmissionStatus*) [virtual]

This function is called when MSD Transmission status is changed.

Parameters

in	<i>phoneId</i>	- Unique Id of phone on which MSD Transmission Status is being reported
in	<i>msdTransmission-Status</i>	- Indicates MSD Transmission status ECallMsdTransmissionStatus

5.4.1.2.2.5 virtual void telux::tel::ICallListener::OnMsdUpdateRequest (int *phoneId*) [virtual]

This function is called when an updated MSD is requested by the PSAP during a standard or NG eCall.

Client is expected to update the MSD using [telux::tel::ICallManager::updateECallMsd](#) upon receiving this notification. Modem updates its internal cache and responds to PSAP with the new MSD. In situations where the client fails to update the MSD, modem will time out and send the outdated MSD from its cache.

On platforms with access control enabled, the caller needs to have TELUX_TEL_ECALL_MGMT permission to receive this notification

Parameters

in	<i>phoneId</i>	- Unique ID of the phone on which this MSD update request is received

Note

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

5.4.1.2.2.6 **virtual void telux::tel::ICallListener::onECallHlapTimerEvent (int *phoneId*, ECallHlapTimerEvents *timersStatus*) [virtual]**

This function is called when the eCall High Level Application Protocol(HLAP) timers status is changed.

Parameters

in	<i>phoneId</i>	- Unique Id of phone on which Hlap timer status is being reported
in	<i>timersStatus</i>	- Indicates the Hlap timer event ECallHlapTimerEvents

Note

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

5.4.1.2.2.7 **virtual void telux::tel::ICallListener::onEmergencyNetworkScanFail (int *phoneId*) [virtual]**

This function is called whenever there is a scan failure after one round of network scan during origination of emergency call or at any time during the emergency call.

During origination of an ecall or in between an ongoing ecall, if the UE is in an area of no/poor coverage and loses service, the modem will perform network scan and try to register on any available network. If the scan completes successfully and the device finds a suitable cell, the ecall will be placed and the call state changes to the active state. If the network scan fails then this function will be invoked after one round of network scan.

Parameters

in	<i>phoneId</i>	- Unique Id of phone on which network scan failure reported.

Note

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

5.4.1.2.2.8 **virtual void telux::tel::ICallListener::onEcbmChange (telux::tel::EcbMode *mode*) [virtual]**

This function is called whenever emergency callback mode(ECBM) changes.

Parameters

in	<i>mode</i>	- Indicates the status of the ECBM. EcbMode

Note

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

5.4.1.3 class telux::tel::ICallManager

Call Manager is the primary interface for call related operations. Allows to conference calls, swap calls, make normal voice call and emergency call, send and update MSD pdu.

Public member functions

- virtual
telux::common::ServiceStatus getServiceStatus ()=0
- virtual telux::common::Status makeCall (int phoneId, const std::string &dialNumber, std::shared_ptr< IMakeCallCallback > callback=nullptr)=0
- virtual telux::common::Status makeECall (int phoneId, const ECallMsdata &eCallMsdata, int category, int variant, std::shared_ptr< IMakeCallCallback > callback=nullptr)=0
- virtual telux::common::Status makeECall (int phoneId, const std::string dialNumber, const ECallMsdata &eCallMsdata, int category, std::shared_ptr< IMakeCallCallback > callback=nullptr)=0
- virtual telux::common::Status makeECall (int phoneId, const std::vector< uint8_t > &msdPdu, int category, int variant, MakeCallCallback callback=nullptr)=0
- virtual telux::common::Status makeECall (int phoneId, const std::string dialNumber, const std::vector< uint8_t > &msdPdu, int category, MakeCallCallback callback=nullptr)=0
- virtual telux::common::Status makeECall (int phoneId, int category, int variant, MakeCallCallback callback=nullptr)=0
- virtual telux::common::Status makeECall (int phoneId, const std::string dialNumber, int category, MakeCallCallback callback=nullptr)=0
- virtual telux::common::Status updateECallMsdata (int phoneId, const ECallMsdata &eCallMsdata, std::shared_ptr< telux::common::ICommandResponseCallback > callback=nullptr)=0
- virtual telux::common::Status updateECallMsdata (int phoneId, const std::vector< uint8_t > &msdPdu, telux::common::ResponseCallback callback)=0
- virtual telux::common::Status requestECallHlapTimerStatus (int phoneId, ECallHlapTimerStatusCallback callback)=0
- virtual std::vector
< std::shared_ptr< ICall > > getInProgressCalls ()=0
- virtual telux::common::Status conference (std::shared_ptr< ICall > call1, std::shared_ptr< ICall > call2, std::shared_ptr< telux::common::ICommandResponseCallback > callback=nullptr)=0
- virtual telux::common::Status swap (std::shared_ptr< ICall > callToHold, std::shared_ptr< ICall > callToActivate, std::shared_ptr< telux::common::ICommandResponseCallback > callback=nullptr)=0
- virtual telux::common::Status hangupForegroundResumeBackground (int phoneId, common::ResponseCallback callback=nullptr)=0
- virtual telux::common::Status requestEcbm (int phoneId, EcbmStatusCallback callback)=0
- virtual telux::common::Status exitEcbm (int phoneId, common::ResponseCallback callback=nullptr)=0

- virtual `telux::common::Status encodeEuroNcapOptionalAdditionalData (telux::tel::ECallOptionalEuroNcapData optionalEuroNcapData, std::vector< uint8_t > &data)=0`
- virtual `telux::common::ErrorCode encodeECallMsd (telux::tel::ECallMsdData eCallMsdData, std::vector< uint8_t > &data)=0`
- virtual `telux::common::Status restartECallHlapTimer (int phoneId, EcallHlapTimerId timerId, int duration, common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status registerListener (std::shared_ptr< telux::tel::ICallListener > listener)=0`
- virtual `telux::common::Status removeListener (std::shared_ptr< telux::tel::ICallListener > listener)=0`
- virtual `~ICallManager ()`

5.4.1.3.1 Constructors and Destructors

5.4.1.3.1.1 virtual `telux::tel::ICallManager::~~ICallManager ( ) [virtual]`

5.4.1.3.2 Member Function Documentation

5.4.1.3.2.1 virtual `telux::common::ServiceStatus telux::tel::ICallManager::getServiceStatus ( ) [pure virtual]`

This status indicates whether the `ICallManager` object is in a usable state.

Returns

SERVICE_AVAILABLE - If CallManager is ready for service. SERVICE_UNAVAILABLE - If CallManager is temporarily unavailable. SERVICE_FAILED - If CallManager encountered an irrecoverable failure.

Note

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

5.4.1.3.2.2 virtual `telux::common::Status telux::tel::ICallManager::makeCall ( int phoneId, const std::string & dialNumber, std::shared_ptr< IMakeCallCallback > callback = nullptr ) [pure virtual]`

Initiate a voice call. This API can also be used for e911/e112 type of regular emergency call. This is not meant for an automotive eCall.

Parameters

in	<i>phoneId</i>	Represents phone corresponding to which on make call operation is performed
in	<i>dialNumber</i>	String representing the dialing number

<code>in</code>	<code>callback</code>	Optional callback pointer to get the response of makeCall request. Possible(not exhaustive) error codes for callback response • <code>telux::common::ErrorCode::SUCCESS</code> • <code>telux::common::ErrorCode::RADIO_NOT_AVAILABLE</code> • <code>telux::common::ErrorCode::DIAL_MODIFIED_TO_USSD</code> • <code>telux::common::ErrorCode::DIAL_MODIFIED_TO_SS</code> • <code>telux::common::ErrorCode::DIAL_MODIFIED_TO_DIAL</code> • <code>telux::common::ErrorCode::INVALID_ARGUMENTS</code> • <code>telux::common::ErrorCode::NO_MEMORY</code> • <code>telux::common::ErrorCode::INVALID_STATE</code> • <code>telux::common::ErrorCode::NO_RESOURCES</code> • <code>telux::common::ErrorCode::INTERNAL_ERR</code> • <code>telux::common::ErrorCode::FDN_CHECK_FAILURE</code> • <code>telux::common::ErrorCode::MODEM_ERR</code> • <code>telux::common::ErrorCode::NO_SUBSCRIPTION</code> • <code>telux::common::ErrorCode::NO_NETWORK_FOUND</code> • <code>telux::common::ErrorCode::INVALID_CALL_ID</code> • <code>telux::common::ErrorCode::DEVICE_IN_USE</code> • <code>telux::common::ErrorCode::MODE_NOT_SUPPORTED</code> • <code>telux::common::ErrorCode::ABORTED</code> • <code>telux::common::ErrorCode::GENERIC_FAILURE</code>

Returns

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

5.4.1.3.2.3 `virtual telux::common::Status telux::tel::ICallManager::makeECall ( int phoneId, const E-CallMsData & eCallMsData, int category, int variant, std::shared_ptr< IMakeCallCallback > callback = nullptr ) [pure virtual]`

Initiate an automotive eCall.

Parameters

in	<i>phoneId</i>	Represents phone corresponding to which make eCall operation is performed
in	<i>eCallMsddata</i>	The structure containing required fields to create eCall Minimum Set of Data (MSD)
in	<i>category</i>	ECallCategory
in	<i>variant</i>	ECallVariant
in	<i>callback</i>	Optional callback pointer to get the response of makeECall request. Possible(not exhaustive) error codes for callback response • telux::common::ErrorCode::SUCCESS • telux::common::ErrorCode::RADIO_NOT_AVAILABLE • telux::common::ErrorCode::NO_MEMORY • telux::common::ErrorCode::MODEM_ERR • telux::common::ErrorCode::INTERNAL_ERR • telux::common::ErrorCode::INVALID_STATE • telux::common::ErrorCode::INVALID_CALL_ID • telux::common::ErrorCode::INVALID_ARGUMENTS • telux::common::ErrorCode::OPERATION_NOT_ALLOWED • telux::common::ErrorCode::GENERIC_FAILURE

Returns

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

5.4.1.3.2.4 `virtual telux::common::Status telux::tel::ICallManager::makeECall ( int phoneId, const std::string dialNumber, const ECallMsddata & eCallMsddata, int category, std::shared_ptr< IMakeCallCallback > callback = nullptr ) [pure virtual]`

Initiate an automotive eCall to the specified phone number for TPS eCall. It will be treated like a regular voice call by the UE and the network.

Parameters

in	<i>phoneId</i>	Represents phone corresponding to which make eCall operation is performed
in	<i>dialNumber</i>	String representing the dialing number
in	<i>eCallMsdata</i>	The structure containing required fields to create eCall Minimum Set of Data (MSD)
in	<i>category</i>	ECallCategory
in	<i>callback</i>	Optional callback pointer to get the response of makeECall request. Possible(not exhaustive) error codes for callback response • telux::common::ErrorCode::SUCCESS • telux::common::ErrorCode::RADIO_NOT_AVAILABLE • telux::common::ErrorCode::NO_MEMORY • telux::common::ErrorCode::MODEM_ERR • telux::common::ErrorCode::INTERNAL_ERR • telux::common::ErrorCode::INVALID_STATE • telux::common::ErrorCode::INVALID_CALL_ID • telux::common::ErrorCode::INVALID_ARGUMENTS • telux::common::ErrorCode::OPERATION_NOT_ALLOWED • telux::common::ErrorCode::GENERIC_FAILURE

Returns

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

Note

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

```
5.4.1.3.2.5 virtual telux::common::Status telux::tel::ICallManager::makeECall ( int phoneId, const
std::vector< uint8_t > & msdPdu, int category, int variant, MakeCallCallback callback =
nullptr ) [pure virtual]
```

Initiate an automotive eCall with raw MSD pdu.

Parameters

in	<i>phoneId</i>	Represents phone corresponding to which on make eCall operation is performed
in	<i>msdPdu</i>	Encoded MSD(Minimum Set of Data) PDU as per spec EN 15722 2015 or GOST R 54620-2011/33464-2015
in	<i>category</i>	ECallCategory
in	<i>variant</i>	ECallVariant

in	<i>callback</i>	Callback function to get the response of makeECall request. Possible(not exhaustive) error codes for callback response • telux::common::ErrorCode::SUCCESS • telux::common::ErrorCode::RADIO_NOT_AVAILABLE • telux::common::ErrorCode::NO_MEMORY • telux::common::ErrorCode::MODEM_ERR • telux::common::ErrorCode::INTERNAL_ERR • telux::common::ErrorCode::INVALID_STATE • telux::common::ErrorCode::INVALID_CALL_ID • telux::common::ErrorCode::INVALID_ARGUMENTS • telux::common::ErrorCode::OPERATION_NOT_ALLOWED • telux::common::ErrorCode::GENERIC_FAILURE

Returns

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

5.4.1.3.2.6 `virtual telux::common::Status telux::tel::lCallManager::makeECall (int phoneId,
const std::string dialNumber, const std::vector< uint8_t > & msdPdu, int category,
MakeCallCallback callback = nullptr) [pure virtual]`

Initiate an automotive eCall with raw MSD pdu, to the specified phone number for TPS eCall. It will be treated like a regular voice call by the UE and the network.

Parameters

in	<i>phoneId</i>	Represents phone corresponding to which on make eCall operation is performed
in	<i>dialNumber</i>	String representing the dialing number
in	<i>msdPdu</i>	Encoded MSD(Minimum Set of Data) PDU as per spec EN 15722 2015 or GOST R 54620-2011/33464-2015
in	<i>category</i>	ECallCategory
in	<i>callback</i>	Callback function to get the response of makeECall request. Possible(not exhaustive) error codes for callback response • telux::common::ErrorCode::SUCCESS • telux::common::ErrorCode::RADIO_NOT_AVAILABLE • telux::common::ErrorCode::NO_MEMORY • telux::common::ErrorCode::MODEM_ERR • telux::common::ErrorCode::INTERNAL_ERR • telux::common::ErrorCode::INVALID_STATE • telux::common::ErrorCode::INVALID_CALL_ID • telux::common::ErrorCode::INVALID_ARGUMENTS • telux::common::ErrorCode::OPERATION_NOT_ALLOWED • telux::common::ErrorCode::GENERIC_FAILURE

Returns

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

Note

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

5.4.1.3.2.7 `virtual telux::common::Status telux::tel::ICallManager::makeECall ( int phoneId, int category, int variant, MakeCallCallback callback = nullptr ) [pure virtual]`

Initiate an automotive eCall without transmitting Minimum Set of Data (MSD) at call connect.

Parameters

in	<i>phoneId</i>	Represents phone corresponding to which make eCall operation is performed
in	<i>category</i>	ECallCategory
in	<i>variant</i>	ECallVariant
in	<i>callback</i>	Optional callback function to get the response of makeECall request. Possible(not exhaustive) error codes for callback response • <code>telux::common::ErrorCode::SUCCESS</code> • <code>telux::common::ErrorCode::RADIO_NOT_AVAILABLE</code> • <code>telux::common::ErrorCode::NO_MEMORY</code> • <code>telux::common::ErrorCode::MODEM_ERR</code> • <code>telux::common::ErrorCode::INTERNAL_ERR</code> • <code>telux::common::ErrorCode::INVALID_STATE</code> • <code>telux::common::ErrorCode::INVALID_CALL_ID</code> • <code>telux::common::ErrorCode::INVALID_ARGUMENTS</code> • <code>telux::common::ErrorCode::OPERATION_NOT_ALLOWED</code> • <code>telux::common::ErrorCode::GENERIC_FAILURE</code>

Returns

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

5.4.1.3.2.8 `virtual telux::common::Status telux::tel::ICallManager::makeECall ( int phoneId, const std::string dialNumber, int category, MakeCallCallback callback = nullptr ) [pure virtual]`

Initiate an automotive eCall to the specified phone number for TPS eCall, without transmitting Minimum Set of Data(MSD) at call connect. It will be treated like a regular voice call by the UE and the network.

in	<i>phoneId</i>	Represents phone corresponding to which make eCall operation is performed
in	<i>dialNumber</i>	String representing the dialing number
in	<i>category</i>	ECallCategory
in	<i>callback</i>	Optional callback function to get the response of makeECall request. Possible(not exhaustive) error codes for callback response • telux::common::ErrorCode::SUCCESS • telux::common::ErrorCode::RADIO_NOT_AVAILABLE • telux::common::ErrorCode::NO_MEMORY • telux::common::ErrorCode::MODEM_ERR • telux::common::ErrorCode::INTERNAL_ERR • telux::common::ErrorCode::INVALID_STATE • telux::common::ErrorCode::INVALID_CALL_ID • telux::common::ErrorCode::INVALID_ARGUMENTS • telux::common::ErrorCode::OPERATION_NOT_ALLOWED • telux::common::ErrorCode::GENERIC_FAILURE

Returns

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

5.4.1.3.2.9 `virtual telux::common::Status telux::tel::ICallManager::updateECallMsdata ( int phoneId, const ECallMsdata & eCallMsdata, std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]`

Update the eCall MSD in modem to be sent to Public Safety Answering [Point](#) (PSAP) when requested.

Parameters

in	<i>phoneId</i>	Represents phone corresponding to which updateECallMsdata operation is performed
in	<i>eCallMsdata</i>	The data structure represents the Minimum Set of Data (MSD)
in	<i>callback</i>	Optional callback pointer to get the response of updateECallMsdata.

Returns

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

5.4.1.3.2.10 `virtual telux::common::Status telux::tel::ICallManager::updateECallMsdata ( int phoneId, const std::vector< uint8_t > & msdPdu, telux::common::ResponseCallback callback ) [pure virtual]`

Update the eCall MSD in modem to be sent to Public Safety Answering [Point](#) (PSAP) when requested.

in	<i>phoneId</i>	Represents phone corresponding to which updateECallMsd operation is performed
in	<i>msdPdu</i>	Encoded MSD(Minimum Set of Data) PDU as per spec EN 15722 2015 or GOST R 54620-2011/33464-2015
in	<i>callback</i>	Callback function to get the response of updateECallMsd.

Returns

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

5.4.1.3.2.11 **virtual telux::common::Status telux::tel::ICallManager::requestECallHlapTimerStatus (int *phoneId*, ECallHlapTimerStatusCallback *callback*) [pure virtual]**

Request for status of eCall High Level Application Protocol(HLAP) timers that are maintained by the UE state machine. This does not retrieve status of timers maintained by the PSAP. The provided timers are as per EN 16062:2015 standard.

Parameters

in	<i>phoneId</i>	Represents phone corresponding on which requestECallHlapTimerStatus operation is performed
in	<i>callback</i>	Callback function to get the response of requestECallHlapTimerStatus

Returns

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

Note

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

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

Get in-progress calls.

Returns

List of active calls.

5.4.1.3.2.13 **virtual telux::common::Status telux::tel::ICallManager::conference (std::shared_ptr< ICall > *call1*, std::shared_ptr< ICall > *call2*, std::shared_ptr< telux::common::ICommand-ResponseCallback > *callback* = nullptr) [pure virtual]**

Merge two calls in a conference.

in	<i>call1</i>	Call object to conference.
in	<i>call2</i>	Call object to conference.
in	<i>callback</i>	Optional callback pointer to get the result of conference function

Returns

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

5.4.1.3.2.14 `virtual telux::common::Status telux::tel::ICallManager::swap ( std::shared_ptr< ICall > callToHold, std::shared_ptr< ICall > callToActivate, std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]`

Swap calls to make one active and put the another on hold.

Parameters

in	<i>callToHold</i>	Active call object to swap to hold state.
in	<i>callToActivate</i>	Hold call object to swap to active state.
in	<i>callback</i>	Optional callback pointer to get the result of swap function

Returns

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

5.4.1.3.2.15 `virtual telux::common::Status telux::tel::ICallManager::hangupForegroundResume-Background ( int phoneld, common::ResponseCallback callback = nullptr ) [pure virtual]`

Hangup all the foreground call(s) if any and accept the background call as the active call. The foreground call here could be active call, incoming call or multiple active calls in case of conference and background call could be held call or waiting call.

If a call(s) is active, the active call(s) will be terminated or if a call is waiting, the waiting call will be accepted and becomes active. Otherwise, if a held call is present, the held call becomes active. In case of hold and waiting calls, the hold call will still be on hold and waiting call will be accepted. In case of hold, active and waiting scenario, the hold call will still be on hold, active call will be ended and waiting call will be accepted.

Parameters

in	<i>callback</i>	- optional callback pointer to get the response of hangup request below are possible error codes for callback response • SUCCESS • RADIO_NOT_AVAILABLE • NO_MEMORY • MODEM_ERR • INTERNAL_ERR • INVALID_STATE • INVALID_CALL_ID • INVALID_ARGUMENTS • OPERATION_NOT_ALLOWED • GENERIC_FAILURE

Returns

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

Note

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

5.4.1.3.2.16 `virtual telux::common::Status telux::tel::ICallManager::requestEcbm ( int phoneId, EcbmStatusCallback callback ) [pure virtual]`

Request for emergency callback mode

Parameters

in	<i>phoneId</i>	Represents the phone corresponding to which the emergency callback mode(ECBM) status is requested. param [in] callback Callback pointer to get the result of ECBM status request

Returns

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

Note

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

5.4.1.3.2.17 `virtual telux::common::Status telux::tel::ICallManager::exitEcbm (int phoneId,
common::ResponseCallback callback = nullptr) [pure virtual]`

Exit emergency callback mode.

in	<i>phoneId</i>	Represents the phone corresponding to which the emergency callback mode(ECBM) exit is requested. param [in] callback Optional callback pointer to get the result of exit ECBM request

Returns

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

Note

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

5.4.1.3.2.18 `virtual telux::common::Status telux::tel::ICallManager::encodeEuroNcapOptional-
AdditionalData (telux::tel::ECallOptionalEuroNcapData optionalEuroNcapData,
std::vector< uint8_t > & data) [pure virtual]`

Gets encoded bytes of optional additional data content as per the Euro NCAP Technical Bulletin TB 040. Client needs to pass this vector of bytes to the data field of the [ECallOptionalPdu](#).
[telux::tel::ECallOptionalPdu::data](#)

Parameters

in	<i>optionalEuroNcap-Data</i>	ECall optional additional data as per Euro NCAP Technical Bulletin TB 040. telux::tel::ECallOptionalEuroNcapData
out	<i>data</i>	Encoded optional additional data.

Returns

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

Note

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

5.4.1.3.2.19 `virtual telux::common::ErrorCode telux::tel::ICallManager::encodeECallMsd (
telux::tel::ECallMsdData eCallMsdData, std::vector< uint8_t > & data) [pure
virtual]`

Gets encoded bytes of eCall MSD according to EN 15722:2015 (MSD version 2) and EN 15722:2020 (MSD version 3).

in	<i>eCallMsdata</i>	eCall MSD data. telux::tel::ECallMsdata
out	<i>data</i>	Encoded bytes of eCall MSD.

Returns

error code for encodeECallMsdata i.e. success or suitable error code. Below are possible error codes.

- `telux::common::ErrorCode::SUCCESS`
- `telux::common::ErrorCode::INVALID_ARGUMENTS`
- `telux::common::ErrorCode::GENERIC_FAILURE`

Note

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

5.4.1.3.2.20 `virtual telux::common::Status telux::tel::ICallManager::restartECallHlapTimer ( int phoneld, EcallHlapTimerId timerId, int duration, common::ResponseCallback callback = nullptr ) [pure virtual]`

Restart T9 and T10 eCall High Level Application Protocol (HLAP) timers with residual timer duration. Application is expected to maintain residual timer information and resume the timers during events like modem reset or transition of device operating mode from low power mode to online.

Notes:

1. Application must restart timer according to eCall operating mode of device. T10 eCall HLAP timer must be restarted only when eCall operating mode is `telux::tel::ECallMode::ECALL_ONLY`.
2. Application must validate the residual timer value before calling the API to prevent invalid data being processed.
3. T9 eCall HLAP timer cannot be restarted after transition of device operating mode from low power mode to online.

On platforms with access control enabled, caller needs to have `TELUX_TEL_ECALL_MGMT` permission to invoke this API successfully.

Parameters

in	<i>timerId</i>	- Represents the timer which is required to be restarted by application. telux::tel::EcallHlapTimerId .
in	<i>duration</i>	- Remaining time duration in seconds for the timer to run.
in	<i>callback</i>	- Callback function to get the response of the restartECallHlapTimer request.

Returns

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

Note

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

5.4.1.3.2.21 `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 ICallListener object which receives event corresponding to phone

Returns

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

5.4.1.3.2.22 `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

Defines type of call like incoming, outgoing and none.

Enumerator

INCOMING
OUTGOING
NONE

5.4.2.2 enum telux::tel::CallState

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

Reason for the recently terminated call (either normally ended or failed)

Enumerator

UNOBTAINABLE_NUMBER Unassigned(unallocated) number
NO_ROUTE_TO_DESTINATION No route to destination

CHANNEL_UNACCEPTABLE Channel unacceptable
OPERATOR_DETERMINED_BARRING Operator determined barring
NORMAL Normal call barring
BUSY User busy
NO_USER_RESPONDING No user responding
NO_ANSWER_FROM_USER User alerting, no answer
NOT_REACHABLE Not reachable
CALL_REJECTED Call rejected
NUMBER_CHANGED Number changed
PREEMPTION Pre-emption
DESTINATION_OUT_OF_ORDER Destination out of order
INVALID_NUMBER_FORMAT Invalid number format (incomplete number)
FACILITY_REJECTED Facility rejected
RESP_TO_STATUS_ENQUIRY Response to STATUS ENQUIRY
NORMAL_UNSPECIFIED Normal, unspecified
CONGESTION No circuit/channel available
NETWORK_OUT_OF_ORDER Network out of order
TEMPORARY_FAILURE Temporary failure
SWITCHING_EQUIPMENT_CONGESTION Switching equipment congestion
ACCESS_INFORMATION_DISCARDED Access information discarded
REQUESTED_CIRCUIT_OR_CHANNEL_NOT_AVAILABLE Requested circuit/channel not available
RESOURCES_UNAVAILABLE_OR_UNSPECIFIED Resource unavailable, unspecified
QOS_UNAVAILABLE Quality of service unavailable
REQUESTED_FACILITY_NOT_SUBSCRIBED Requested facility not subscribed
INCOMING_CALLS_BARRED_WITHIN_CUG Incoming calls barred within the CUG
BEARER_CAPABILITY_NOT_AUTHORIZED Bearer capability not authorized
BEARER_CAPABILITY_UNAVAILABLE Bearer capability not presently available
SERVICE_OPTION_NOT_AVAILABLE Service or option not available, unspecified
BEARER_SERVICE_NOT_IMPLEMENTED Bearer service not implemented
ACM_LIMIT_EXCEEDED ACM equal to or greater than ACMmax
REQUESTED_FACILITY_NOT_IMPLEMENTED Requested facility not implemented
ONLY_DIGITAL_INFORMATION_BEARER_AVAILABLE Only restricted digital information bearer capability is available
SERVICE_OR_OPTION_NOT_IMPLEMENTED Service or option not implemented, unspecified
INVALID_TRANSACTION_IDENTIFIER Invalid transaction identifier value
USER_NOT_MEMBER_OF_CUG User not member of CUG
INCOMPATIBLE_DESTINATION Incompatible destination
INVALID_TRANSIT_NW_SELECTION Invalid transit network selection
SEMANTICALLY_INCORRECT_MESSAGE Semantically incorrect message
INVALID_MANDATORY_INFORMATION Invalid mandatory information
MESSAGE_TYPE_NON_IMPLEMENTED Message type non-existent or not implemented
MESSAGE_TYPE_NOT_COMPATIBLE_WITH_PROTOCOL_STATE Message type not compatible with protocol state
INFORMATION_ELEMENT_NON_EXISTENT Information element non-existent or not implemented
CONDITIONAL_IE_ERROR Conditional IE error
MESSAGE_NOT_COMPATIBLE_WITH_PROTOCOL_STATE Message not compatible with protocol state
RECOVERY_ON_TIMER_EXPIRED Recovery on timer expiry
PROTOCOL_ERROR_UNSPECIFIED Protocol error, unspecified
INTERWORKING_UNSPECIFIED Interworking, unspecified

CALL_BARRED Call barred
FDN_BLOCKED FDN blocked
IMSI_UNKNOWN_IN_VLR Incorrect IMSI
IMEI_NOT_ACCEPTED IMEI not accepted
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
RADIO_OFF Radio is OFF
OUT_OF_SERVICE No cellular coverage
NO_VALID_SIM No valid SIM is present
RADIO_INTERNAL_ERROR Internal error at Modem
NETWORK_RESP_TIMEOUT No response from network
NETWORK_REJECT Explicit network reject
RADIO_ACCESS_FAILURE RRC connection failure. Eg.RACH
RADIO_LINK_FAILURE Radio Link Failure
RADIO_LINK_LOST Radio link lost due to poor coverage
RADIO_UPLINK_FAILURE Radio uplink failure
RADIO_SETUP_FAILURE RRC connection setup failure
RADIO_RELEASE_NORMAL RRC connection release, normal
RADIO_RELEASE_ABNORMAL RRC connection release, abnormal
ACCESS_CLASS_BLOCKED Access class barring
NETWORK_DETACH Explicit network detach
EMERGENCY_TEMP_FAILURE Emergency redial temporary failure
EMERGENCY_PERM_FAILURE Emergency redial permanent failure
HO_NOT_FEASIBLE Hand over not feasible
USER_BUSY User busy
USER_REJECT User reject
LOW_BATTERY Battery is low
BLACKLISTED_CALL_ID Blacklisted caller id
CS_RETRY_REQUIRED Retry CS call, VoLTE service can't be provided by the network or remote
end
CDMA_LOCKED_UNTIL_POWER_CYCLE MS is locked until next power cycle
CDMA_DROP Drop call
CDMA_INTERCEPT INTERCEPT order received, MS state idle entered
CDMA_REORDER MS has been redirected, call is cancelled
CDMA_SO_REJECT Service option rejection
CDMA_RETRY_ORDER Requested service is rejected, retry delay is set
CDMA_ACCESS_FAILURE Unable to obtain access to the CDMA system
CDMA_PREEMPTED Not a preempted call
CDMA_NOT_EMERGENCY For non-emergency number dialed during emergency callback mode
CDMA_ACCESS_BLOCKED CDMA network access probes blocked
NETWORK_UNAVAILABLE Network unavailable
FEATURE_UNAVAILABLE Feature unavailable
SIP_ERROR SIP internal error
MISC SIP miscellaneous error
ANSWERED_ELSEWHERE MT call has ended due to a release from the network because the call was
answered elsewhere
PULL_OUT_OF_SYNC MultiEndpoint - call pull request has failed
CAUSE_CALL_PULLED MultiEndpoint - call has been pulled from primary to secondary
SIP_REDIRECTED Request is redirected

SIP_BAD_REQUEST Bad request
SIP_FORBIDDEN Forbidden
SIP_NOT_FOUND Remote URI not found
SIP_NOT_SUPPORTED Not supported
SIP_REQUEST_TIMEOUT Request timed out
SIP_TEMPORARILY_UNAVAILABLE Temporarily unavailable
SIP_BAD_ADDRESS Address incomplete
SIP_BUSY User busy
SIP_REQUEST_CANCELLED Request(call) rejected
SIP_NOT_ACCEPTABLE Not acceptable
SIP_NOT_REACHABLE Not reachable
SIP_SERVER_INTERNAL_ERROR Server internal error
SIP_SERVER_NOT_IMPLEMENTED Server not implemented
SIP_SERVER_BAD_GATEWAY Server bad gateway
SIP_SERVICE_UNAVAILABLE Service unavailable
SIP_SERVER_TIMEOUT Server time out
SIP_SERVER_VERSION_UNSUPPORTED Server version not supported
SIP_SERVER_MESSAGE_TOOLARGE Server message is too large
SIP_SERVER_PRECONDITION_FAILURE Server pre-condition failure
SIP_USER_REJECTED User(call) rejected
SIP_GLOBAL_ERROR Global error
MEDIA_INIT_FAILED Media resource initialization failure
MEDIA_NO_DATA RTP timeout(no audio/video traffic in the session)
MEDIA_NOT_ACCEPTABLE Media is not supported
MEDIA_UNSPECIFIED_ERROR Media unspecified error
HOLD_RESUME_FAILED Resume failed for hold call
HOLD_RESUME_CANCELED Resume cancelled for hold call
HOLD_REINVITE_COLLISION Reinvite collision for hold call
SIP_ALTERNATE_EMERGENCY_CALL Alternate emergency call
NO_CSFB_IN_CS_ROAM CS fallback in roaming not allowed
SRV_NOT_REGISTERED Service not registered
CALL_TYPE_NOT_ALLOWED Call type not allowed
EMRG_CALL_ONGOING Emergency call is in progress
CALL_SETUP_ONGOING Call setup is in progress
MAX_CALL_LIMIT_REACHED Maximum call limit reached
UNSUPPORTED_SIP_HDRS Unsupported sip header
CALL_TRANSFER_ONGOING Call transfer is in progress
PRACK_TIMEOUT Memory allocation failure or RTP open failure
QOS_FAILURE Call failed due to lack of dedicated bearer
ONGOING_HANDOVER Call rejected due to pending handover
VT_WITH_TTY_NOT_ALLOWED TTY and VT are not supported together
CALL_UPGRADE_ONGOING Upgrade request is in progress
CONFERENCE_WITH_TTY_NOT_ALLOWED Call from conference server received when TTY is ON
CALL_CONFERENCE_ONGOING Conference call is ongoing
VT_WITH_AVPF_NOT_ALLOWED VT call with AVPF
ENCRYPTION_CALL_ONGOING Encrypted call could not coexist with other calls
CALL_ONGOING_CW_DISABLED Call waiting disabled during incoming call
CALL_ON_OTHER_SUB Active call on other subscription
ONE_X_COLLISION CDMA collision
UI_NOT_READY UI is not ready during the incoming call

CS_CALL_ONGOING CS call ongoing when incoming call is received

REJECTED_ELSEWHERE One of the devices (interconnected endpoints) rejected the call

USER_REJECTED_SESSION_MODIFICATION Upgrade/downgrade rejected

USER_CANCELLED_SESSION_MODIFICATION Upgrade/downgrade cancelled

SESSION_MODIFICATION_FAILED Upgrade/downgrade failed

SIP_UNAUTHORIZED Unauthorized

SIP_PAYMENT_REQUIRED Payment required

SIP_METHOD_NOT_ALLOWED Method requested in the address line was not allowed for the address identified by the request-URI

SIP_PROXY_AUTHENTICATION_REQUIRED Client must first authenticate with a proxy

SIP_REQUEST_ENTITY_TOO_LARGE Request entity body is larger than what the server is willing to process

SIP_REQUEST_URI_TOO_LARGE Server is refusing to service because the request-URI is longer than the server willing to interpret

SIP_EXTENSION_REQUIRED Extension to process a request is not listed in the supported header field in the request

SIP_INTERVAL_TOO_BRIEF Expiration time of the resource refreshed by the request is too short

SIP_CALL_OR_TRANS_DOES_NOT_EXIST Request received by a UAS does not match any existing dialog or transaction

SIP_LOOP_DETECTED Server detected a loop

SIP_TOO_MANY_HOPS Request received has Max-Forwards header field at 0

SIP_AMBIGUOUS Requested URI was ambiguous

SIP_REQUEST_PENDING Request was received by a UAS that had a pending request within the same dialog

SIP_UNDECIPHERABLE Request has an encrypted MIME body for which the recipient does not possess an appropriate decryption key

RETRY_ON_IMS_WITHOUT_RTT Call should be tried on IMS with RTT disabled

MAX_PS_CALLS Maximum PS calls exceeded

SIP_MULTIPLE_CHOICES Multiple choices

SIP_MOVED_PERMANENTLY Moved permanently

SIP_MOVED_TEMPORARILY Moved temporarily

SIP_USE_PROXY Use proxy

SIP_ALTERNATE_SERVICE Alternate service

SIP_UNSUPPORTED_URI_SCHEME Unsupported URI scheme

SIP_REMOTE_UNSUPP_MEDIA_TYPE Unsupported media type

SIP_BAD_EXTENSION Bad extension

DSDA_CONCURRENT_CALL_NOT_POSSIBLE Concurrent call is not possible

EPSFB_FAILURE Call ended due to evolved packet system fallback (EPSFB) failure

TWAIT_EXPIRED Call ended due to twait timer expired

TCP_CONNECTION_REQ Call ended due to TCP connection

THERMAL_EMERGENCY Thermal emergency

ERROR_UNSPECIFIED 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
SmsEncoding	encoding	Data encoding type
int	numberOf-Segments	Number of segments
int	segmentSize	Max size of each segment
int	numberOf-CharsLeftIn-LastSegment	characters left in last segment

5.5.1.2 struct telux::tel::MessagePartInfo

Structure containing information about the part of multi-part SMS such as concatenated message reference number, number of segments and segment number. During concatenation this information along with originating address helps in associating each part of the multi-part message to the corresponding multi-part message.

Data fields

Type	Field	Description
uint16_t	refNumber	Concatenated message reference number as per 3GPP TS 23.040 9.2.3.24.1. For each part of multipart message this message reference will be the same
uint8_t	numberOf-Segments	Number of segments
uint8_t	segment-Number	Segment Number

5.5.1.3 class telux::tel::SmsMessage

Data structure represents an incoming SMS. This is applicable for single part message or part of the multipart message.

Public member functions

- [SmsMessage](#) (std::string text, std::string sender, std::string receiver, [SmsEncoding](#) encoding, std::string pdu, [PduBuffer](#) pduBuffer, std::shared_ptr< [MessagePartInfo](#) > info)
- const std::string & [getText](#) () const
- const std::string & [getSender](#) () const
- const std::string & [getReceiver](#) () const
- [SmsEncoding](#) [getEncoding](#) () const
- const std::string & [getPdu](#) () const
- [PduBuffer](#) [getRawPdu](#) () const
- std::shared_ptr< [MessagePartInfo](#) > [getMessagePartInfo](#) ()
- const std::string [toString](#) () const

5.5.1.3.1 Constructors and Destructors

5.5.1.3.1.1 `telux::tel::SmsMessage::SmsMessage ( std::string text, std::string sender, std::string receiver, SmsEncoding encoding, std::string pdu, PduBuffer pduBuffer, std::shared_ptr< MessagePartInfo > info )`

5.5.1.3.2 Member Function Documentation

5.5.1.3.2.1 `const std::string& telux::tel::SmsMessage::getText ( ) const`

Get the message text for the single part message or part of the multipart message.

Returns

String containing SMS message.

5.5.1.3.2.2 `const std::string& telux::tel::SmsMessage::getSender ( ) const`

Get the originating address (sender address).

Returns

String containing sender address.

5.5.1.3.2.3 `const std::string& telux::tel::SmsMessage::getReceiver ( ) const`

Get the destination address (receiver address).

Returns

String containing receiver address

5.5.1.3.2.4 SmsEncoding telux::tel::SmsMessage::getEncoding () const

Get encoding format used for the single part message or part of the multipart message.

Returns

SMS message encoding used.

5.5.1.3.2.5 const std::string& telux::tel::SmsMessage::getPdu () const

Get the raw PDU for the single part message or part of the multipart message.

Returns

String containing raw PDU content.

Deprecated Use API [SmsMessage::getRawPdu](#)

5.5.1.3.2.6 PduBuffer telux::tel::SmsMessage::getRawPdu () const

Get the raw PDU buffer for the single part message or part of the multipart message.

Returns

Buffer containing raw PDU content.

Note

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

5.5.1.3.2.7 std::shared_ptr<MessagePartInfo> telux::tel::SmsMessage::getMessagePartInfo ()

Applicable for multi-part SMS only. Get the information such as segment number, number of segments and concatenated reference number corresponding to the part of multi-part SMS.

Returns

If a message is single part SMS the method returns null otherwise returns message part information.

Note

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

5.5.1.3.2.8 `const std::string telux::tel::SmsMessage::toString ( ) const`

Get the text related informative representation of this object.

Returns

String containing informative string.

5.5.1.4 `class telux::tel::ISmsManager`

SmsManager class is the primary interface to manage SMS operations such as send and receive an SMS text and raw encoded PDU(s). This class handles single part and multi-part messages.

Public member functions

- virtual `telux::common::ServiceStatus getServiceStatus ()=0`
- virtual `telux::common::Status sendSms (const std::string &message, const std::string &receiverAddress, std::shared_ptr< telux::common::ICommandResponseCallback > callback=nullptr, std::shared_ptr< telux::common::ICommandResponseCallback > deliveryCallback=nullptr)=0`
- virtual `telux::common::Status sendSms (std::string message, std::string receiverAddress, bool deliveryReportNeeded=true, SmsResponseCb sentCallback=nullptr, std::string smscAddr="")=0`
- virtual `telux::common::Status sendRawSms (const std::vector< PduBuffer > rawPdus, SmsResponseCb sentCallback=nullptr)=0`
- virtual `telux::common::Status requestSmscAddress (std::shared_ptr< ISmscAddressCallback > callback=nullptr)=0`
- virtual `telux::common::Status setSmscAddress (const std::string &smcAddress, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `MessageAttributes calculateMessageAttributes (const std::string &message)=0`
- virtual `int getPhoneId ()=0`
- virtual `telux::common::Status registerListener (std::weak_ptr< ISmsListener > listener)=0`
- virtual `telux::common::Status removeListener (std::weak_ptr< ISmsListener > listener)=0`
- virtual `~ISmsManager ()`

5.5.1.4.1 Constructors and Destructors

5.5.1.4.1.1 `virtual telux::tel::ISmsManager::~ISmsManager ( ) [virtual]`

5.5.1.4.2 Member Function Documentation

5.5.1.4.2.1 `virtual telux::common::ServiceStatus telux::tel::ISmsManager::getServiceStatus ( ) [pure virtual]`

This status indicates whether the `ISmsManager` object is in a usable state.

Returns

[telux::common::ServiceStatus](#)

Note

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

5.5.1.4.2.2 `virtual telux::common::Status telux::tel::ISmsManager::sendSms (const std::string & message, const std::string & receiverAddress, std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr, std::shared_ptr< telux::common::ICommandResponseCallback > deliveryCallback = nullptr) [pure virtual]`

Send SMS to the destination address. When registered on IMS the SMS will be attempted over IMS. If sending SMS over IMS fails, an automatic retry would be attempted to send the message over CS. Only support UCS2 format, GSM 7 bit default alphabet and does not support National language shift tables.

Parameters

in	<i>message</i>	Message text to be sent
in	<i>receiverAddress</i>	Receiver or destination address
in	<i>sentCallback</i>	Optional callback pointer to get the response of send SMS request.
in	<i>deliveryCallback</i>	Optional callback pointer to get message delivery status

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

Returns

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

5.5.1.4.2.3 `virtual telux::common::Status telux::tel::ISmsManager::sendSms ( std::string message, std::string receiverAddress, bool deliveryReportNeeded = true, SmsResponseCb sentCallback = nullptr, std::string smscAddr = "" ) [pure virtual]`

Send single or multipart SMS to the destination address. When registered on IMS the SMS will be attempted over IMS. If sending SMS over IMS fails, an automatic retry would be attempted to send the message over CS. Only support UCS2 format, GSM 7 bit default alphabet and does not support National language shift tables.

Parameters

in	<i>message</i>	Message text to be send.
in	<i>receiverAddress</i>	Receiver or destination address
in	<i>deliveryReport-Needed</i>	Delivery status received in the listener API telux::tel::ISmsListener if deliveryReportNeeded is true. Provided recipient responds to SMSC before the validity period expires. If deliveryReportNeeded is false delivery report will not be received.
in	<i>sentCallback</i>	Optional callback pointer to get the sent response for single part or multi-part SMS.
in	<i>smcAddr</i>	SMS is sent to SMSC address. If SMSC address is empty then pre-configured SMSC address is used.

Returns

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

Note

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

5.5.1.4.2.4 `virtual telux::common::Status telux::tel::ISmsManager::sendRawSms ( const std::vector< PduBuffer > rawPdus, SmsResponseCb sentCallback = nullptr ) [pure virtual]`

Send an SMS that is provided as a raw encoded PDU(s). When registered on IMS the SMS will be attempted over IMS. If sending SMS over IMS fails, an automatic retry would be attempted to send the message over CS. If the SMS is a multi-part message, the API expects multiple PDU to be passed to it.

Parameters

in	<i>rawPdus</i>	Each element in the vector represents a part of a multipart message. For single part message the vector will have single element.
in	<i>sentCallback</i>	Optional callback to get the sent response for single part or multi-part SMS.

Returns

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

Note

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

5.5.1.4.2.5 `virtual telux::common::Status telux::tel::ISmsManager::requestSmscAddress ( std::shared_ptr< ISmscAddressCallback > callback = nullptr ) [pure virtual]`

Request for Short Messaging Service Center (SMSC) Address. Purpose of SMSC is to store, forward, convert and deliver Short Message Service (SMS) messages.

Parameters

in	<i>callback</i>	Optional callback pointer to get the response of Smsc address request

Returns

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

5.5.1.4.2.6 `virtual telux::common::Status telux::tel::ISmsManager::setSmscAddress ( const std::string & smcAddress, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Sets the Short Message Service Center(SMSC) address on the device.

This will change the SMSC address for all the SMS messages sent from any app.

Parameters

in	<i>smcAddress</i>	SMSC address
in	<i>callback</i>	Optional callback pointer to get the response of set SMSC address

Returns

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

5.5.1.4.2.7 `virtual MessageAttributes telux::tel::ISmsManager::calculateMessageAttributes ( const std::string & message ) [pure virtual]`

Calculate message attributes for the given message.

Parameters

in	<i>message</i>	Message to send

Returns

[MessageAttributes](#) structure containing encoding type, number of segments, max size of segment and characters left in last segment.

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

Get associated phone id for this SMSManager.

Returns

PhoneId.

5.5.1.4.2.9 virtual telux::common::Status telux::tel::ISmsManager::registerListener (std::weak_ptr< ISmsListener > *listener*) [pure virtual]

Register a listener for Sms events

Parameters

in	<i>listener</i>	Pointer to ISmsListener object which receives event corresponding to SMS

Returns

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

5.5.1.4.2.10 virtual telux::common::Status telux::tel::ISmsManager::removeListener (std::weak_ptr< ISmsListener > *listener*) [pure virtual]

Remove a previously added listener.

Parameters

in	<i>listener</i>	Pointer to ISmsListener object

Returns

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

5.5.1.5 class telux::tel::ISmsListener

A listener class receives notification for the incoming message(s) and delivery report for sent message(s).

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

Public member functions

- virtual void [onIncomingSms](#) (int phoneId, std::shared_ptr< [SmsMessage](#) > message)
- virtual void [onIncomingSms](#) (int phoneId, std::shared_ptr< std::vector< [SmsMessage](#) >> messages)
- virtual void [onDeliveryReport](#) (int phoneId, int msgRef, std::string receiverAddress, [telux::common::ErrorCode](#) error)

- virtual [~ISmsListener](#) ()

5.5.1.5.1 Constructors and Destructors

5.5.1.5.1.1 virtual `telux::tel::ISmsListener::~~ISmsListener ( )` [virtual]

5.5.1.5.2 Member Function Documentation

5.5.1.5.2.1 virtual void `telux::tel::ISmsListener::onIncomingSms ( int phoneId, std::shared_ptr< SmsMessage > message )` [virtual]

This function will be invoked when a single part message is received or when a part of a multi-part message is received.

Parameters

in	<i>phoneId</i>	Unique identifier per SIM slot. Phone on which the message is received.
in	<i>SmsMessage</i>	Pointer to SmsMessage object

5.5.1.5.2.2 virtual void `telux::tel::ISmsListener::onIncomingSms ( int phoneId, std::shared_ptr< std::vector< SmsMessage >> messages )` [virtual]

This function will be invoked when either a single part message is received, or when all the parts of a multipart message have been received.

Parameters

in	<i>phoneId</i>	Unique identifier per SIM slot. Phone on which the message is received.
in	<i>messages</i>	Pointer to list of SmsMessage received corresponding to single part or all parts of multipart message.

Note

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

5.5.1.5.2.3 virtual void `telux::tel::ISmsListener::onDeliveryReport ( int phoneId, int msgRef, std::string receiverAddress, telux::common::ErrorCode error )` [virtual]

This function will be invoked when either a delivery report for a single part message is received or when the delivery report for part of a multi-part message is received. In order to determine delivery of all parts of the multi-part message, the client application shall compare message reference received in the delivery indications with message references received in [telux::tel::SmsResponseCb](#).

in	<i>phoneId</i>	Unique identifier per SIM slot. Phone on which the message is received.
in	<i>msgRef</i>	Message reference number (as per spec 3GPP TS 23.040 9.2.2.3) for a single part message or part of multipart message.
in	<i>receiverAddress</i>	Receiver or destination address
in	<i>error</i>	telux::common::ErrorCode

Note

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

5.5.1.6 class telux::tel::ISmscAddressCallback

Interface for SMS callback object. Client needs to implement this interface to get single shot responses for send SMS.

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

Public member functions

- virtual void [smscAddressResponse](#) (const std::string &address, [telux::common::ErrorCode](#) error)=0

5.5.1.6.1 Member Function Documentation**5.5.1.6.1.1 virtual void telux::tel::ISmscAddressCallback::smscAddressResponse (const std::string &address, telux::common::ErrorCode error) [pure virtual]**

This function is called with the response to the Smsc address request.

Parameters

in	<i>address</i>	Smsc address
in	<i>error</i>	ErrorCode

5.5.2 Enumeration Type Documentation**5.5.2.1 enum telux::tel::SmsEncoding**

Specifies the encoding of the SMS message.

Enumerator

GSM7 GSM 7-bit default alphabet encoding
GSM8 GSM 8-bit data encoding
UCS2 UCS-2 encoding
UNKNOWN Unknown encoding

5.6 SIM Card Services

This section contains APIs related to Card Services.

5.6.1 Data Structure Documentation

5.6.1.1 class telux::tel::ICardApp

Represents a single card application.

Public member functions

- virtual [AppType](#) [getAppType](#) ()=0
- virtual [AppState](#) [getAppState](#) ()=0
- virtual std::string [getAppId](#) ()=0
- virtual [telux::common::Status](#) [changeCardPassword](#) ([CardLockType](#) lockType, std::string oldPwd, std::string newPwd, [PinOperationResponseCb](#) callback)=0
- virtual [telux::common::Status](#) [unlockCardByPuk](#) ([CardLockType](#) lockType, std::string puk, std::string newPin, [PinOperationResponseCb](#) callback)=0
- virtual [telux::common::Status](#) [unlockCardByPin](#) ([CardLockType](#) lockType, std::string pin, [PinOperationResponseCb](#) callback)=0
- virtual [telux::common::Status](#) [queryPin1LockState](#) ([QueryPin1LockResponseCb](#) callback)=0
- virtual [telux::common::Status](#) [queryFdnLockState](#) ([QueryFdnLockResponseCb](#) callback)=0
- virtual [telux::common::Status](#) [setCardLock](#) ([CardLockType](#) lockType, std::string password, bool isEnabled, [PinOperationResponseCb](#) callback)=0
- virtual [~ICardApp](#) ()

5.6.1.1.1 Constructors and Destructors

5.6.1.1.1.1 virtual telux::tel::ICardApp::~~ICardApp () [virtual]

5.6.1.1.2 Member Function Documentation

5.6.1.1.2.1 virtual AppType telux::tel::ICardApp::getAppType () [pure virtual]

Get Application type like SIM, USIM, RUIM, CSIM or ISIM.

Returns

[AppType](#).

5.6.1.1.2.2 virtual AppState telux::tel::ICardApp::getAppState () [pure virtual]

Get Application state like PIN1, PUK required and others.

Returns

AppState.

5.6.1.1.2.3 virtual std::string telux::tel::ICardApp::getAppld () [pure virtual]

Get application identifier.

Returns

Application Id.

5.6.1.1.2.4 virtual telux::common::Status telux::tel::ICardApp::changeCardPassword (CardLockType lockType, std::string oldPwd, std::string newPwd, PinOperationResponseCb callback) [pure virtual]

Change the password used in PIN1/PIN2 lock.

Parameters

in	<i>lockType</i>	CardLockType. Applicable lock types are PIN1 and PIN2.
in	<i>oldPwd</i>	Old password
in	<i>newPwd</i>	New password
in	<i>callback</i>	Callback function to get the response of change pin password.

5.6.1.1.2.5 virtual telux::common::Status telux::tel::ICardApp::unlockCardByPuk (CardLockType lockType, std::string puk, std::string newPin, PinOperationResponseCb callback) [pure virtual]

Unlock the Sim card for an app by entering PUK and new pin.

Parameters

in	<i>lockType</i>	CardLockType. Applicable lock types are PUK1 and PUK2
in	<i>puk</i>	PUK1/PUK2
in	<i>pin</i>	New PIN1/PIN2
in	<i>callback</i>	Callback function to get the response of unlock card lock.

5.6.1.1.2.6 virtual telux::common::Status telux::tel::ICardApp::unlockCardByPin (CardLockType lockType, std::string pin, PinOperationResponseCb callback) [pure virtual]

Unlock the Sim card for an app by entering PIN.

in	<i>lockType</i>	CardLockType. Applicable lock types are PIN1 and PIN2.
in	<i>pin</i>	New PIN1/PIN2
in	<i>callback</i>	Callback function to get the response of unlock card lock.

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

Query Pin1 lock state.

Parameters

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

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

Query FDN lock state.

Parameters

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

5.6.1.1.2.9 virtual telux::common::Status telux::tel::ICardApp::setCardLock (CardLockType *lockType*, std::string *password*, bool *isEnabled*, PinOperationResponseCb *callback*) [pure virtual]

Enable or disable FDN or Pin1 lock.

Parameters

in	<i>lockType</i>	CardLockType. Applicable lock type such as PIN1 and FDN
in	<i>password</i>	Password of PIN1 and FDN
in	<i>isEnabled</i>	If true then enable else disable.
in	<i>callback</i>	Callback function to get the response of set card lock.

5.6.1.2 struct telux::tel::lccResult

The APDU response with status for transmit APDU operation.

Public member functions

- const std::string [toString](#) () const

Data Fields

- int [sw1](#)
- int [sw2](#)

- std::string [payload](#)
- std::vector< int > [data](#)

5.6.1.2.1 Member Function Documentation

5.6.1.2.1.1 const std::string telux::tel::lccResult::toString () const

5.6.1.2.2 Field Documentation

5.6.1.2.2.1 int telux::tel::lccResult::sw1

Status word 1 for command processing status

5.6.1.2.2.2 int telux::tel::lccResult::sw2

Status word 2 for command processing qualifier

5.6.1.2.2.3 std::string telux::tel::lccResult::payload

response as a hex string

5.6.1.2.2.4 std::vector<int> telux::tel::lccResult::data

vector of raw data received as part of response to the card services request

5.6.1.3 class telux::tel::ICardManager

[ICardManager](#) provide APIs for slot count, retrieve slot ids, get card state and get card.

Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status](#) [getSlotCount](#) (int &count)=0
- virtual [telux::common::Status](#) [getSlotIds](#) (std::vector< int > &slotIds)=0
- virtual std::shared_ptr< [ICard](#) > [getCard](#) (int slotId=DEFAULT_SLOT_ID, [telux::common::Status](#) *status=nullptr)=0
- virtual [telux::common::Status](#) [registerListener](#) (std::shared_ptr< [ICardListener](#) > listener)=0
- virtual [telux::common::Status](#) [removeListener](#) (std::shared_ptr< [ICardListener](#) > listener)=0
- virtual [~ICardManager](#) ()

5.6.1.3.1 Constructors and Destructors

5.6.1.3.1.1 `virtual telux::tel::ICardManager::~ICardManager ( ) [virtual]`

5.6.1.3.2 Member Function Documentation

5.6.1.3.2.1 `virtual bool telux::tel::ICardManager::isSubsystemReady ( ) [pure virtual]`

Checks the status of telephony subsystems and returns the result.

Returns

If true then CardManager is ready for service.

5.6.1.3.2.2 `virtual std::future<bool> telux::tel::ICardManager::onSubsystemReady ( ) [pure virtual]`

Wait for telephony subsystem to be ready.

Returns

A future that caller can wait on to be notified when card manager is ready.

5.6.1.3.2.3 `virtual telux::common::Status telux::tel::ICardManager::getSlotCount ( int & count ) [pure virtual]`

Get SIM slot count.

Parameters

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 ICardListener 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 ICardListener object that needs to be removed.

Returns

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

5.6.1.4 class telux::tel::ICard

[ICard](#) represents currently inserted UICC or eUICC.

Public member functions

- virtual `telux::common::Status getState (CardState &cardState)=0`
- virtual `std::vector< std::shared_ptr< ICardApp > > getApplications (telux::common::Status *status=nullptr)=0`
- virtual `telux::common::Status openLogicalChannel (std::string applicationId, std::shared_ptr< ICardChannelCallback > callback=nullptr)=0`
- virtual `telux::common::Status closeLogicalChannel (int channelId, std::shared_ptr< telux::common::ICommandResponseCallback > callback=nullptr)=0`
- virtual `telux::common::Status transmitApuLogicalChannel (int channel, uint8_t cla, uint8_t instruction, uint8_t p1, uint8_t p2, uint8_t p3, std::vector< uint8_t > data, std::shared_ptr< ICardCommandCallback > callback=nullptr)=0`
- virtual `telux::common::Status transmitApuBasicChannel (uint8_t cla, uint8_t instruction, uint8_t p1, uint8_t p2, uint8_t p3, std::vector< uint8_t > data, std::shared_ptr< ICardCommandCallback > callback=nullptr)=0`
- virtual `telux::common::Status exchangeSimIO (uint16_t fileId, uint8_t command, uint8_t p1, uint8_t p2, uint8_t p3, std::string filePath, std::vector< uint8_t > data, std::string pin2, std::string aid, std::shared_ptr< ICardCommandCallback > callback=nullptr)=0`
- virtual `int getSlotId ()=0`
- virtual `telux::common::Status requestEid (EidResponseCallback callback)=0`
- virtual `~ICard ()`

5.6.1.4.1 Constructors and Destructors

5.6.1.4.1.1 virtual `telux::tel::ICard::~~ICard ( ) [virtual]`

Destructor for `ICard`

5.6.1.4.2 Member Function Documentation

5.6.1.4.2.1 virtual `telux::common::Status telux::tel::ICard::getState ( CardState & cardState ) [pure virtual]`

Get the card state for the slot id.

Parameters

out	<i>cardState</i>	<code>CardState</code> - state of the card.

Returns

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

5.6.1.4.2.2 `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 getApplications i.e. success or suitable status code.

Returns

List of card applications.

5.6.1.4.2.3 `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.

5.6.1.4.2.4 `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.

5.6.1.4.2.5 `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.

5.6.1.4.2.6 `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.

5.6.1.4.2.7 `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

5.6.1.4.2.8 `virtual int telux::tel::ICard::getSlotId ( ) [pure virtual]`

Get associated slot id for [ICard](#)

Returns

SlotId

5.6.1.4.2.9 `virtual telux::common::Status telux::tel::ICard::requestEid ( EidResponseCallback callback ) [pure virtual]`

Request eUICC identifier (EID) of eUICC card.

Parameters

in	<i>callback</i>	Callback function to get the result of request EID.

Returns

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

Dependencies Card should be eUICC capable

5.6.1.5 class telux::tel::ICardChannelCallback

Interface for Card callback object. Client needs to implement this interface to get single shot responses for commands like open logical channel and close logical channel.

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

Public member functions

- virtual void [onChannelResponse](#) (int channel, [IccResult](#) result, [telux::common::ErrorCode](#) error)=0

5.6.1.5.1 Member Function Documentation

5.6.1.5.1.1 virtual void telux::tel::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>	IccResult 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>	IccResult of transmit APDU command
in	<i>error</i>	ErrorCode of the request, Possible error codes are • SUCCESS • INTERNAL • NO_MEMORY • INVALID_ARG • MISSING_ARG

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	<i>slotId</i>	Slot identifier.

5.6.1.8 struct telux::tel::CardReaderStatus

Structure contains identity of card reader status

Data fields

Type	Field	Description
int	id	Card Reader ID
bool	isRemovable	Card reader is removable
bool	isPresent	Card reader is present
bool	isID1size	Card reader present is ID-1 size

Type	Field	Description
bool	isCardPresent	Card is present in reader
bool	isCardPowered-On	Card in reader is powered

5.6.1.9 class telux::tel::ISapCardManager

[ISapCardManager](#) provide APIs for SAP related operations.

Public member functions

- virtual bool [isReady](#) ()=0
- virtual std::future< bool > [onReady](#) ()=0
- virtual [telux::common::Status](#) [getState](#) ([SapState](#) &sapState)=0
- virtual [telux::common::Status](#) [requestSapState](#) ([SapStateResponseCallback](#) callback)=0
- virtual [telux::common::Status](#) [openConnection](#) ([SapCondition](#) sapCondition=[SapCondition::SAP_CONDITION_BLOCK_VOICE_OR_DATA](#), std::shared_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status](#) [closeConnection](#) (std::shared_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status](#) [requestAtr](#) (std::shared_ptr< [IAtrResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status](#) [transmitApu](#) (uint8_t cla, uint8_t instruction, uint8_t p1, uint8_t p2, uint8_t lc, std::vector< uint8_t > data, uint8_t le=0, std::shared_ptr< [ISapCardCommandCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status](#) [requestSimPowerOff](#) (std::shared_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status](#) [requestSimPowerOn](#) (std::shared_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status](#) [requestSimReset](#) (std::shared_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status](#) [requestCardReaderStatus](#) (std::shared_ptr< [ICardReaderCallback](#) > callback=nullptr)=0
- virtual int [getSlotId](#) ()=0
- virtual [telux::common::Status](#) [registerListener](#) (std::shared_ptr< [ISapCardListener](#) > listener)=0
- virtual [telux::common::Status](#) [removeListener](#) (std::shared_ptr< [ISapCardListener](#) > listener)=0
- virtual [~ISapCardManager](#) ()

5.6.1.9.1 Constructors and Destructors

5.6.1.9.1.1 `virtual telux::tel::ISapCardManager::~ISapCardManager ( ) [virtual]`

5.6.1.9.2 Member Function Documentation

5.6.1.9.2.1 `virtual bool telux::tel::ISapCardManager::isReady ( ) [pure virtual]`

Checks the status of SIM access profile(SAP) subsystem and returns the result.

Returns

If true then SapCardManager is ready for service.

5.6.1.9.2.2 `virtual std::future<bool> telux::tel::ISapCardManager::onReady ( ) [pure virtual]`

Wait for IM access profile(SAP) subsystem to be ready.

Returns

A future that caller can wait on to be notified when card manager is ready.

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

Get SIM access profile (SAP) client connection state.

Parameters

out	<i>sapState</i>	SapState of the SIM Card

Returns

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

Deprecated Use [requestSapState\(\)](#) API below to get SAP state

5.6.1.9.2.4 `virtual telux::common::Status telux::tel::ISapCardManager::requestSapState ( SapStateResponseCallback callback ) [pure virtual]`

Get SIM access profile(SAP) client connection state.

Parameters

out	<i>callback</i>	Callback function pointer to get the response of requestSapState.

Returns

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

5.6.1.9.2.5 `virtual telux::common::Status telux::tel::ISapCardManager::openConnection ( SapCondition sapCondition = SapCondition::SAP_CONDITION_BLOCK_VOICE_OR_DATA, std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]`

Establishes SIM access profile (SAP) client connection with SIM Card.

Parameters

in	<i>sapCondition</i>	Condition to enable sap connection.
in	<i>callback</i>	Optional callback to get the response of open sap connection request or possible error codes i.e. • SUCCESS • INTERNAL • NO_MEMORY • INVALID_ARG • MISSING_ARG

Returns

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

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

Releases a SAP connection to SIM Card.

Parameters

in	<i>callback</i>	Optional callback to get the response of close sap connection request or possible error codes i.e. • SUCCESS • INTERNAL • NO_MEMORY • INVALID_ARG • MISSING_ARG

Returns

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

5.6.1.9.2.7 `virtual telux::common::Status telux::tel::ISapCardManager::requestAtr ( std::shared_ptr< IAtrResponseCallback > callback = nullptr ) [pure virtual]`

Request for SAP Answer To Reset command.

Parameters

in	<i>callback</i>	Optional callback to get the response of requestAtr.

Returns

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

5.6.1.9.2.8 `virtual telux::common::Status telux::tel::ISapCardManager::transmitApdu ( uint8_t cla, uint8_t instruction, uint8_t p1, uint8_t p2, uint8_t lc, std::vector< uint8_t > data, uint8_t le = 0, std::shared_ptr< ISapCardCommandCallback > callback = nullptr ) [pure virtual]`

Send the Apdu on SAP mode.

Parameters

in	<i>cla</i>	Class of the APDU command.
in	<i>instruction</i>	Instruction of the APDU command.
in	<i>p1</i>	Instruction Parameter 1 value of the APDU command.
in	<i>p2</i>	Instruction Parameter 1 value of the APDU command.
in	<i>lc</i>	Number of bytes present in the data field of the APDU command. If 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.

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

Send the SAP SIM power off request.

Parameters

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

Returns

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

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

Send the SAP SIM power on request.

Parameters

in	callback	Optional callback to get the response for SIM power on.

Returns

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

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

Send the SAP SIM reset request.

Parameters

in	callback	Optional callback to get the response for SIM reset

Returns

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

```
5.6.1.9.2.12 virtual telux::common::Status telux::tel::ISapCardManager::requestCardReaderStatus (
    std::shared_ptr< ICardReaderCallback > callback = nullptr ) [pure virtual]
```

Send the SAP Card Reader Status request command.

Parameters

in	callback	Optional callback to get the response for card reader status

Returns

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

5.6.1.9.2.13 virtual int telux::tel::ISapCardManager::getSlotId () [pure virtual]

Get associated slot id for the SapCardManager.

Returns

SlotId

5.6.1.9.2.14 virtual telux::common::Status telux::tel::ISapCardManager::registerListener (std::shared_ptr< ISapCardListener > *listener*) [pure virtual]

Register a listener for SAP events.

Parameters

<i>in</i>	<i>listener</i>	Pointer to ISapCardListener object that processes the notification.

Returns

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

5.6.1.9.2.15 virtual telux::common::Status telux::tel::ISapCardManager::removeListener (std::shared_ptr< ISapCardListener > *listener*) [pure virtual]

Remove a previously added listener.

Parameters

<i>in</i>	<i>listener</i>	Pointer to ISapCardListener object that needs to be removed.

Returns

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

5.6.1.10 class telux::tel::IAtrResponseCallback**Public member functions**

- virtual void [atrResponse](#) (std::vector< int > responseAtr, [telux::common::ErrorCode](#) error)=0

5.6.1.10.1 Member Function Documentation**5.6.1.10.1.1 virtual void telux::tel::IAtrResponseCallback::atrResponse (std::vector< int > *responseAtr*, [telux::common::ErrorCode](#) *error*) [pure virtual]**

This function is called in response to requestAtr() request.

in	<i>responseAtr</i>	response ATR values
in	<i>error</i>	ErrorCode of the request possible error codes are • SUCCESS • INTERNAL • NO_MEMORY • INVALID_ARG • MISSING_ARG

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 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>	IccResult of transmit APDU command
in	<i>error</i>	ErrorCode of the request, possible error codes are • SUCCESS • INTERNAL • NO_MEMORY • INVALID_ARG • MISSING_ARG

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 virtual void [telux::tel::ICardReaderCallback::cardReaderResponse](#) ([CardReaderStatus](#) *cardReaderStatus*, [telux::common::ErrorCode](#) *error*) [pure virtual]

This function is called in response to requestCardReaderStatus() method.

in	<i>cardReaderStatus</i>	Structure contains the identity of the card reader
in	<i>error</i>	ErrorCode of the request

5.6.1.13 class telux::tel::ISapCardListener

Interface for SAP Listener object. Client needs to implement this interface to get access to SAP service notifications like service status change.

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

Public member functions

- virtual [~ISapCardListener](#) ()

5.6.1.13.1 Constructors and Destructors

5.6.1.13.1.1 virtual telux::tel::ISapCardListener::~~ISapCardListener () [virtual]

5.6.2 Enumeration Type Documentation

5.6.2.1 enum telux::tel::CardState

Defines all state of Card like absent, present etc

Enumerator

CARDSTATE_UNKNOWN Unknown card state
CARDSTATE_ABSENT Card is absent
CARDSTATE_PRESENT Card is present
CARDSTATE_ERROR Card is having error, either card is removed and not readable
CARDSTATE_RESTRICTED Card is present but not usable due to carrier restrictions.

5.6.2.2 enum telux::tel::CardError

Defines the reasons for error in CardState

Enumerator

UNKNOWN Unknown error
POWER_DOWN Power down
POLL_ERROR Poll error
NO_ATR_RECEIVED No ATR received
VOLT_MISMATCH Volt mismatch
PARITY_ERROR Parity error
POSSIBLY_REMOVED Unknown, possibly removed
TECHNICAL_PROBLEMS Card returned technical problems
NULL_BYTES Card returned NULL bytes
SAP_CONNECTED Terminal in SAP mode
CMD_TIMEOUT Command timeout error

5.6.2.3 enum telux::tel::CardLockType

Defines all types of card locks which uses in PIN management APIs

Enumerator

PIN1 Lock type is PIN1
PIN2 Lock type is PIN2
PUK1 Lock type is Pin Unblocking Key1
PUK2 Lock type is Pin Unblocking Key2
FDN Lock type is Fixed Dialing Number

5.6.2.4 enum telux::tel::AppType

Defines all type of UICC application such as SIM, RUIM, USIM, CSIM and ISIM.

Enumerator

APPTYPE_UNKNOWN Unknown application type
APPTYPE_SIM UICC application type is SIM
APPTYPE_USIM UICC application type is USIM
APPTYPE_RUIM UICC application type is RSIM
APPTYPE_CSIM UICC application type is CSIM
APPTYPE_ISIM UICC application type is ISIM

5.6.2.5 enum telux::tel::AppState

Defines all application states.

Enumerator

APPSTATE_UNKNOWN Unknown application state
APPSTATE_DETECTED application state detected
APPSTATE_PIN If PIN1 or UPin is required
APPSTATE_PUK If PUK1 or Puk for UPin is required
APPSTATE_SUBSCRIPTION_PERSO PersoSubstate should be look at when application state is assigned to this value
APPSTATE_READY application State is ready

5.6.2.6 enum telux::tel::SapState

Defines all SIM access profile (SAP) connection states.

Enumerator

SAP_STATE_NOT_ENABLED SAP connection not enabled
SAP_STATE_CONNECTING SAP State is connecting
SAP_STATE_CONNECTED_SUCCESSFULLY SAP connection is successful
SAP_STATE_CONNECTION_ERROR SAP connection error
SAP_STATE_DISCONNECTING SAP state is disconnecting
SAP_STATE_DISCONNECTED_SUCCESSFULLY SAP state disconnection is successful

5.6.2.7 enum telux::tel::SapCondition

Indicates type of connection required, default behavior is to block a SAP connection when a voice or data call is active.

Enumerator

SAP_CONDITION_BLOCK_VOICE_OR_DATA Block a SAP connection when a voice or data call is active (Default)
SAP_CONDITION_BLOCK_DATA Block a SAP connection when a data call is active
SAP_CONDITION_BLOCK_VOICE Block a SAP connection when a voice call is active
SAP_CONDITION_BLOCK_NONE Allow Sap connection in all cases

5.7 Location Services

This section contains APIs related to Location Services.

5.7.1 Data Structure Documentation

5.7.1.1 class telux::loc::IDgnssStatusListener

Listener class for getting RTCM injection event notification information.

Public member functions

- virtual void [onDgnssStatusUpdate](#) ([DgnssStatus](#) status)
- virtual [~IDgnssStatusListener](#) ()

5.7.1.1.1 Constructors and Destructors

5.7.1.1.1.1 virtual telux::loc::IDgnssStatusListener::~~IDgnssStatusListener () [virtual]

Destructor of IRTCMStatusListener

5.7.1.1.2 Member Function Documentation

5.7.1.1.2.1 virtual void telux::loc::IDgnssStatusListener::onDgnssStatusUpdate ([DgnssStatus](#) status) [virtual]

This function is called asynchronously to report RTCM injection status

Parameters

in	status	- the status enum.

5.7.1.2 class telux::loc::IDgnssManager

IRtcManager provides interface to inject RTCM data into modem, register event listener reported by cdfw(correction data framework).

Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status](#) [registerListener](#) (std::weak_ptr< [IDgnssStatusListener](#) > listener)=0
- virtual [telux::common::Status](#) [deRegisterListener](#) (void)=0
- virtual [telux::common::Status](#) [createSource](#) ([DgnssDataFormat](#) dataFormat)=0
- virtual [telux::common::Status](#) [releaseSource](#) (void)=0

- virtual [telux::common::Status injectCorrectionData](#) (const uint8_t *buffer, uint32_t bufferSize)=0
- virtual [~IDgnssManager](#) ()

5.7.1.2.1 Constructors and Destructors

5.7.1.2.1.1 virtual [telux::loc::IDgnssManager::~IDgnssManager](#) () [virtual]

Destructor of IRtcmManager

5.7.1.2.2 Member Function Documentation

5.7.1.2.2.1 virtual bool [telux::loc::IDgnssManager::isSubsystemReady](#) () [pure virtual]

Checks the status of location subsystems and returns the result.

Returns

True if location subsystem is ready for service otherwise false.

5.7.1.2.2.2 virtual [std::future<bool>](#) [telux::loc::IDgnssManager::onSubsystemReady](#) () [pure virtual]

Wait for location subsystem to be ready.

Returns

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

5.7.1.2.2.3 virtual [telux::common::Status](#) [telux::loc::IDgnssManager::registerListener](#) ([std::weak_ptr<IDgnssStatusListener>](#) *listener*) [pure virtual]

Register a listener for Dgnss injection status update.

Parameters

in	<i>listener</i>	- Pointer of IDgnssStatusListener object that processes the notification.

Returns

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

5.7.1.2.2.4 `virtual telux::common::Status telux::loc::IDgnssManager::deRegisterListener ( void )`
`[pure virtual]`

deRegister a listener for Dgnss injection status update.

Returns

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

5.7.1.2.2.5 `virtual telux::common::Status telux::loc::IDgnssManager::createSource ( DgnssDataFormat dataFormat )`
`[pure virtual]`

Create a Dgnss injection source. Only one source is permitted at any given time. If a new source is to be used, user must call [releaseSource\(\)](#) to release previous source before calling this function.

Parameters

<i>in</i>	<i>format</i>	Dgnss injection data format.

Returns

Success of suitable status code

5.7.1.2.2.6 `virtual telux::common::Status telux::loc::IDgnssManager::releaseSource ( void )`
`[pure virtual]`

Release current Dgnss injection source (previously created by [createSource\(\)](#) call) This function is to be called if it's determined that current injection data is not suitable anymore, and a new source will be created and used as injection source.

Parameters

<i>none</i>	

Returns

none

5.7.1.2.2.7 `virtual telux::common::Status telux::loc::IDgnssManager::injectCorrectionData ( const uint8_t* buffer, uint32_t bufferSize )`
`[pure virtual]`

Inject correction data This function is to be called when a source has been created, either through a explicit call to [createSource\(\)](#), or after DgnssManager object was instantiated through the factory method(The factory method create a default source for DgnssManager object).

Parameters

<i>in</i>	<i>buffer</i>	buffer contains the data to be injected.
<i>in</i>	<i>bufferSize</i>	size of the buffer.

Returns

success or suitable status code.

5.7.1.3 struct telux::loc::GnssKinematicsData

Specifies kinematics related information.

Data fields

Type	Field	Description
KinematicData-Validity	bodyFrame-DataMask	Contains Body frame LocPosDataMask bits.
float	longAccel	Forward Acceleration in body frame (m/s ²)
float	latAccel	Sideward Acceleration in body frame (m/s ²)
float	vertAccel	Vertical Acceleration in body frame (m/s ²)
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

5.7.1.4 struct telux::loc::LLAInfo

The location info is calculated according to the vehicle's GNSS antenna where as Vehicle Reference Point(-VRP) refers to a point on the vehicle where the display of the car sits. The VRP based info is calculated by adding that extra difference between GNSS antenna and the VRP on the top where the location info is recieved. The VRP parameters can be configured through [ILocationConfigurator::configureLeverArm](#). [LLAInfo](#) specifies latitude, longitude and altitude info of location for VRP-based.

Data fields

Type	Field	Description
double	latitude	Latitude, in unit of degrees, range [-90.0, 90.0].
double	longitude	Longitude, in unit of degrees, range [-180.0, 180.0].
float	altitude	Altitude above the WGS 84 reference ellipsoid, in unit of meters.

Type	Field	Description

5.7.1.5 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 purp due to possible discontinuities. Unit: Millisecond
uint32_t	numClock-Resets	Number of clock resets/discontinuities detected, affecting the local hardware counter value.

5.7.1.6 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 purposes due to possible discontinuities. Unit: Millisecond
uint32_t	numClock-Resets	Number of clock resets/discontinuities detected, affecting the local hardware counter value.
uint8_t	gloFourYear	GLONASS four year number from 1996. Refer to GLONASS ICD. Applicable only for GLONASS and shall be ignored for other constellations. If unknown shall be set to 255

5.7.1.7 union telux::loc::SystemTimeInfo**Data fields**

Type	Field	Description
TimeInfo	gps	
TimeInfo	gal	
TimeInfo	bds	
TimeInfo	qzss	
GlonassTime-Info	glo	

5.7.1.8 struct telux::loc::SystemTime

Data fields

Type	Field	Description
GnssSystem	gnssSystem-TimeSrc	Specifies GNSS system time reported. Mandatory field
SystemTime-Info	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

5.7.1.9 struct telux::loc::GnssMeasurementInfo

Data fields

Type	Field	Description
GnssSignal	gnssSignalType	GnssSignalType mask
GnssSystem	gnss-Constellation	Specifies GNSS Constellation Type
uint16_t	gnssSvId	GNSS SV ID. For GPS: 1 to 32 For GLONASS: 65 to 96. When slot-number to SV ID mapping is unknown, set as 255. For SBAS: 120 to 151 For QZSS-L1CA:193 to 197 For BDS: 201 to 237 For GAL: 301 to 336

5.7.1.10 struct telux::loc::SvUsedInPosition

Specify the set of SVs that are used to calculate GnssLocation.

Data fields

Type	Field	Description
uint64_t	gps	Specify the set of SVs from GPS constellation that are used to compute the position. Bit 0 to Bit 31 corresponds to GPS SV id 1 to 32.
uint64_t	glo	Specify the set of SVs from GLONASS constellation that are used to compute the position. Bit 0 to Bit 31 corresponds to GLO SV id 65 to 96.
uint64_t	gal	Specify the set of SVs from GALILEO constellation that are used to compute the position. Bit 0 to Bit 35 corresponds to GAL SV id 301 to 336.
uint64_t	bds	Specify the set of SVs from BEIDOU constellation that are used to compute the position. Bit 0 to Bit 36 corresponds to BDS SV id 201 to 237.

Type	Field	Description
uint64_t	qzss	Specify the set of SVs from QZSS constellation that are used to compute the position. Bit 0 to Bit 4 corresponds to BDS SV id 193 to 197.

5.7.1.11 struct telux::loc::GnssData

Data fields

Type	Field	Description
GnssData-Validity	gnssData-Mask[GnssDataSignalTypes::GNSS_DATA_MAX_NUMBER_OF_SIGNAL_TYPES]	bitwise OR of GnssDataValidityType
double	jammer-Ind[GnssDataSignalTypes::GNSS_DATA_MAX_NUMBER_OF_SIGNAL_TYPES]	Jammer Indication Each index represents the measurement for the signal type in GnssDataSignalTypes
double	agc[GnssDataSignalTypes::GNSS_DATA_MAX_NUMBER_OF_SIGNAL_TYPES]	Automatic gain control Each index represents the measurement for the signal type in GnssDataSignalTypes

5.7.1.12 struct telux::loc::SvBlackListInfo

Specify parameters related to enable/disable SVs

Data fields

Type	Field	Description
Gnss-Constellation-Type	constellation	constellation for the sv
uint32_t	svId	sv id for the constellation: 0 means blacklist for all SVIds of a given constellation type GLONASS SV id range: 65 to 96 QZSS SV id range: 193 to 197 BDS SV id range: 201 to 237 GAL SV id range: 301 to 336 SBAS SV id range: 120 to 158 and 183 to 191

5.7.1.13 struct telux::loc::LeverArmParams

Specify parameters related to lever arm

Data fields

Type	Field	Description
float	forwardOffset	Offset along the vehicle forward axis, in unit of meters
float	sidewaysOffset	Offset along the vehicle starboard axis, in unit of meters
float	upOffset	Offset along the vehicle up axis, in unit of meters

5.7.1.14 struct telux::loc::ExternalLocationInfo

Specifies parameters related to External Location information.

Data fields

Type	Field	Description
uint64_t	timestamp	UTC timestamp elapsed since January 1, 1970, in unit of milliseconds.
double	latitude	Latitude, in unit of degrees, range [-90.0, 90.0].
double	longitude	Longitude, in unit of degrees, range [-180.0, 180.0].
float	horizontal-Accuracy	Horizontal accuracy, in unit of meters. The best results are expected when the accuracy is estimated with 68% horizontal confidence.

5.7.1.15 struct telux::loc::GnssMeasurementsData

Specify the signal measurement information such as satellite vehicle pseudo range, satellite vehicle time, carrier phase measurement etc. from GNSS positioning engine.

Data fields

Type	Field	Description
Gnss-Measurements-DataValidity	valid	Bitwise OR of GnssMeasurementsDataValidityType to specify the valid fields in GnssMeasurementsData .
int16_t	svId	Specify satellite vehicle ID number.
Gnss-Constellation-Type	svType	SV constellation type.
double	timeOffsetNs	Time offset when the measurement was taken, in unit of nanoseconds.
Gnss-Measurements-StateValidity	stateMask	Bitwise OR of GnssMeasurementsStateValidityType to specify the GNSS measurement state.
int64_t	receivedSv-TimeNs	Received GNSS time of the week in nanoseconds when the measurement was taken.
int64_t	receivedSv-Time-UncertaintyNs	Satellite time. All SV times in the current measurement block are already propagated to a common reference time epoch, in unit of nano seconds.

Type	Field	Description
double	carrierToNoise-DbHz	Signal strength, carrier to noise ratio, in unit of dB-Hz.
double	pseudorange-RateMps	Uncorrected pseudorange rate, in unit of metres/second.
double	pseudorange-Rate-Uncertainty-Mps	Uncorrected pseudorange rate uncertainty, in unit of meters/second.
Gnss-Measurements-AdrState-Validity	adrStateMask	Bitwise OR of GnssMeasurementsAdrStateValidityType.
double	adrMeters	Accumulated delta range, in unit of meters.
double	adrUncertainty-Meters	Accumulated delta range uncertainty, in unit of meters.
float	carrier-FrequencyHz	Carrier frequency of the tracked signal, in unit of Hertz.
int64_t	carrierCycles	The number of full carrier cycles between the receiver and the satellite.
double	carrierPhase	The RF carrier phase that the receiver has detected.
double	carrierPhase-Uncertainty	The RF carrier phase uncertainty.
Gnss-Measurements-Multipath-Indicator	multipath-Indicator	Multipath indicator, could be unknown, present or not present.
double	signalToNoise-RatioDb	Signal to noise ratio, in unit of dB.
double	agcLevelDb	Automatic gain control level, in unit of dB.
GnssSignal	gnssSignalType	GnssSignalType mask
double	baseband-CarrierToNoise	Carrier-to-noise ratio of the signal measured at baseband, in unit of dB-Hz.
double	fullInterSignal-Bias	The full inter-signal bias (ISB) in nanoseconds. This value is the sum of the estimated receiver-side and the space-segment-side inter-system bias, inter-frequency bias and inter-code bias.
double	fullInterSignal-Bias-Uncertainty	Uncertainty associated with the full inter-signal bias in nanoseconds.

5.7.1.16 struct telux::loc::GnssMeasurementsClock

Specify GNSS measurements clock. The main equation describing the relationship between various components is: $utcTimeNs = timeNs - (fullBiasNs + biasNs) - leapSecond * 1,000,000,000$

Data fields

Type	Field	Description
Gnss-Measurements-ClockValidity	valid	Bitwise OR of GnssMeasurementsClockValidityType.
int16_t	leapSecond	Leap second, in unit of seconds.
int64_t	timeNs	Time, monotonically increasing as long as the power is on, in unit of nanoseconds.
double	time-UncertaintyNs	Time uncertainty (one sigma), in unit of nanoseconds.
int64_t	fullBiasNs	Full bias, in unit of nanoseconds.
double	biasNs	Sub-nanoseconds bias, in unit of nanoseconds.
double	bias-UncertaintyNs	Bias uncertainty (one sigma), in unit of nanoseconds.
double	driftNsps	Clock drift, in unit of nanoseconds/second.
double	drift-Uncertainty-Nsps	Clock drift uncertainty (one sigma), in unit of nanoseconds/second.
uint32_t	hwClock-Discontinuity-Count	HW clock discontinuity count - incremented for each discontinuity in HW clock.

5.7.1.17 struct telux::loc::GnssMeasurements

Specify GNSS measurements clock and data. [GnssMeasurementInfo](#) is used to convey the satellite vehicle info whose measurements are actually used to generate the current position report. While [GnssMeasurements](#) contains the satellite measurements that device observed during tracking session, regardless the measurement is used or not used to compute the fix. Furthermore [GnssMeasurements](#) contains much richer set of information which can enable other third party engines to utilize the measurements and compute the position by itself.

Data fields

Type	Field	Description
Gnss-Measurements-Clock	clock	GNSS measurements clock info.
vector< Gnss-Measurements-Data >	measurements	GNSS measurements data.

5.7.1.18 struct telux::loc::LeapSecondChangeInfo

Data fields

Type	Field	Description
TimeInfo	timeInfo	GPS timestamp that corresponds to the last known leap second change event. The info can be available on two scenario: 1: This leap second change event has been scheduled and yet to happen 2: This leap second change event has already happened and next leap second change event has not yet been scheduled.
uint8_t	leapSeconds-BeforeChange	Number of leap seconds prior to the leap second change event that corresponds to the timestamp at timeInfo.
uint8_t	leapSeconds-AfterChange	Number of leap seconds after the leap second change event that corresponds to the timestamp at timeInfo.

5.7.1.19 struct telux::loc::LeapSecondInfo**Data fields**

Type	Field	Description
LeapSecond-InfoValidity	valid	
uint8_t	current	Current leap seconds, in unit of seconds. This info will only be available only if the leap second change info is not available.
LeapSecond-ChangeInfo	info	

5.7.1.20 struct telux::loc::LocationSystemInfo**Data fields**

Type	Field	Description
Location-SystemInfo-Validity	valid	
LeapSecond-Info	info	

5.7.1.21 struct telux::loc::GnssEnergyConsumedInfo

Specify the info regarding energy consumed by GNSS engine.

Data fields

Type	Field	Description
GnssEnergy-ConsumedInfo-Validity	valid	Bitwise OR of GnssEnergyConsumedInfoValidityType to specify the valid fields in GnssEnergyConsumedInfo .
uint64_t	energySince-FirstBoot	Energy consumed by the modem GNSS engine since device first ever bootup, in unit of 0.1 milli watt seconds.

5.7.1.22 struct telux::loc::RobustLocationVersion

Specify the versioning info of robust location module for the GNSS standard position engine (SPE).

Data fields

Type	Field	Description
uint8_t	major	Major version number.
uint16_t	minor	Minor version number.

5.7.1.23 struct telux::loc::RobustLocationConfiguration

Specify the robust location configuration used by the GNSS standard position engine (SPE)

Data fields

Type	Field	Description
Robust-LocationConfig	validMask	Validity mask
bool	enabled	Specify whether robust location feature is enabled or not.
bool	enabledFor-E911	Specify whether robust location feature is enabled or not when device is on E911 call.
Robust-Location-Version	version	Specify the version info of robust location module used by the GNSS standard position engine (SPE).

5.7.1.24 struct telux::loc::NmeaConfig

Specify the Nmea Config Parameters

Data fields

Type	Field	Description
NmeaSentence-Config	sentenceConfig	Specify the sentences to be configured.
Geodetic-DatumType	datumType	Specify the datum type to be configured.

5.7.1.25 struct telux::loc::BodyToSensorMountParams

Specify vehicle body-to-Sensor mount parameters for use by dead reckoning positioning engine.

Data fields

Type	Field	Description
float	rollOffset	The misalignment of the sensor board along the horizontal plane of the vehicle chassis measured looking from the vehicle to forward direction. In unit of degrees. Range: [-180.0, 180.0].

Type	Field	Description
float	yawOffset	The misalignment along the horizontal plane of the vehicle chassis measured looking from the vehicle to the right side. Positive pitch indicates vehicle is inclined such that forward wheels are at higher elevation than rear wheels. In unit of degrees. Range: [-180.0, 180.0].
float	pitchOffset	The angle between the vehicle forward direction and the sensor axis as seen from the top of the vehicle, and measured in counterclockwise direction. In unit of degrees. Range: [-180.0, 180.0].
float	offsetUnc	Single uncertainty number that may be the largest of the uncertainties for roll offset, pitch offset and yaw offset. In unit of degrees. Range: [-180.0, 180.0].

5.7.1.26 struct telux::loc::DREngineConfiguration

Specify the dead reckoning engine configuration parameters.

Data fields

Type	Field	Description
DRConfig-Validity	validMask	Specify the valid fields.
BodyToSensor-MountParams	mountParam	Body to sensor mount parameters used by dead reckoning positioning engine.
float	speedFactor	Vehicle Speed Scale Factor configuration input for the dead reckoning positioning engine. The multiplicative scale factor is applied to the received Vehicle Speed value (in meter/second) to obtain the true Vehicle Speed. Range is [0.9 to 1.1]. Note: The scale factor is specific to a given vehicle make & model.
float	speedFactor-Unc	Vehicle Speed Scale Factor Uncertainty (68% confidence) configuration input for the dead reckoning positioning engine. Range is [0.0 to 0.1]. Note: The scale factor uncertainty is specific to a given vehicle make & model.
float	gyroFactor	Gyroscope Scale Factor configuration input for the dead reckoning positioning engine. The multiplicative scale factor is applied to received gyroscope value to obtain the true value. Range is [0.9 to 1.1]. Note: The scale factor is specific to the Gyroscope sensor and typically derived from either sensor data-sheet or from actual calibration.
float	gyroFactorUnc	Gyroscope Scale Factor uncertainty (68% confidence) configuration input for the dead reckoning positioning engine. Range is [0.0 to 0.1]. Note: The scale factor uncertainty is specific to the Gyroscope sensor and typically derived from either sensor data-sheet or from actual calibration.

5.7.1.27 struct telux::loc::XtraConfig

Xtra feature configuration parameters

Data fields

Type	Field	Description
uint32_t	download-IntervalMinute	Number of minutes between periodic, consecutive successful XTRA assistance data downloads. If 0 is specified, modem default download for XTRA assistance data will be performed.
uint32_t	download-TimeoutSec	Connection timeout when connecting backend for both xtra assistance data download and NTP time download. If 0 is specified, the download timeout value will use device default values.
uint32_t	download-RetryInterval-Minute	Interval to wait before retrying for xtra assistance data's download in case of failure. If 0 is specified, XTRA download retry will follow device default behavior and downloadRetryAttempts will also use device default value.
uint32_t	download-RetryAttempts	Total number of allowed retry attempts for assistance data's download in case of failure. If 0 is specified, XTRA download retry will follow device default behavior and downloadRetryIntervalMinute will also use device default value.
string	caPath	Path to the certificate authority (CA) repository that needs to be used for XTRA assistance data download. If empty string is specified, device default CA repository will be used.
vector< string >	serverURLs	URLs from which XTRA assistance data will be fetched. At least one and up to three URLs need to be configured when this API is used. The URLs, if provided, shall include the port number to be used for download. Valid xtra server URLs should start with "https://". Example of a valid URL : https://path.exampleserver.net:443
vector< string >	ntpServerURLs	URLs for NTP server to fetch current time. If no NTP server URL is provided, then device will use the default NTP server. The URLs, if provided, shall include the port number to be used for download. Example of a valid ntp server URL is: ntp.exampleserver.com:123.
bool	isIntegrity-Download-Enabled	Enable or disable XTRA integrity download. true: enable XTRA integrity download. false: disable XTRA integrity download.

Type	Field	Description
uint32_t	integrity-Download-IntervalMinute	Download interval for xtra integrity, only applicable if XTRA integrity download is enabled. If 0 is specified, the download timeout value will use device default value.
DebugLog-Level	daemonDebug-LogLevel	Level of debug log messages that will be logged.

5.7.1.28 struct telux::loc::XtraStatus

Specify Xtra assistant data's current status, validity and whether it is enabled.

Data fields

Type	Field	Description
bool	featureEnabled	XTRA assistance data and NTP time download is enabled or disabled.
XtraDataStatus	xtraDataStatus	XTRA assistance data status. If XTRA assistance data download is not enabled, this field will be set to XTRA_DATA_STATUS_UNKNOWN.
uint32_t	xtraValidFor-Hours	Number of hours that xtra assistance data will remain valid. This field will be valid when xtraDataStatus is set to XTRA_DATA_STATUS_VALID. For all other XtraDataStatus, this field will be set to 0.

5.7.1.29 class telux::loc::ILocationInfoBase

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

Public member functions

- virtual [LocationInfoValidity](#) [getLocationInfoValidity](#) ()=0
- virtual [LocationTechnology](#) [getTechMask](#) ()=0
- virtual float [getSpeed](#) ()=0
- virtual double [getLatitude](#) ()=0
- virtual double [getLongitude](#) ()=0
- virtual double [getAltitude](#) ()=0
- virtual float [getHeading](#) ()=0
- virtual float [getHorizontalUncertainty](#) ()=0
- virtual float [getVerticalUncertainty](#) ()=0
- virtual uint64_t [getTimeStamp](#) ()=0
- virtual float [getSpeedUncertainty](#) ()=0

- virtual float [getHeadingUncertainty](#) ()=0
- virtual uint64_t [getElapsedRealTime](#) ()=0
- virtual uint64_t [getElapsedRealTimeUncertainty](#) ()=0
- virtual [~ILocationInfoBase](#) ()

5.7.1.29.1 Constructors and Destructors

5.7.1.29.1.1 virtual [telux::loc::ILocationInfoBase::~ILocationInfoBase](#) () [virtual]

Destructor for [ILocationInfoBase](#)

5.7.1.29.2 Member Function Documentation

5.7.1.29.2.1 virtual LocationInfoValidity [telux::loc::ILocationInfoBase::getLocationInfoValidity](#) ()
[pure virtual]

Retrieves the validity of the Location basic Info.

Returns

Location basic validity mask.

5.7.1.29.2.2 virtual LocationTechnology [telux::loc::ILocationInfoBase::getTechMask](#) () [pure virtual]

Retrieves technology used in computing this fix.

Returns

Location technology mask.

5.7.1.29.2.3 virtual float [telux::loc::ILocationInfoBase::getSpeed](#) () [pure virtual]

Retrieves Speed.

Returns

speed in meters per second.

5.7.1.29.2.4 virtual double telux::loc::ILocationInfoBase::getLatitude () [pure virtual]

Retrieves latitude. Positive and negative values indicate northern and southern latitude respectively

- Units: Degrees
- Range: -90.0 to 90.0

Returns

Latitude if available else returns NaN.

5.7.1.29.2.5 virtual double telux::loc::ILocationInfoBase::getLongitude () [pure virtual]

Retrieves longitude. Positive and negative values indicate eastern and western longitude respectively

- Units: Degrees
- Range: -180.0 to 180.0

Returns

Longitude if available else returns NaN.

5.7.1.29.2.6 virtual double telux::loc::ILocationInfoBase::getAltitude () [pure virtual]

Retrieves altitude above the WGS 84 reference ellipsoid.

- Units: Meters

Returns

Altitude if available else returns NaN.

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

Retrieves heading/bearing.

- Units: Degrees
- Range: 0 to 359.999

Returns

Heading if available else returns NaN.

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

Retrieves the horizontal uncertainty.

Returns

Horizontal uncertainty.

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

Retrieves the vertical uncertainty.

- Units: Meters

Returns

Vertical uncertainty if available else returns NaN.

5.7.1.29.2.10 virtual uint64_t telux::loc::ILocationInfoBase::getTimeStamp () [pure virtual]

Retrieves UTC timeInfo for the location fix.

- Units: Milliseconds since Jan 1, 1970

Returns

TimeStamp in milliseconds if available else returns UNKNOWN_TIMESTAMP which is zero(as UTC timeStamp has elapsed since January 1, 1970, it cannot be 0)

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

Retrieves 3-D speed uncertainty/accuracy.

- Units: Meters per Second

Returns

Speed uncertainty if available else returns NaN.

5.7.1.29.2.12 virtual float telux::loc::ILocationInfoBase::getHeadingUncertainty () [pure virtual]

Retrieves heading uncertainty.

- Units: Degrees
- Range: 0 to 359.999

Returns

Heading uncertainty if available else returns NaN.

5.7.1.29.2.13 `virtual uint64_t telux::loc::ILocationInfoBase::getElapsedRealTime ( ) [pure virtual]`

Boot timestamp corresponding to the UTC timestamp for Location fix.

- Units: Nano-second

Returns

elapsed real time.

5.7.1.29.2.14 `virtual uint64_t telux::loc::ILocationInfoBase::getElapsedRealTimeUncertainty ( ) [pure virtual]`

Retrieves elapsed real time uncertainty.

- Units: Nano-second

Returns

elapsed real time uncertainty.

5.7.1.30 `class telux::loc::ILocationInfoEx`

[ILocationInfoEx](#) provides interface to get richer position related information like latitude, longitude, altitude and other information like time stamp, session status, dop, reliabilities, uncertainties etc.

Public member functions

- virtual [LocationInfoExValidity](#) `getLocationInfoExValidity ()=0`
- virtual float `getAltitudeMeanSeaLevel ()=0`
- virtual float `getPositionDop ()=0`
- virtual float `getHorizontalDop ()=0`
- virtual float `getVerticalDop ()=0`
- virtual float `getGeometricDop ()=0`
- virtual float `getTimeDop ()=0`
- virtual float `getMagneticDeviation ()=0`
- virtual [LocationReliability](#) `getHorizontalReliability ()=0`
- virtual [LocationReliability](#) `getVerticalReliability ()=0`
- virtual float `getHorizontalUncertaintySemiMajor ()=0`
- virtual float `getHorizontalUncertaintySemiMinor ()=0`
- virtual float `getHorizontalUncertaintyAzimuth ()=0`
- virtual float `getEastStandardDeviation ()=0`

- virtual float `getNorthStandardDeviation` ()=0
- virtual uint16_t `getNumSvUsed` ()=0
- virtual `SvUsedInPosition` `getSvUsedInPosition` ()=0
- virtual void `getSVIds` (std::vector< uint16_t > &idsOfUsedSVs)=0
- virtual `SbasCorrection` `getSbasCorrection` ()=0
- virtual `GnssPositionTech` `getPositionTechnology` ()=0
- virtual `GnssKinematicsData` `getBodyFrameData` ()=0
- virtual std::vector
< `GnssMeasurementInfo` > `getmeasUsageInfo` ()=0
- virtual `SystemTime` `getGnssSystemTime` ()=0
- virtual float `getTimeUncMs` ()=0
- virtual `telux::common::Status` `getLeapSeconds` (uint8_t &leapSeconds)=0
- virtual `telux::common::Status` `getVelocityEastNorthUp` (std::vector< float > &velocityEastNorthUp)=0
- virtual `telux::common::Status` `getVelocityUncertaintyEastNorthUp` (std::vector< float > &velocityUncertaintyEastNorthUp)=0
- virtual uint8_t `getCalibrationConfidencePercent` ()=0
- virtual `DrCalibrationStatus` `getCalibrationStatus` ()=0
- virtual `LocationAggregationType` `getLocOutputEngType` ()=0
- virtual `PositioningEngine` `getLocOutputEngMask` ()=0
- virtual float `getConformityIndex` ()=0
- virtual `LLAInfo` `getVRPBasedLLA` ()=0
- virtual std::vector< float > `getVRPBasedENUVelocity` ()=0
- virtual `AltitudeType` `getAltitudeType` ()=0
- virtual `ReportStatus` `getReportStatus` ()=0
- virtual uint32_t `getIntegrityRiskUsed` ()=0
- virtual float `getProtectionLevelAlongTrack` ()=0
- virtual float `getProtectionLevelCrossTrack` ()=0
- virtual float `getProtectionLevelVertical` ()=0
- virtual std::vector< uint16_t > `getDgnssStationIds` ()=0
- virtual `~ILocationInfoEx` ()

5.7.1.30.1 Constructors and Destructors

5.7.1.30.1.1 `virtual telux::loc::ILocationInfoEx::~~ILocationInfoEx ( ) [virtual]`

Destructor for [ILocationInfoEx](#)

5.7.1.30.2 Member Function Documentation

5.7.1.30.2.1 `virtual LocationInfoExValidity telux::loc::ILocationInfoEx::getLocationInfoExValidity ( ) [pure virtual]`

Retrives the validity of the location info ex. It provides the validity of various information like dop, reliabilities, uncertainties etc.

Returns

Location ex validity mask

5.7.1.30.2.2 `virtual float telux::loc::ILocationInfoEx::getAltitudeMeanSeaLevel ( ) [pure virtual]`

Retrieves the altitude with respect to mean sea level.

- Units: Meters

Returns

Altitude with respect to mean sea level if available else returns NaN.

5.7.1.30.2.3 `virtual float telux::loc::ILocationInfoEx::getPositionDop ( ) [pure virtual]`

Retrieves position dilution of precision.

Returns

Position dilution of precision if available else returns NaN. Range: 1 (highest accuracy) to 50 (lowest accuracy)

5.7.1.30.2.4 `virtual float telux::loc::ILocationInfoEx::getHorizontalDop ( ) [pure virtual]`

Retrieves horizontal dilution of precision.

Returns

Horizontal dilution of precision if available else returns NaN. Range: 1 (highest accuracy) to 50 (lowest accuracy)

5.7.1.30.2.5 virtual float telux::loc::ILocationInfoEx::getVerticalDop () [pure virtual]

Retrieves vertical dilution of precision.

Returns

Vertical dilution of precision if available else returns NaN Range: 1 (highest accuracy) to 50 (lowest accuracy)

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

Retrieves geometric dilution of precision.

Returns

geometric dilution of precision.

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

Retrieves time dilution of precision.

Returns

Time dilution of precision.

5.7.1.30.2.8 virtual float telux::loc::ILocationInfoEx::getMagneticDeviation () [pure virtual]

Retrieves the difference between the bearing to true north and the bearing shown on magnetic compass. The deviation is positive when the magnetic north is east of true north.

- Units: Degrees

Returns

Magnetic Deviation if available else returns NaN

5.7.1.30.2.9 virtual LocationReliability telux::loc::ILocationInfoEx::getHorizontalReliability () [pure virtual]

Specifies the reliability of the horizontal position.

Returns

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

5.7.1.30.2.10 virtual LocationReliability telux::loc::ILocationInfoEx::getVerticalReliability () [pure virtual]

Specifies the reliability of the vertical position.

Returns

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

5.7.1.30.2.11 virtual float telux::loc::ILocationInfoEx::getHorizontalUncertaintySemiMajor () [pure virtual]

Retrieves semi-major axis of horizontal elliptical uncertainty.

- Units: Meters

Returns

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

5.7.1.30.2.12 virtual float telux::loc::ILocationInfoEx::getHorizontalUncertaintySemiMinor () [pure virtual]

Retrieves semi-minor axis of horizontal elliptical uncertainty.

- Units: Meters

Returns

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

5.7.1.30.2.13 virtual float telux::loc::ILocationInfoEx::getHorizontalUncertaintyAzimuth () [pure virtual]

Retrieves elliptical horizontal uncertainty azimuth of orientation.

- Units: Decimal degrees
- Range: 0 to 180

Returns

Elliptical horizontal uncertainty azimuth of orientation if available else returns NaN.

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

Retrieves east standard deviation.

- Units: Meters

Returns

East Standard Deviation.

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

Retrieves north standard deviation.

- Units: Meters

Returns

North Standard Deviation.

5.7.1.30.2.16 `virtual uint16_t telux::loc::ILocationInfoEx::getNumSvUsed ( ) [pure virtual]`

Retrieves number of satellite vehicles used in position report.

Returns

number of Sv used.

5.7.1.30.2.17 `virtual SvUsedInPosition telux::loc::ILocationInfoEx::getSvUsedInPosition ( ) [pure virtual]`

Retrives the set of satellite vehicles that are used to calculate position.

Returns

set of satellite vehicles for different constellations.

5.7.1.30.2.18 `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.

5.7.1.30.2.19 `virtual SbasCorrection telux::loc::ILocationInfoEx::getSbasCorrection ( ) [pure virtual]`

Retrieves navigation solution mask used to indicate SBAS corrections.

Returns

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

5.7.1.30.2.20 `virtual GnssPositionTech telux::loc::ILocationInfoEx::getPositionTechnology ( ) [pure virtual]`

Retrieves position technology mask used to indicate which technology is used.

Returns

- Position technology used in computing this fix.

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

Retrieves position related information.

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

Retrieves gnss measurement usage info.

5.7.1.30.2.23 `virtual SystemTime telux::loc::ILocationInfoEx::getGnssSystemTime ( ) [pure virtual]`

Retrieves type of gnss system.

Returns

- Type of Gnss System.

5.7.1.30.2.24 `virtual float telux::loc::ILocationInfoEx::getTimeUncMs ( ) [pure virtual]`

Retrieves time uncertainty.

Returns

- Time uncertainty in milliseconds.

5.7.1.30.2.25 `virtual telux::common::Status telux::loc::ILocationInfoEx::getLeapSeconds ( uint8_t & leapSeconds ) [pure virtual]`

Retrieves leap seconds if available.

Parameters

out	<i>leapSeconds</i>	- leap seconds • Units: Seconds

Returns

Status of leap seconds.

5.7.1.30.2.26 `virtual telux::common::Status telux::loc::ILocationInfoEx::getVelocityEastNorthUp ( std::vector< float > & velocityEastNorthUp ) [pure virtual]`

Retrieves east, North, Up velocity if available.

Parameters

out	<i>velocityEastNorthUp</i>	- east, North, Up velocity • Units: Meters/second

Returns

Status of availability of east, North, Up velocity.

5.7.1.30.2.27 `virtual telux::common::Status telux::loc::ILocationInfoEx::getVelocityUncertainty-EastNorthUp ( std::vector< float > & velocityUncertaintyEastNorthUp ) [pure virtual]`

Retrieves east, North, Up velocity uncertainty if available.

Parameters

out	<i>velocityUncertainty-EastNorthUp</i>	- east, North, Up velocity uncertainty • Units: Meters/second

Returns

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

5.7.1.30.2.28 `virtual uint8_t telux::loc::ILocationInfoEx::getCalibrationConfidencePercent ( ) [pure virtual]`

Sensor calibration confidence percent, range [0, 100].

Returns

the percentage of calibration taking all the parameters into account.

5.7.1.30.2.29 virtual DrCalibrationStatus telux::loc::ILocationInfoEx::getCalibrationStatus () [pure virtual]

Sensor calibration status.

Returns

mask indicating the calibration status with respect to different parameters.

5.7.1.30.2.30 virtual LocationAggregationType telux::loc::ILocationInfoEx::getLocOutputEngType () [pure virtual]

Location engine type. When the type is set to LOC_ENGINE_SRC_FUSED, the fix is the propagated/aggregated reports from all engines running on the system (e.g.: DR/SPE/PPE) based QTI algorithm. To check which location engine contributes to the fused output, check for locOutputEngMask.

Returns

the type of engine that was used for calculating the position fix.

5.7.1.30.2.31 virtual PositioningEngine telux::loc::ILocationInfoEx::getLocOutputEngMask () [pure virtual]

When loc output eng type is set to fused, this field indicates the set of engines contribute to the fix.

Returns

the combination of position engines used in calculating the position report when the loc output end type is set to fused.

5.7.1.30.2.32 virtual float telux::loc::ILocationInfoEx::getConformityIndex () [pure virtual]

When robust location is enabled, this field will indicate how well the various input data considered for navigation solution conforms to expectations.

Returns

values in the range [0.0, 1.0], with 0.0 for least conforming and 1.0 for most conforming.

5.7.1.30.2.33 virtual LLAInfo telux::loc::ILocationInfoEx::getVRPBasedLLA () [pure virtual]

Vehicle Reference Point(VRP) based latitude, longitude and altitude information.

5.7.1.30.2.34 `virtual std::vector<float> telux::loc::ILocationInfoEx::getVRPBasedENUVelocity ( )`
`[pure virtual]`

VRP-based east, north and up velocity information.

Returns

- vector of directional velocities in this order {east velocity, north velocity, up velocity}

5.7.1.30.2.35 `virtual AltitudeType telux::loc::ILocationInfoEx::getAltitudeType ( )` `[pure virtual]`

Determination of altitude is assumed or calculated. ASSUMED means there may not be enough satellites to determine the precise altitude.

Returns

altitude type ASSUMED/CALCULATED or if not available then UNKNOWN.

5.7.1.30.2.36 `virtual ReportStatus telux::loc::ILocationInfoEx::getReportStatus ( )` `[pure virtual]`

Indicates the status of this report in terms of how optimally the report was calculated by the engine.

Returns

Status of the report. Returns ReportStatus::UNKNOWN if status is unavailable.

5.7.1.30.2.37 `virtual uint32_t telux::loc::ILocationInfoEx::getIntegrityRiskUsed ( )` `[pure virtual]`

Integrity risk used for protection level parameters. Unit of 2.5e-10. Valid range is [1 to (4e9-1)]. Values other than valid range means integrity risk is disabled and [ILocationInfoEx::getProtectionLevelAlongTrack](#), [ILocationInfoEx::getProtectionLevelCrossTrack](#) and [ILocationInfoEx::getProtectionLevelVertical](#) will not be available.

5.7.1.30.2.38 `virtual float telux::loc::ILocationInfoEx::getProtectionLevelAlongTrack ( )` `[pure virtual]`

Along-track protection level at specified integrity risk, in unit of meter.

5.7.1.30.2.39 `virtual float telux::loc::ILocationInfoEx::getProtectionLevelCrossTrack ( )` `[pure virtual]`

Cross-track protection level at specified integrity risk, in unit of meter.

5.7.1.30.2.40 `virtual float telux::loc::ILocationInfoEx::getProtectionLevelVertical ( )` `[pure virtual]`

Vertical component protection level at specified integrity risk, in unit of meter.

5.7.1.30.2.41 `virtual std::vector<uint16_t> telux::loc::ILocationInfoEx::getDgnssStationIds ( )`
`[pure virtual]`

List of DGNSS station IDs providing corrections. Range:

- SBAS – 120 to 158 and 183 to 191
- Monitoring station – 1000-2023 (Station ID biased by 1000)
- Other values reserved.

5.7.1.31 `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
- virtual [~ISVInfo](#) `()`

5.7.1.31.1 Constructors and Destructors

5.7.1.31.1.1 `virtual telux::loc::ISVInfo::~ISVInfo ( )` `[virtual]`

Destructor for [ISVInfo](#)

5.7.1.31.2 Member Function Documentation

5.7.1.31.2.1 virtual GnssConstellationType telux::loc::ISVInfo::getConstellation () [pure virtual]

Indicates to which constellation this satellite vehicle belongs.

Returns

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

5.7.1.31.2.2 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)

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

Health status of satellite vehicle.

Returns

HealthStatus of Satellite Vehicle if available else returns UNKNOWN.

- [SVHealthStatus](#)

5.7.1.31.2.4 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](#)

5.7.1.31.2.5 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.

5.7.1.31.2.6 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.

5.7.1.31.2.7 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.

5.7.1.31.2.8 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.

5.7.1.31.2.9 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.

5.7.1.31.2.10 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.

5.7.1.31.2.11 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.

5.7.1.31.2.12 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.

5.7.1.32 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
- virtual ~[IGnssSVInfo](#) ()

5.7.1.32.1 Constructors and Destructors**5.7.1.32.1.1 virtual telux::loc::IGnssSVInfo::~IGnssSVInfo () [virtual]**

Destructor for [IGnssSVInfo](#)

5.7.1.32.2 Member Function Documentation**5.7.1.32.2.1 virtual AltitudeType telux::loc::IGnssSVInfo::getAltitudeType () [pure virtual]**

Indicates whether altitude is assumed or calculated.

Returns

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

5.7.1.32.2.2 `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.

5.7.1.33 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`
- virtual `~IGnssSignalInfo ()`

5.7.1.33.1 Constructors and Destructors

5.7.1.33.1.1 `virtual telux::loc::IGnssSignalInfo::~~IGnssSignalInfo ( ) [virtual]`

Destructor for [IGnssSignalInfo](#)

5.7.1.33.2 Member Function Documentation

5.7.1.33.2.1 `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.

5.7.1.34 class telux::loc::LocationFactory

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

Public member functions

- `std::shared_ptr< ILocationManager > getLocationManager ()`
- `std::shared_ptr< ILocationConfigurator > getLocationConfigurator ()`
- `std::shared_ptr< IDgnssManager > getDgnssManager (DgnssDataFormat`

dataFormat=DgnssDataFormat::DATA_FORMAT_RTCM_3)

Static Public Member Functions

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

5.7.1.34.1 Member Function Documentation

5.7.1.34.1.1 static [LocationFactory](#)& [telux::loc::LocationFactory::getInstance](#) () [static]

Get Location Factory instance.

5.7.1.34.1.2 std::shared_ptr<[ILocationManager](#)> [telux::loc::LocationFactory::getLocationManager](#) ()

Get instance of Location Manager

Returns

Pointer of [ILocationManager](#) object.

5.7.1.34.1.3 std::shared_ptr<[ILocationConfigurator](#)> [telux::loc::LocationFactory::getLocationConfigurator](#) ()

Get instance of Location Configurator.

Returns

Pointer of [ILocationConfigurator](#) object.

5.7.1.34.1.4 std::shared_ptr<[IDgnssManager](#)> [telux::loc::LocationFactory::getDgnssManager](#) ([DgnssDataFormat dataFormat = DgnssDataFormat::DATA_FORMAT_RTCM_3](#))

Get instance of Dgnss manager

Returns

Pointer of [IDgnssManager](#) object.

5.7.1.35 class [telux::loc::ILocationListener](#)

Listener class for getting location updates and satellite vehicle information.

The methods in listener can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe.

Public member functions

- virtual void [onBasicLocationUpdate](#) (const std::shared_ptr< [ILocationInfoBase](#) > &locationInfo)
- virtual void [onDetailedLocationUpdate](#) (const std::shared_ptr< [ILocationInfoEx](#) > &locationInfo)
- virtual void [onDetailedEngineLocationUpdate](#) (const std::vector< std::shared_ptr< [ILocationInfoEx](#) > > &locationEngineInfo)
- virtual void [onGnssSVInfo](#) (const std::shared_ptr< [IGnssSVInfo](#) > &gnssSVInfo)
- virtual void [onGnssSignalInfo](#) (const std::shared_ptr< [IGnssSignalInfo](#) > &info)
- virtual void [onGnssNmeaInfo](#) (uint64_t timestamp, const std::string &nmea)
- virtual void [onGnssMeasurementsInfo](#) (const [telux::loc::GnssMeasurements](#) &measurementInfo)
- virtual [~ILocationListener](#) ()

5.7.1.35.1 Constructors and Destructors

5.7.1.35.1.1 virtual [telux::loc::ILocationListener::~~ILocationListener](#) () [[virtual](#)]

Destructor of [ILocationListener](#)

5.7.1.35.2 Member Function Documentation

5.7.1.35.2.1 virtual void [telux::loc::ILocationListener::onBasicLocationUpdate](#) (const std::shared_ptr< [ILocationInfoBase](#) > & *locationInfo*) [[virtual](#)]

This function is called when device receives location update.

Parameters

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

5.7.1.35.2.2 virtual void [telux::loc::ILocationListener::onDetailedLocationUpdate](#) (const std::shared_ptr< [ILocationInfoEx](#) > & *locationInfo*) [[virtual](#)]

This function is called when device receives Gnss location update.

Parameters

in	<i>locationInfo</i>	- Contains richer set of location information like latitude, longitude, timeInfo, heading, altitude, velocity and other information such as deviations, elliptical accuracies etc.

5.7.1.35.2.3 `virtual void telux::loc::ILocationListener::onDetailedEngineLocationUpdate ( const std::vector< std::shared_ptr< ILocationInfoEx > > & locationEngineInfo ) [virtual]`

This function is called when device receives multiple Gnss location update from the different engine types requested, which are SPE/PPE/FUSED. This API will be called ONLY if we use startDetailedEngineReports.

Parameters

in	<i>locationInfo</i>	- Contains a list of location infos. Each element in the list corresponds to one of SPE/PPE/FUSED.

5.7.1.35.2.4 `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

5.7.1.35.2.5 `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

5.7.1.35.2.6 `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

5.7.1.35.2.7 `virtual void telux::loc::ILocationListener::onGnssMeasurementsInfo ( const telux::loc::GnssMeasurements & measurementInfo ) [virtual]`

This function is called when device receives signal measurement information such as satellite vehicle pseudo range, satellite vehicle clock time, carrier phase measurement etc.

in	<i>measurementInfo</i>	- GNSS measurement information

Note

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

5.7.1.36 class telux::loc::ILocationSystemInfoListener**Public member functions**

- virtual void [onLocationSystemInfo](#) (const [LocationSystemInfo](#) &locationSystemInfo)
- virtual [~ILocationSystemInfoListener](#) ()

5.7.1.36.1 Constructors and Destructors

5.7.1.36.1.1 virtual telux::loc::ILocationSystemInfoListener::~~ILocationSystemInfoListener ()
[virtual]

Destructor of [ILocationSystemInfoListener](#)

5.7.1.36.2 Member Function Documentation

5.7.1.36.2.1 virtual void telux::loc::ILocationSystemInfoListener::onLocationSystemInfo (const [LocationSystemInfo](#) & *locationSystemInfo*) [virtual]

This function is called when device receives location related system information such as leap second change.

Parameters

in	<i>locationSystemInfo</i>	- contains location system information such as current leap seconds change

5.7.1.37 class telux::loc::ILocationInjectionListener**Public member functions**

- virtual void [onStartInjection](#) (const uint32_t timeInMilliseconds)
- virtual void [onStopInjection](#) ()
- virtual [~ILocationInjectionListener](#) ()

5.7.1.37.1 Constructors and Destructors

5.7.1.37.1.1 virtual telux::loc::ILocationInjectionListener::~~ILocationInjectionListener () [virtual]

Destructor of [ILocationSystemInfoListener](#)

5.7.1.37.2 Member Function Documentation

5.7.1.37.2.1 virtual void telux::loc::ILocationInjectionListener::onStartInjection (const uint32_t *timeInMilliseconds*) [virtual]

This function is called when the Location service is ready to start receiving the location reports using the LocationConfigurator::injectLocationData API.

Parameters

in	<i>timeInMillis</i>	- contains the time in milliseconds which represents the Max rate of at which the LocationConfigurator::injectLocationData API can be called.

5.7.1.37.2.2 virtual void telux::loc::ILocationInjectionListener::onStopInjection () [virtual]

This function notifies the Location service's intent to stop injecting the location reports using the LocationConfigurator::injectLocationData API.

5.7.1.38 class telux::loc::ILocationConfigListener

[ILocationConfigListener](#) interface is used to receive notifications related to configuration events.

The listener method can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe.

Public member functions

- virtual void [onXtraStatusUpdate](#) (const [XtraStatus](#) xtraStatus)
- virtual [~ILocationConfigListener](#) ()

5.7.1.38.1 Constructors and Destructors

5.7.1.38.1.1 virtual telux::loc::ILocationConfigListener::~~ILocationConfigListener () [virtual]

5.7.1.38.2 Member Function Documentation

5.7.1.38.2.1 virtual void telux::loc::ILocationConfigListener::onXtraStatusUpdate (const XtraStatus *xtraStatus*) [virtual]

The API is invoked when there is any update in the Xtra assistance data.

<code>in</code>	<code>xtraStatus</code>	- Xtra assistant data's current status, validity and whether it is enabled.

5.7.2 Enumeration Type Documentation

5.7.2.1 `enum telux::loc::DgnssDataFormat`

Defines RTCM injection data format

Enumerator

DATA_FORMAT_UNKNOWN Source data format is unknown
DATA_FORMAT_RTCM_3 Source data format is RTCM_3
DATA_FORMAT_3GPP_RTK_R15 Source data format is 3GPP RTK Rel-15

5.7.2.2 `enum telux::loc::DgnssStatus`

Defines status reported by cdfw for RTCM injection.

Enumerator

DATA_SOURCE_NOT_SUPPORTED Dgnss subsystem doesn't support the data source
DATA_FORMAT_NOT_SUPPORTED Dgnss subsystem doesn't support the data format
OTHER_SOURCE_IN_USE After the source injects the data, dgnss subsystem discovers there is another higher priority source injecting the data at the same time, and the current injected data is dropped
MESSAGE_PARSE_ERROR There is a parsing error such as unrecognized format, CRC check failure, value range check failure, etc.; the injected data is dropped
DATA_SOURCE_NOT_USABLE Data source is not usable anymore

5.7.2.3 `enum telux::loc::HorizontalAccuracyLevel`

Defines the horizontal accuracy level of the fix.

Enumerator

LOW Client requires low horizontal accuracy
MEDIUM Client requires medium horizontal accuracy
HIGH Client requires high horizontal accuracy

5.7.2.4 `enum telux::loc::LocationReliability`

Specifies the reliability of the position.

Enumerator

UNKNOWN Unknown location reliability
NOT_SET Location reliability is not set
VERY_LOW Location reliability is very low
LOW Location reliability is low, little or no cross-checking is possible
MEDIUM Location reliability is medium, limited cross-check passed

HIGH Location reliability is high, strong cross-check passed

5.7.2.5 enum telux::loc::AltitudeType

Indicates whether altitude is assumed or calculated.

Enumerator

UNKNOWN Unknown altitude type

CALCULATED Altitude is calculated

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

5.7.2.6 enum telux::loc::GnssConstellationType

Defines constellation type of GNSS.

Enumerator

UNKNOWN Unknown constellation type

GPS GPS satellite

GALILEO GALILEO satellite

SBAS SBAS satellite

COMPASS COMPASS satellite.

Deprecated constellation type is not supported.

GLONASS GLONASS satellite

BDS BDS satellite

QZSS QZSS satellite

NAVIC NAVIC satellite

5.7.2.7 enum telux::loc::SVHealthStatus

Health status indicates whether satellite is operational or not. This information comes from the most recent data transmitted in satellite almanacs.

Enumerator

UNKNOWN Unknown sv health status

UNHEALTHY satellite is not operational and cannot be used in position calculations

HEALTHY satellite is fully operational

5.7.2.8 enum telux::loc::SVStatus

Satellite vehicle processing status.

Enumerator

UNKNOWN Unknown sv status

IDLE SV is not being actively processed

SEARCH The system is searching for this SV

TRACK SV is being tracked

5.7.2.9 enum telux::loc::SVInfoAvailability

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

Enumerator

UNKNOWN Unknown sv info availability
YES Ephemeris or Almanac exists
NO Ephemeris or Almanac doesn't exist

5.7.2.10 enum telux::loc::GnssPositionTechType

Specifies which position technology was used to generate location information in the [ILocationInfoEx](#).

Enumerator

GNSS_DEFAULT Technology used to generate location info is unknown.
GNSS_SATELLITE Satellites-based technology was used to generate location info.
GNSS_CELLID Cell towers were used to generate location info.
GNSS_WIFI Wi-Fi access points were used to generate location info.
GNSS_SENSORS Sensors were used to generate location info.
GNSS_REFERENCE_LOCATION Reference location was used to generate location info.
GNSS_INJECTED_COARSE_POSITION Coarse position injected into the location engine was used to generate location info.
GNSS_AFLT AFLT was used to generate location info.
GNSS_HYBRID GNSS and network-provided measurements were used to generate location info.
GNSS_PPE Precise position engine was used to generate location info.
GNSS_VEHICLE Location was calculated using Vehicular data.
GNSS_VISUAL Location was calculated using Visual data.
GNSS_PROPAGATED Location was calculated using Propagation logic, which uses cached measurements.

5.7.2.11 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

5.7.2.12 enum telux::loc::GnssSystem

Specifies type of system.

Enumerator

GNSS_LOC_SV_SYSTEM_UNKNOWN UNKNOWN satellite.
GNSS_LOC_SV_SYSTEM_GPS GPS satellite.
GNSS_LOC_SV_SYSTEM_GALILEO GALILEO satellite.
GNSS_LOC_SV_SYSTEM_SBAS SBAS satellite.
GNSS_LOC_SV_SYSTEM_COMPASS COMPASS satellite.
Deprecated constellation type is not supported.
GNSS_LOC_SV_SYSTEM_GLONASS GLONASS satellite.
GNSS_LOC_SV_SYSTEM_BDS BDS satellite.
GNSS_LOC_SV_SYSTEM_QZSS QZSS satellite.
GNSS_LOC_SV_SYSTEM_NAVIC NAVIC satellite.

5.7.2.13 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

5.7.2.14 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

5.7.2.15 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

5.7.2.16 enum telux::loc::LocationTechnologyType**Enumerator**

LOC_GNSS location was calculated using GNSS
LOC_CELL location was calculated using Cell
LOC_WIFI location was calculated using WiFi
LOC_SENSORS location was calculated using Sensors
LOC_REFERENCE_LOCATION Location was calculated using Reference location.
LOC_INJECTED_COARSE_POSITION Location was calculated using Coarse position injected into the location engine.
LOC_AFLT Location was calculated using AFLT.
LOC_HYBRID Location was calculated using GNSS and network-provided measurements.
LOC_PPE Location was calculated using Precise position engine.
LOC_VEH Location was calculated using Vehicular data.
LOC_VIS Location was calculated using Visual data.
LOC_PROPAGATED Location was calculated using Propagation logic, which uses cached measurements.

5.7.2.17 enum telux::loc::LocationValidityType**Enumerator**

HAS_LAT_LONG_BIT Location has valid latitude and longitude.
HAS_ALTITUDE_BIT Location has valid altitude.
HAS_SPEED_BIT Location has valid speed.
HAS_HEADING_BIT Location has valid heading.

HAS_HORIZONTAL_ACCURACY_BIT
HAS_VERTICAL_ACCURACY_BIT Location has valid vertical accuracy.
HAS_SPEED_ACCURACY_BIT Location has valid speed accuracy.
HAS_HEADING_ACCURACY_BIT Location has valid heading accuracy.
HAS_TIMESTAMP_BIT Location has valid timestamp.
HAS_ELAPSED_REAL_TIME_BIT Location has valid elapsed real time.
HAS_ELAPSED_REAL_TIME_UNC_BIT Location has valid elapsed real time uncertainty.

5.7.2.18 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_NUM_SV_USED_IN_POSITION valid number of sv used
HAS_CALIBRATION_CONFIDENCE_PERCENT valid sensor calibrationConfidencePercent
HAS_CALIBRATION_STATUS valid sensor calibrationConfidence
HAS_OUTPUT_ENG_TYPE valid output engine type
HAS_OUTPUT_ENG_MASK valid output engine mask
HAS_CONFORMITY_INDEX_FIX valid conformity index
HAS_LLA_VRP_BASED valid lla vrp based
HAS_ENU_VELOCITY_VRP_BASED valid enu velocity vrp based
HAS_ALTITUDE_TYPE valid altitude type
HAS_REPORT_STATUS valid report status
HAS_INTEGRITY_RISK_USED valid integrity risk
HAS_PROTECT_LEVEL_ALONG_TRACK valid protect level along track

HAS_PROTECT_LEVEL_CROSS_TRACK valid protect level cross track
HAS_PROTECT_LEVEL_VERTICAL valid protect level vertical
HAS_DGNSS_STATION_ID valid dgnssStationId

5.7.2.19 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

5.7.2.20 enum telux::loc::GnssDataValidityType

Enumerator

HAS_JAMMER Jammer Indicator is available
HAS_AGC AGC is available

5.7.2.21 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

5.7.2.22 enum telux::loc::LocReqEngineType

Specifies the type of engine requested for fixes

Enumerator

- LOC_REQ_ENGINE_FUSED_BIT** Indicate that the fused/default position is needed to be reported back for the tracking sessions. The default position is the propagated/aggregated reports from all engines running on the system (e.g.: DR/SPE/PPE) according to QTI algorithm.
- LOC_REQ_ENGINE_SPE_BIT** Indicate that the unmodified SPE position is needed to be reported back for the tracking sessions.
- LOC_REQ_ENGINE_PPE_BIT** Indicate that the unmodified PPE position is needed to be reported back for the tracking sessions.
- LOC_REQ_ENGINE_VPE_BIT** Indicate that the unmodified VPE position is needed to be reported back for the tracking sessions.

5.7.2.23 enum telux::loc::LocationAggregationType

Specifies the type of engine for the reported fixes

Enumerator

- LOC_OUTPUT_ENGINE_FUSED** This is the propagated/aggregated report from the fixes of all engines running on the system (e.g.: DR/SPE/PPE).
- LOC_OUTPUT_ENGINE_SPE** This fix is the unmodified fix from modem GNSS engine
- LOC_OUTPUT_ENGINE_PPE** This is the unmodified fix from PPP engine
- LOC_OUTPUT_ENGINE_VPE** This is the unmodified fix from VPE engine.

5.7.2.24 enum telux::loc::PositioningEngineType

Specifies the type of engine responsible for fixes when the engine type is fused

Enumerator

- STANDARD_POSITIONING_ENGINE**
- DEAD_RECKONING_ENGINE**
- PRECISE_POSITIONING_ENGINE** For precise position engines.
- VP_POSITIONING_ENGINE** For VP position engine.

5.7.2.25 enum telux::loc::LeverArmType

Lever ARM type

Enumerator

- LEVER_ARM_TYPE_GNSS_TO_VRP** Lever arm parameters regarding the VRP (Vehicle Reference Point) w.r.t the origin (at the GNSS Antenna)
- LEVER_ARM_TYPE_DR_IMU_TO_GNSS** Lever arm regarding GNSS Antenna w.r.t the origin at the IMU (inertial measurement unit) for DR (dead reckoning engine)
- LEVER_ARM_TYPE_VEPP_IMU_TO_GNSS** Lever arm regarding GNSS Antenna w.r.t the origin at the IMU (inertial measurement unit) for VEPP (vision enhanced precise positioning engine)

5.7.2.26 enum telux::loc::GnssMeasurementsDataValidityType

Specify valid fields in [GnssMeasurementsData](#).

Enumerator

SV_ID_BIT Validity of svId.
SV_TYPE_BIT Validity of svType.
STATE_BIT Validity of stateMask.
RECEIVED_SV_TIME_BIT Validity of receivedSvTimeNs.
RECEIVED_SV_TIME_UNCERTAINTY_BIT Validity of receivedSvTimeUncertaintyNs.
CARRIER_TO_NOISE_BIT Validity of carrierToNoiseDbHz.
PSEUDORANGE_RATE_BIT Validity of pseudorangeRateMps.
PSEUDORANGE_RATE_UNCERTAINTY_BIT Validity of pseudorangeRateUncertaintyMps.
ADR_STATE_BIT Validity of adrStateMask.
ADR_BIT Validity of adrMeters.
ADR_UNCERTAINTY_BIT Validity of adrUncertaintyMeters.
CARRIER_FREQUENCY_BIT Validity of carrierFrequencyHz.
CARRIER_CYCLES_BIT Validity of carrierCycles.
CARRIER_PHASE_BIT Validity of carrierPhase.
CARRIER_PHASE_UNCERTAINTY_BIT Validity of carrierPhaseUncertainty.
MULTIPATH_INDICATOR_BIT Validity of multipathIndicator.
SIGNAL_TO_NOISE_RATIO_BIT Validity of signalToNoiseRatioDb.
AUTOMATIC_GAIN_CONTROL_BIT Validity of agcLevelDb.
GNSS_SIGNAL_TYPE Validity of signal type.
BASEBAND_CARRIER_TO_NOISE Validity of basebandCarrierToNoise.
FULL_ISB Validity of fullInterSignalBias.
FULL_ISB_UNCERTAINTY Validity of fullInterSignalBiasUncertainty.

5.7.2.27 enum telux::loc::GnssMeasurementsStateValidityType

Specify GNSS measurement state in [GnssMeasurementsData::stateMask](#).

Enumerator

UNKNOWN_BIT State is unknown.
CODE_LOCK_BIT State is "code lock".
BIT_SYNC_BIT State is "bit sync".
SUBFRAME_SYNC_BIT State is "subframe sync".
TOW_DECODED_BIT State is "tow decoded".
MSEC_AMBIGUOUS_BIT State is "msec ambiguous".
SYMBOL_SYNC_BIT State is "symbol sync".
GLO_STRING_SYNC_BIT State is "GLONASS string sync".
GLO_TOD_DECODED_BIT State is "GLONASS TOD decoded".
BDS_D2_BIT_SYNC_BIT State is "BDS D2 bit sync".
BDS_D2_SUBFRAME_SYNC_BIT State is "BDS D2 subframe sync".
GAL_E1BC_CODE_LOCK_BIT State is "Galileo E1BC code lock".
GAL_E1C_2ND_CODE_LOCK_BIT State is "Galileo E1C second code lock".
GAL_E1B_PAGE_SYNC_BIT State is "Galileo E1B page sync".
SBAS_SYNC_BIT State is "SBAS sync".

5.7.2.28 enum telux::loc::GnssMeasurementsAdrStateValidityType

Specify accumulated delta range state in [GnssMeasurementsData::adrStateMask](#).

Enumerator

UNKNOWN_STATE State is unknown.
VALID_BIT State is valid.
RESET_BIT State is "reset".
CYCLE_SLIP_BIT State is "cycle slip".

5.7.2.29 enum telux::loc::GnssMeasurementsMultipathIndicator

Specify the GNSS multipath indicator state in [GnssMeasurementsData::multipathIndicator](#).

Enumerator

UNKNOWN_INDICATOR Multipath indicator is unknown.
PRESENT Multipath indicator is present.
NOT_PRESENT Multipath indicator is not present.

5.7.2.30 enum telux::loc::GnssMeasurementsClockValidityType

Specify the valid fields in [GnssMeasurementsClock](#).

Enumerator

LEAP_SECOND_BIT Validity of leapSecond.
TIME_BIT Validity of timeNs.
TIME_UNCERTAINTY_BIT Validity of timeUncertaintyNs.
FULL_BIAS_BIT Validity of fullBiasNs.
BIAS_BIT Validity of biasNs.
BIAS_UNCERTAINTY_BIT Validity of biasUncertaintyNs.
DRIFT_BIT Validity of driftNsps.
DRIFT_UNCERTAINTY_BIT Validity of driftUncertaintyNsps.
HW_CLOCK_DISCONTINUITY_COUNT_BIT Validity of hwClockDiscontinuityCount.

5.7.2.31 enum telux::loc::LeapSecondInfoValidityType

Specify the valid fields in [LeapSecondInfo](#)

Enumerator

LEAP_SECOND_SYS_INFO_CURRENT_LEAP_SECONDS_BIT Current leap second info is available.
 This info will only be available if the leap second change info is not available. If leap second change info is available, to figure out the current leap second info, compare current gps time with the gps timestamp of leap second change to know whether to choose leapSecondBefore or leapSecondAfter as current leap second.
LEAP_SECOND_SYS_INFO_LEAP_SECOND_CHANGE_BIT The last known leap change event is available. The info can be available on two scenarios: 1: This leap second change event has been scheduled and yet to happen 2: This leap second change event has already happened and next

leap second change event has not yet been scheduled.

5.7.2.32 enum telux::loc::LocationSystemInfoValidityType

Specify the set of valid fields in [LocationSystemInfo](#)

Enumerator

LOCATION_SYS_INFO_LEAP_SECOND contains current leap second or leap second change info

5.7.2.33 enum telux::loc::GnssEnergyConsumedInfoValidityType

Specify the valid fields in [GnssEnergyConsumedInfo](#).

Enumerator

ENERGY_CONSUMED_SINCE_FIRST_BOOT_BIT validity of [GnssEnergyConsumedInfo](#)

5.7.2.34 enum telux::loc::AidingDataType

Specifies the set of aiding data. This is referenced in the deleteAidingData for deleting any aiding data.

Enumerator

AIDING_DATA_EPHEMERIS Mask to delete ephemeris aiding data

AIDING_DATA_DR_SENSOR_CALIBRATION Mask to delete calibration data from dead reckoning position engine

5.7.2.35 enum telux::loc::TerrestrialTechnologyType

Specifies the set of terrestrial technologies.

Enumerator

GTP_WWAN Cell-based technology

5.7.2.36 enum telux::loc::RobustLocationConfigType

Specify the valid mask for robust location configuration used by the GNSS standard position engine (SPE).

Enumerator

VALID_ENABLED Validity of enabled

VALID_ENABLED_FOR_E911 Validity of enabledForE911.

VALID_VERSION Validity of version.

5.7.2.37 enum telux::loc::NmeaSentenceType

Specifies the HLOS generated NMEA sentence types.

Enumerator

GGA GGA NMEA sentence
RMC RMC NMEA sentence
GSA GSA NMEA sentence
VTG VTG NMEA sentence
GNS GNS NMEA sentence
DTM DTM NMEA sentence
GPGLSV GPGLSV NMEA sentence for SVs from GPS constellation
GLGSV GLGSV NMEA sentence for SVs from GLONASS constellation
GAGSV GAGSV NMEA sentence for SVs from GALILEO constellation
GQGSV GQGSV NMEA sentence for SVs from QZSS constellation
GBGSV GBGSV NMEA sentence for SVs from BEIDOU constellation
GIGSV GIGSV NMEA sentence for SVs from NAVIC constellation
ALL All NMEA sentences

5.7.2.38 enum telux::loc::GeodeticDatumType

Specify the Geodetic datum for NMEA sentence types that are generated.

Enumerator

GEODETIC_TYPE_NONE No type
GEODETIC_TYPE_WGS_84 Geodetic datum type to indicate the use of World Geodetic System 1984 (WGS84) system
GEODETIC_TYPE_PZ_90 Geodetic datum type to indicate the use of PZ90/GLONASS system

5.7.2.39 enum telux::loc::DRConfigValidityType

Specify the valid mask for the configuration parameters of dead reckoning position engine

Enumerator

BODY_TO_SENSOR_MOUNT_PARAMS_VALID Validity of body to sensor mount parameters.
VEHICLE_SPEED_SCALE_FACTOR_VALID Validity of vehicle speed scale factor.
VEHICLE_SPEED_SCALE_FACTOR_UNC_VALID Validity of vehicle speed scale factor uncertainty.
GYRO_SCALE_FACTOR_VALID Validity of gyro scale factor.
GYRO_SCALE_FACTOR_UNC_VALID Validity of gyro scale factor uncertainty.

5.7.2.40 enum telux::loc::GnssReportType

Specifies the set of gnss reports.

Enumerator

LOCATION Location reports
SATELLITE_VEHICLE Satellite reports
NMEA Nmea reports
DATA Data reports
MEASUREMENT 1Hz measurement reports

5.7.2.41 enum telux::loc::EngineType

Specify the position engine types

Enumerator

- UNKNOWN** Unknown engine type.
- SPE** Standard GNSS position engine.
- PPE** Precise position engine.
- DRE** Dead reckoning position engine.
- VPE** Vision positioning engine.

5.7.2.42 enum telux::loc::ReportStatus

Specify the status of the report

Enumerator

- UNKNOWN** Report status is unknown.
- SUCCESS** Report status is successful. The engine is able to calculate the desired fix. Most of the fields in [ILocationInfoEx](#) will be valid.
- INTERMEDIATE** Report is still in progress. The engine has not completed its calculations when this report was generated. Accuracy of various fields is non-optimal. Only some of the fields in [ILocationInfoEx](#) will be valid.
- FAILURE** Report status has failed. The engine is not able to calculate the fix. Most of the fields in [ILocationInfoEx](#) will be invalid.

5.7.2.43 enum telux::loc::DebugLogLevel

Specify the logcat debug level during XTRA's param configuration. Currently, only XTRA daemon will support the runtime configuration of the debug log level.

Enumerator

- DEBUG_LOG_LEVEL_NONE** No message is logged.
- DEBUG_LOG_LEVEL_ERROR** Only error level debug messages will get logged.
- DEBUG_LOG_LEVEL_WARNING** Only warning and error level debug messages will get logged.
- DEBUG_LOG_LEVEL_INFO** Only info, warning and error level debug messages will get logged.
- DEBUG_LOG_LEVEL_DEBUG** Only debug, info, warning and error level debug messages will get logged.
- DEBUG_LOG_LEVEL_VERBOSE** Verbose, debug, info, warning and error level debug messages will get logged.

5.7.2.44 enum telux::loc::XtraDataStatus

Provides the status of the previously downloaded Xtra data.

Enumerator

STATUS_UNKNOWN If XTRA feature is disabled or if XTRA feature is enabled, but XTRA daemon has not yet retrieved the assistance data status from modem on early stage of device bootup, xtra data status will be unknown.

STATUS_NOT_AVAIL If XTRA feature is enabled, but XTRA data is not present on the device.

STATUS_NOT_VALID If XTRA feature is enabled, XTRA data has been downloaded ever but no longer valid.

STATUS_VALID If XTRA feature is enabled, XTRA data has been downloaded and is currently valid.

5.7.2.45 enum telux::loc::LocConfigIndicationsType

Enum of all the possible indications invoked by a Location Configurator listener.

Enumerator

LOC_CONF_IND_XTRA_STATUS

5.7.3 Function Documentation**5.7.3.1 virtual bool telux::loc::ILocationConfigurator::isSubsystemReady () [pure virtual]**

Checks the status of location configuration subsystems and returns the result.

Returns

True if location configuration subsystem is ready for service otherwise false.

5.7.3.2 virtual std::future<bool> telux::loc::ILocationConfigurator::onSubsystemReady () [pure virtual]

Wait for location configuration subsystem to be ready.

Returns

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

5.7.3.3 virtual telux::common::Status telux::loc::ILocationConfigurator::configureCTunc (bool enable, telux::common::ResponseCallback callback = nullptr, float timeUncertainty = DEFAULT_TUNC_THRESHOLD, uint32_t energyBudget = DEFAULT_TUNC_ENERGY_THRESHOLD) [pure virtual]

This API enables or disables the constrained time uncertainty(C-TUNC) feature. When the vehicle is turned off this API helps to put constraint on the time uncertainty. For multiple invocations of this API, client should wait for the command to finish, e.g.: via ResponseCallback recieved before issuing a second configureCTunc command.

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.

5.7.3.4 virtual telux::common::Status telux::loc::ILocationConfigurator::configurePACE (bool *enable*, telux::common::ResponseCallback *callback* = *nullptr*) [pure virtual]

This API enables or disables position assisted clock estimator feature. For multiple invocations of this API, client should wait for the command to finish, e.g.: via ResponseCallback recieved before issuing a second configurePACE command.

Parameters

in	<i>enable</i>	- to enable/disable position assisted clock estimator feature.
in	<i>callback</i>	- Optional callback to get the response of enablement/disablement of PACE.

5.7.3.5 virtual telux::common::Status telux::loc::ILocationConfigurator::deleteAllAidingData (telux::common::ResponseCallback *callback* = *nullptr*) [pure virtual]

This API deletes all form of aiding data from all position engines. This API deletes all assistance data used by GPS engine and force engine to do a cold start for next session. Invoking this API will trigger cold start of all position engines on the device. This will cause significant delay for the position engines to produce next fix and may have other performance impact.

Parameters

in	<i>callback</i>	- Optional callback to get the response of delete aiding data.

5.7.3.6 virtual telux::common::Status telux::loc::LocationConfigurator::configureLeverArm (const LeverArmConfigInfo & *info*, telux::common::ResponseCallback *callback* = *nullptr*) [pure virtual]

This API sets the lever arm parameters for the vehicle. For multiple invocations of this API client should wait for the command to finish, e.g.: via ResponseCallback received before issuing a second configureLeverArm command.

Parameters

in	<i>info</i>	- lever arm configuration info regarding below three types of lever arm info: a: GNSS Antenna w.r.t the origin at the IMU (inertial measurement unit) for DR engine b: GNSS Antenna w.r.t the origin at the IMU (inertial measurement unit) for VEPP engine c: VRP (Vehicle Reference Point) w.r.t the origin (at the GNSS Antenna). Vehicle manufacturers prefer the position output to be tied to a specific point in the vehicle rather than where the antenna is placed (midpoint of the rear axle is typical).
in	<i>callback</i>	- Optional callback to get the response of configure lever arm.

5.7.3.7 virtual telux::common::Status telux::loc::LocationConfigurator::configureConstellations (const SvBlackList & *list*, telux::common::ResponseCallback *callback* = *nullptr*, bool *resetToDefault* = *false*) [pure virtual]

This API blacklists some constellations or subset of SVs from the constellation from being used by the GNSS engine on modem. For multiple invocations of this API, client should wait for the command to finish, e.g.: via ResponseCallback received before issuing a second configureConstellations command. This API call is not incremental and the new settings will completely overwrite the previous call. Supported constellations for this API are GLONASS, QZSS, BEIDOU, GALILEO and SBAS. For other constellations NOTSUPPORTED status will be returned. When resetToDefault is false then the list is expected to contain the constellations or SVs that should be blacklisted. An empty list could be specified to allow all constellations/SVs (i.e. none will be blacklisted) in determining the fix. When resetToDefault is set to true, the device will revert to the default list of SV/ constellations to be blacklisted.

Parameters

in	<i>list</i>	- specify the set of constellations and SVs that should not be used by the GNSS engine on modem. Constellations and SVs not specified in blacklistedSvList could get used by the GNSS engine on modem.
in	<i>callback</i>	- Optional callback to get the response of configure constellations.
in	<i>resetToDefault</i>	- when set to true, the device will revert to the default list of SV/constellation to be blacklisted. When set to false, list will be inspected to determine what should be blacklisted.

5.7.3.8 **virtual telux::common::Status telux::loc::ILocationConfigurator::configureRobustLocation (bool *enable*, bool *enableForE911* = *false*, telux::common::ResponseCallback *callback* = *nullptr*) [pure virtual]**

This API enables/disables robust location feature and enables/disables robust location while device is on E911. This API focuses on detection and reporting GNSS spoofing in position, time and navigation data. When this API is enabled it reports confidence of the GNSS spoofing by the `getConformityIndex()` API defined in the [ILocationInfoEx](#) class.

Parameters

in	<i>enable</i>	- true to enable robust location and false to disable robust location.
in	<i>enableForE911</i>	- true to enable robust location when the device is on E911 session and false to disable on E911 session. This parameter is only valid if robust location is enabled.
in	<i>callback</i>	- Optional callback to get the response of configure robust location.

Note

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

5.7.3.9 **virtual telux::common::Status telux::loc::ILocationConfigurator::requestRobustLocation (GetRobustLocationCallback *cb*) [pure virtual]**

This API retrieves the robust location settings used by the GNSS engine.

Parameters

in	<i>cb</i>	- callback to retrieve robust location information.

Returns

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

Note

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

5.7.3.10 **virtual telux::common::Status telux::loc::ILocationConfigurator::configureMinGpsWeek (uint16_t *minGpsWeek*, telux::common::ResponseCallback *callback* = *nullptr*) [pure virtual]**

This API configures the minimum GPS week used by the modem GNSS engine. Client should wait for the command to finish, e.g.: via `ResponseCallback` received before issuing a second `configureMinGpsWeek` command.

<code>in</code>	<code>minGpsWeek</code>	- minimum GPS week to be used by modem GNSS engine.
<code>in</code>	<code>callback</code>	- Optional callback to get the response of configure minimum GPS week.

Returns

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

Note

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

5.7.3.11 **virtual telux::common::Status telux::loc::ILocationConfigurator::requestMinGpsWeek (GetMinGpsWeekCallback *cb*) [pure virtual]**

This API retrieves the minimum GPS week configuration used by the modem GNSS engine.

Parameters

<code>in</code>	<code>cb</code>	- callback to retrieve the minimum gps week.

Returns

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

Note

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

5.7.3.12 **virtual telux::common::Status telux::loc::ILocationConfigurator::configureMinSVElevation (uint8_t *minSVElevation*, telux::common::ResponseCallback *callback* = nullptr) [pure virtual]**

This API configures the minimum SV elevation angle setting used by the GNSS standard position engine. Configuring minimum SV elevation setting will not cause position engine to stop tracking low elevation SVs. This elevation mask is used to filter out the SVs that are used in determining the position. SVs with an elevation below this setting will be excluded from position determination. So configuring this to a large angle will filter out most of the SVs and will have adverse affects on the performance of the position engine.

This setting does not impact the SV information and SV measurement reports retrieved from APIs such as `IGnssSvInfo::getSVInfoList`, [ILocationListener::onGnssMeasurementsInfo](#).

Client should wait for the command to finish, e.g.: via `ResponseCallback` received, before issuing a second `configureMinElevation` command. If this API is called while the GNSS Position Engine is in the middle of a session, `ResponseCallback` will still be invoked shortly after to indicate the setting has been received by the SPE engine. However the engine can take some time to apply this setting if it is in middle of a session (as long as 255 seconds in some implementations). It is advised to use this API with caution and only for very limited usage scenario, e.g.: for performance test, certification process and for one-time device configuration.

in	<i>minSVElevation</i>	- minimum SV elevation to be used by GNSS standard position engine (SPE). Valid range is [0, 90] in unit of degree.
in	<i>callback</i>	- Optional callback to get the response of configure minimum SV Elevation angle.

Returns

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

Note

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

5.7.3.13 **virtual telux::common::Status telux::loc::ILocationConfigurator::requestMinSVElevation (GetMinSVElevationCallback *cb*) [pure virtual]**

This API retrieves the minimum SV Elevation configuration used by the modem GNSS SPE engine. If this API is invoked right after the configureMinSVElevation, the returned setting may not match the one specified in configureMinSVElevation, as the setting received via configureMinSVElevation might not have been applied yet as it takes time to apply the setting if the GNSS SPE engine has an on-going session.

Parameters

in	<i>cb</i>	- callback to retrieve the minimum SV elevation.

Returns

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

Note

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

5.7.3.14 **virtual telux::common::Status telux::loc::ILocationConfigurator::deleteAidingData (AidingData *aidingDataMask*, telux::common::ResponseCallback *callback = nullptr*) [pure virtual]**

This API deletes specified aiding data from all position engines on the device. For example, removing ephemeris data may trigger GNSS engine to do a warm start.

Parameters

in	<i>aidingDataMask</i>	- specify the set of aiding data to be deleted from all position engines.
in	<i>callback</i>	- Optional callback to get the response of delete aiding data.

Returns

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

Note

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

5.7.3.15 **virtual telux::common::Status telux::loc::ILocationConfigurator::configureDR (const DREngineConfiguration & *config*, telux::common::ResponseCallback *callback* = nullptr) [pure virtual]**

This API configures various parameters for dead reckoning position engine. Clients should wait for the command to finish e.g.: via ResponseCallback to be received before issuing a second configureDREngineParams command. Behavior is not defined if client issues a second request of configureDREngineParams without waiting for the completion of the previous configureDREngineParams request.

Parameters

in	<i>config</i>	- specify dead reckoning engine configuration.
in	<i>callback</i>	- Optional callback to get the response of configureDREngineParams.

Returns

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

Note

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

5.7.3.16 **virtual telux::common::Status telux::loc::ILocationConfigurator::configureNmeaTypes (const NmeaSentenceConfig *nmeaType*, telux::common::ResponseCallback *callback* = nullptr) [pure virtual]**

This API is used to configure the NMEA sentence types that clients will receive via [ILocationManager::startDetailedReports](#) or [ILocationManager::startDetailedEngineReports](#). Without prior invocation to this API, all NMEA sentences supported in the system will get generated and delivered to all the clients that register to receive NMEA sentences. The NMEA sentence type configuration is common across all clients and updating it will affect all clients. This API call is not incremental and the new NMEA sentence types will completely overwrite the previous call to this API.

Parameters

in	<i>nmeaType</i>	- specify the set of NMEA sentences
in	<i>callback</i>	- Optional callback to get the response of configureNmeaTypes.

Returns

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

Note

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

5.7.3.17 **virtual telux::common::Status telux::loc::ILocationConfigurator::configureNmea (const NmeaConfig *configParams*, telux::common::ResponseCallback *callback* = *nullptr*) [pure virtual]**

This API is used to configure the NMEA sentences that the clients will receive via [ILocationManager::startDetailedReports](#) or [ILocationManager::startDetailedEngineReports](#). Without prior invocation to this API, all NMEA sentences supported in the system will get generated and delivered to all the clients that register to receive NMEA sentences. The NMEA sentence type configuration is common across all clients and updating it will affect all clients.

Please note that for the NMEA datum type request to be successful, the nmea provider configuration in the GPS configuration file should be set to application processor.

This API call is not incremental and the new NMEA configuration will completely overwrite the previous call to this API.

Parameters

in	<i>configParams</i>	- Configuration Parameters for Nmea on the device.
in	<i>callback</i>	- Optional callback to get the response of configureNmea.

Returns

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

Note

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

5.7.3.18 **virtual telux::common::Status telux::loc::ILocationConfigurator::provideConsentForTerrestrialPositioning (bool *userConsent*, telux::common::ResponseCallback *callback* = *nullptr*) [pure virtual]**

Clients can request Terrestrial Positioning using [ILocationManager::getTerrestrialPosition](#). Terrestrial Positioning requires sending device data to the cloud to get the position. This functionality requires user consent. This API needs to be invoked to provide the user consent.

The consent will remain effective across power cycles, until this API is called with a different value.

in	<i>userConsent</i>	- true indicates user consents to sending device data to cloud, false indicates user does not consent.
in	<i>callback</i>	- Optional callback to get the response of <code>provideConsentForTerrestrialPositioning</code> .

Returns

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

Note

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

5.7.3.19 **virtual telux::common::Status telux::loc::ILocationConfigurator::register-LocationInjector (std::weak_ptr< ILocationInjectionListener > *listener*) [pure virtual]**

This API registers an [ILocationInjectionListener](#) listener and will receive notifications regarding when to start or stop the location report injection using the [injectLocationData](#) API.

Parameters

in	<i>listener</i>	- Pointer of ILocationInjectionListener object.

Returns

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

Note

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

5.7.3.20 **virtual telux::common::Status telux::loc::ILocationConfigurator::deregister-LocationInjector (std::weak_ptr< ILocationInjectionListener > *listener*) [pure virtual]**

This API removes a previously registered listener and will also stop receiving notifications related to location data injection for that particular listener.

Parameters

in	<i>listener</i>	- Pointer of ILocationInjectionListener object.

Returns

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

Note

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

5.7.3.21 **virtual telux::common::Status telux::loc::ILocationConfigurator::inject-LocationData (const ExternalLocationInfo & *location*) [pure virtual]**

Inject best location data

This API is used to inject the [ExternalLocationInfo](#) data into the GNSS engine.

This API is intended to be invoked after [ILocationInjectionListener::onStartInjection](#) API call and is not expected to be invoked after [ILocationInjectionListener::onStopInjection](#) API call. If this API is used without meeting the above mentioned pre-requisites, the GNSS engine might produce low quality GNSS position.

Parameters

in	<i>location</i>	ExternalLocationInfo data to be injected.

Returns

, Status of injectLocationData i.e. success or suitable status code.

Note

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

5.7.3.22 **virtual telux::common::Status telux::loc::ILocationConfigurator::configure-EngineIntegrityRisk (const EngineType *engineType*, uint32_t *integrityRisk*, telux::common::ResponseCallback *callback* = nullptr) [pure virtual]**

This API is used to instruct the specified engine to use the provided integrity risk level for protection level calculation in position report. This API can be called when a position session is in progress. Prior to calling this API for a particular engine, the engine shall not calculate the protection levels and shall not include the protection levels in its position report. The implementation might not support protection levels across all engines. For engines that don't support it, ResponseCallback will get invoked with ErrorCode::NOT_SUPPORTED.

Parameters

in	<i>engineType</i>	- the engine that is instructed to use the specified integrity risk level for protection level calculation.
in	<i>integrityRisk</i>	- the integrity risk level used for calculating protection level. The integrity risk is defined as a probability per epoch, in unit of 2.5e-10. The valid range for actual integrity is [2.5e-10, 1-2.5e-10]), this corresponds to range of [1,4e9-1] of this parameter.

<i>in</i>	<i>callback</i>	- Optional callback to get the response of <code>configureEngineIntegrityRisk</code> .

Returns

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

Note

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

5.7.3.23 **virtual telux::common::Status telux::loc::ILocationConfigurator::configureXtraParams (bool *enable*, const XtraConfig *configParams*, telux::common::ResponseCallback *callback* = nullptr) [pure virtual]**

This API is used to enable/disable the XTRA (Predicted GNSS Satellite Orbit Data) feature on device. If XTRA feature is to be enabled, this API is also used to configure the various XTRA settings in device.

Clients need to note the below-

1. Wait for the ongoing request to finish prior to the next invocation else the behavior is undefined.
2. The API is non-incremental i.e, the second call will overwrite the first call. Also the configured XTRA params will be persistent.

Parameters

<i>in</i>	<i>enable</i>	- Enable XTRA Feature on the device. False would disable both the XTRA Assistance Data and NTP Time Download.
<i>in</i>	<i>configParams</i>	- Configuration Parameters for XTRA on the device.
<i>in</i>	<i>callback</i>	- Optional callback stating the response errorcode.

Note

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

5.7.3.24 **virtual telux::common::Status telux::loc::ILocationConfigurator::requestXtraStatus (GetXtraStatusCallback *callback*) [pure virtual]**

This API is used to query xtra feature setting and xtra assistance data status used by the GNSS standard position engine (SPE). If `XTRA_DATA_STATUS_UNKNOWN` is returned but XTRA feature is enabled, the client shall wait a few seconds before calling this API again.

Parameters

<i>in</i>	<i>callback</i>	- Callback to get the Xtra data status information.

Note

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

5.7.3.25 **virtual telux::common::Status telux::loc::ILocationConfigurator::register-Listener (LocConfigIndications *indicationList*, std::weak_ptr< ILocationConfigListener > *listener*) [pure virtual]**

This API is used to register a configuration listener for getting specific indications/updates.

Parameters

in	<i>indicationList</i>	- List of indications client wants to register under telux::loc::LocConfigIndicationsType .
in	<i>listener</i>	- Pointer of ILocationConfigListener object.

Note

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

5.7.3.26 **virtual telux::common::Status telux::loc::ILocationConfigurator::de-RegisterListener (LocConfigIndications *indicationList*, std::weak_ptr< ILocationConfigListener > *listener*) [pure virtual]**

This API is used to deregister a configuration listener from specific indications/updates.

Parameters

in	<i>indicationList</i>	- List of indications client wants to deregister from under telux::loc::LocConfigIndicationsType .
in	<i>listener</i>	- Pointer of ILocationConfigListener object.

Note

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

5.7.3.27 **virtual telux::loc::ILocationConfigurator::~~ILocationConfigurator () [virtual]**

Destructor of [ILocationConfigurator](#)

5.7.3.28 **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.

5.7.3.29 **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.

5.7.3.30 **virtual telux::common::Status telux::loc::ILocationManager::register-ListenerEx (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 ILocationListener object that processes the notification.

Returns

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

5.7.3.31 **virtual telux::common::Status telux::loc::ILocationManager::deRegister-ListenerEx (std::weak_ptr< ILocationListener > *listener*) [pure virtual]**

Remove a previously registered listener.

Parameters

in	<i>listener</i>	- Previously registered ILocationListener that needs to be removed.

Returns

Status of removeListener success or suitable status code

5.7.3.32 **virtual telux::common::Status telux::loc::ILocationManager::startDetailed-Reports (uint32_t *interval*, telux::common::ResponseCallback *callback* = nullptr, GnssReportTypeMask *reportMask* = DEFAULT_GNSS_REPORT) [pure virtual]**

Starts the richer location reports by configuring the time between them as the interval. Any of the 3 APIs that is startDetailedReports or startDetailedEngineReports or startBasicReports can be called one after the other irrespective of order, without calling stopReports in between any of them and the API which is called last will be honored for providing the callbacks. If multiple clients invoke this API with different interval, then all the clients will be benefited with interval which is smallest among all the intervals. Calling this Api will result in [ILocationListener::onDetailedLocationUpdate](#), [ILocationListener::onGnssSVInfo](#), [ILocationListener::onGnssSignalInfo](#), [ILocationListener::onGnssNmeaInfo](#) and [ILocationListener::onGnssMeasurementsInfo](#) APIs on the listener being invoked, assuming they have not been disabled using the GnssReportTypeMask. If a client issues second request to this API then new request for GnssReportTypeMask will over write the previous call to this API.

Parameters

in	<i>interval</i>	- Minimum time interval between two consecutive reports in milliseconds.

E.g. If minInterval is 1000 milliseconds, reports will be provided with a periodicity of 1 second or more depending on the number of applications listening to location updates.

Parameters

in	<i>callback</i>	- Optional callback to get the response of set minimum interval for reports.
in	<i>reportMask</i>	- Optional field to specify which reports a client is interested in. By default all the reports will be enabled.

Returns

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

5.7.3.33 **virtual telux::common::Status telux::loc::ILocationManager::start-DetailedEngineReports (uint32_t *interval*, LocReqEngine *engineType*, telux::common::ResponseCallback *callback* = nullptr, GnssReportTypeMask *reportMask* = DEFAULT_GNSS_REPORT) [pure virtual]**

Starts a session which may provide richer default combined position reports and position reports from other engines. The fused position report type will always be supported if at least one engine in the system is producing valid report. Any of the 3 APIs that is startDetailedReports or startDetailedEngineReports or startBasicReports can be called one after the other irrespective of order, without calling stopReports in

between any of them and the API which is called last will be honored for providing the callbacks. If multiple clients invoke this API with different interval, then all the clients will be benefited with interval which is smallest among all the intervals. Calling this Api will result in [ILocationListener::onDetailedEngineLocationUpdate](#), [ILocationListener::onGnssSVInfo](#), [ILocationListener::onGnssSignalInfo](#), [ILocationListener::onGnssNmeaInfo](#) and [ILocationListener::onGnssMeasurementsInfo](#) APIs on the listener being invoked, assuming they have not been disabled using the GnssReportTypeMask. If a client issues second request to this API then new request for GnssReportTypeMask will over write the previous call to this API.

Parameters

in	<i>interval</i>	- Minimum time interval between two consecutive reports in milliseconds.

E.g. If minInterval is 1000 milliseconds, reports will be provided with a periodicity of 1 second or more depending on the number of applications listening to location updates.

Parameters

in	<i>engineType</i>	- The type of engine requested for fixes such as SPE or PPE or FUSED. The FUSED includes all the engines that are running to generate the fixes such as reports from SPE, PPE and DRE.
in	<i>callback</i>	- Optional callback to get the response of set minimum interval for reports.
in	<i>reportMask</i>	- Optional field to specify which reports a client is interested in. By default all the reports will be enabled.

Returns

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

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

Starts the Location report by configuring the time and distance between the consecutive reports. Any of the 3 APIs that is startDetailedReports or startDetailedEngineReports or startBasicReports can be called one after the other irrespective of order, without calling stopReports in between any of them and the API which is called last will be honored for providing the callbacks. If multiple clients invoke this API with different interval, then all the clients will be benefited with interval which is smallest among all the intervals.

This Api enables the onBasicLocationUpdate Api on the listener.

Parameters

in	<i>distanceInMeters</i>	- distanceInMeters between two consecutive reports in meters. intervalInMs - Minimum time interval between two consecutive reports in milliseconds.

E.g. If `intervalInMs` is 1000 milliseconds and `distanceInMeters` is 100m, reports will be provided according to the condition that happens first. So we need to provide both the parameters for evaluating the report.

The underlying system may have a minimum distance threshold(e.g. 1 meter). Effective distance will not be smaller than this lower bound.

The effective distance may have a granularity level higher than 1 m, e.g. 5 m. So `distanceInMeters` being 59 may be honored at 60 m, depending on the system.

Where there is another application in the system having a session with shorter distance, this client may benefit and receive reports at that distance.

Parameters

<i>in</i>	<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.

5.7.3.35 **virtual telux::common::Status telux::loc::ILocationManager::registerFor-SystemInfoUpdates (`std::weak_ptr< ILocationSystemInfoListener > listener`, `telux::common::ResponseCallback callback = nullptr`) [pure virtual]**

This API registers a [ILocationSystemInfoListener](#) listener and will receive information related to location system that are not tied with location fix session, e.g.: next leap second event. The `startBasicReports`, `startDetailedReports`, `startDetailedEngineReports` does not need to be called before calling this API, in order to receive updates.

Parameters

<i>in</i>	<i>listener</i>	- Pointer of ILocationSystemInfoListener object.
<i>in</i>	<i>callback</i>	- Optional callback to get the response of location system info.

Returns

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

Note

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

5.7.3.36 **virtual telux::common::Status telux::loc::ILocationManager::deRegister-ForSystemInfoUpdates (std::weak_ptr< ILocationSystemInfoListener > *listener*, telux::common::ResponseCallback *callback* = nullptr) [pure virtual]**

This API removes a previously registered listener and will also stop receiving informations related to location system for that particular listener.

Parameters

in	<i>listener</i>	- Previously registered ILocationSystemInfoListener that needs to be removed.
in	<i>callback</i>	- Optional callback to get the response of location system info.

Returns

Status of deRegisterForSystemInfoUpdates success or suitable status code.

Note

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

5.7.3.37 **virtual telux::common::Status telux::loc::ILocationManager::request-EnergyConsumedInfo (GetEnergyConsumedCallback *cb*) [pure virtual]**

This API receives information on energy consumed by modem GNSS engine. If this API is called on this object while this is already a pending request, then it will overwrite the callback to be invoked and the callback from the previous invocation will not be called.

Parameters

in	<i>cb</i>	- callback to get the information of Gnss energy consumed.

Returns

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

Note

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

5.7.3.38 **virtual telux::common::Status telux::loc::ILocationManager::stopReports (telux::common::ResponseCallback *callback* = *nullptr*) [pure virtual]**

This API will stop reports started using startDetailedReports or startBasicReports or registerListener or setMinIntervalForReports.

Parameters

in	<i>callback</i>	- Optional callback to get the response of stop reports.

Returns

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

5.7.3.39 **virtual telux::common::Status telux::loc::ILocationManager::getTerrestrial-Position (uint32_t *timeoutMsec*, TerrestrialTechnology *techMask*, GetTerrestrialInfoCallback *cb*, telux::common::ResponseCallback *callback* = *nullptr*) [pure virtual]**

This API retrieves single-shot terrestrial position using the set of specified terrestrial technologies. This API can be invoked even while there is an on-going tracking session that was started using startBasicReports/startDetailedReports/startDetailedEngineReports. If this API is invoked while there is already a pending request for terrestrial position, the request will fail and ResponseCallback will get invoked with ErrorCode::OP_IN_PROGRESS. To cancel a pending request, use [ILocationManager::cancelTerrestrialPositionRequest](#). Before using this API, user consent needs to be set true via ILocationConfigurator::provideConsentForTerrestrialPositioning.

Parameters

in	<i>timeoutMsec</i>	- the time in milliseconds within which the client is expecting a response. If the system is unable to provide a report within this time, the ResponseCallback will be invoked with ErrorCode::OPERATION_TIMEOUT.
in	<i>techMask</i>	- the set of terrestrial technologies that are allowed to be used for producing the position.
in	<i>cb</i>	- callback to receive terrestrial position. This callback will only be invoked when ResponseCallback is invoked with SUCCESS.
in	<i>callback</i>	- Optional callback to get the response of getTerrestrialPosition.

Returns

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

Note

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

5.7.3.40 **virtual telux::common::Status telux::loc::ILocationManager::cancelTerrestrialPositionRequest (telux::common::ResponseCallback *callback* = *nullptr*) [pure virtual]**

This API cancels the pending request invoked by [ILocationManager::getTerrestrialPosition](#). If this API is invoked while there is no pending request for terrestrial position from [ILocationManager::getTerrestrialPosition](#), then ResponseCallback will be invoked with ErrorCode::INVALID_ARGUMENTS.

Parameters

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

Returns

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

Note

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

5.7.3.41 **virtual telux::loc::ILocationManager::~~ILocationManager () [virtual]**

Destructor of [ILocationManager](#)

5.7.4 Variable Documentation

5.7.4.1 **const float telux::loc::UNKNOWN_CARRIER_FREQ = -1**

5.7.4.2 **const int telux::loc::UNKNOWN_SIGNAL_MASK = 0**

5.7.4.3 **const uint64_t telux::loc::UNKNOWN_TIMESTAMP = 0**

5.7.4.4 **const float telux::loc::DEFAULT_TUNC_THRESHOLD = 0.0**

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

5.7.4.5 **const int telux::loc::DEFAULT_TUNC_ENERGY_THRESHOLD = 0**

Default value for energy consumed of time uncertainty. The default here means that the engine is allowed to use infinite power. Units: 100 micro watt second.

5.7.4.6 **const uint32_t telux::loc::DEFAULT_GNSS_REPORT = 0xffffffff**

0xffffffff indicates all the reports are enabled.

5.8 Data Services

This section contains APIs related to Cellular Data Services.

5.8.1 Define Documentation

5.8.1.1 #define PROFILE_ID_MAX 0x7FFFFFFF

Default data profile id.

5.8.1.2 #define MAX_QOS_FILTERS 16

Max filters in one flow

5.8.1.3 #define IP_PROT_UNKNOWN 0xFF

Default IP Protocol number in IPv4 or IPv6 headers.

5.8.2 Data Structure Documentation

5.8.2.1 class telux::data::IDataConnectionManager

[IDataConnectionManager](#) is a primary interface for cellular connectivity. This interface provides APIs for start and stop data call connections, get data call information and listener for monitoring data calls. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready.

Public member functions

- virtual [telux::common::ServiceStatus](#) [getServiceStatus](#) ()=0
- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status](#) [setDefaultProfile](#) ([OperationType](#) oprType, uint8_t profileId, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [startDataCall](#) (int profileId, [IpFamilyType](#) ipFamilyType=[IpFamilyType::IPV4V6](#), [DataCallResponseCb](#) callback=nullptr, [OperationType](#) operationType=[OperationType::DATA_LOCAL](#), std::string interfaceName="")=0
- virtual [telux::common::Status](#) [stopDataCall](#) (int profileId, [IpFamilyType](#) ipFamilyType=[IpFamilyType::IPV4V6](#), [DataCallResponseCb](#) callback=nullptr, [OperationType](#) operationType=[OperationType::DATA_LOCAL](#))=0
- virtual [telux::common::Status](#) [stopDataCall](#) (std::string interfaceName, [telux::common::ResponseCallback](#) callback=nullptr, [OperationType](#) operationType=[OperationType::DATA_LOCAL](#))=0
- virtual [telux::common::Status](#) [registerListener](#) (std::weak_ptr< [IDataConnectionListener](#) > listener)=0

- virtual `telux::common::Status deregisterListener (std::weak_ptr< IDataConnectionListener > listener)=0`
- virtual `int getSlotId ()=0`
- virtual `telux::common::Status requestDataCallList (OperationType type, DataCallListResponseCb callback)=0`
- virtual `~IDataConnectionManager ()`

5.8.2.1.1 Constructors and Destructors

5.8.2.1.1.1 `virtual telux::data::IDataConnectionManager::~IDataConnectionManager ( ) [virtual]`

Destructor for `IDataConnectionManager`

5.8.2.1.2 Member Function Documentation

5.8.2.1.2.1 `virtual telux::common::ServiceStatus telux::data::IDataConnectionManager::getServiceStatus ( ) [pure virtual]`

Returns current initialization status of data connection manager.

Returns

`SERVICE_AVAILABLE` If data connection manager is ready for service.
`SERVICE_UNAVAILABLE` If data connection manager is temporarily unavailable.
`SERVICE_FAILED` If data connection manager encountered an irrecoverable failure.

5.8.2.1.2.2 `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.

5.8.2.1.2.3 `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.

5.8.2.1.2.4 `virtual telux::common::Status telux::data::IDataConnectionManager::setDefaultProfile ( OperationType oprType, uint8_t profileId, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Set a profile as default which results in the following:

- Default routes in the system will route traffic over the network interface associated with this profile.
- Bridge 0 will be associated with traffic from this profile.

Parameters

in	<i>operationType</i>	telux::data::OperationType
in	<i>profileId</i>	Profile identifier to be associated with default handler
in	<i>callback</i>	optional callback to get the response setDefaultProfile

Returns

Immediate status of setDefaultProfile i.e. success or suitable status.

Note

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

5.8.2.1.2.5 `virtual telux::common::Status telux::data::IDataConnectionManager::startDataCall ( int profileId, IpFamilyType ipFamilyType = IpFamilyType::IPV4V6, DataCallResponseCb callback = nullptr, OperationType operationType = OperationType::DATA_LOCAL, std::string interfaceName = "" ) [pure virtual]`

Starts a data call corresponding to default or specified profile identifier.

This will bring up data call connection based on specified profile identifier, IP family type, and operation type (local/remote). Clients can also specify network interface name that will be associated with data call to be started. This is an asynchronous API. If `telux::common::Status::SUCCESS` is returned, client provided callback will be invoked at later time with error code and DataCall object associated with requested call. Clients might receive additional notification for the final data call status. For details see [telux::data::DataCallResponseCb](#).

Note

if application starts data call on IPV4V6 then it's expected to stop the data call on same ip family type (i.e IPV4V6).

Parameters

in	<i>profileId</i>	Profile identifier corresponding to which data call bring up will be done. Use IDataProfileManager::requestProfileList 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 telux::data::OperationType

in	<i>interfaceName</i>	Optional name to be assigned to the network interface representing this data call. This option is not supported on all platforms. See documentation of telux::data::DataCallResponseCb for error returned on such platforms.

Returns

Immediate status of [startDataCall\(\)](#) request sent i.e. success or suitable status code.

Note

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

5.8.2.1.2.6 `virtual telux::common::Status telux::data::IDataConnectionManager::stopDataCall ( int profileId, IpFamilyType ipFamilyType = IpFamilyType::IPV4V6, DataCallResponseCb callback = nullptr, OperationType operationType = OperationType::DATA_LOCAL ) [pure virtual]`

Stops a data call corresponding to default or specified profile identifier.

This will tear down specific data call connection based on profile identifier.

Note

If application starts data call on IPV4V6 then it's expected to stop the data call on same ip family type (i.e IPV4V6). Client can only stop data call it started.

Parameters

in	<i>profileId</i>	Profile identifier corresponding to which data call tear down will be done. Use data profile manager to get the list of available profiles.
in	<i>ipFamilyType</i>	Identifies IP family type
out	<i>callback</i>	Optional callback to get the response of stop data call
in	<i>operationType</i>	Optional telux::data::OperationType

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.

5.8.2.1.2.7 `virtual telux::common::Status telux::data::IDataConnectionManager::stopDataCall ( std::string interfaceName, telux::common::ResponseCallback callback = nullptr, OperationType operationType = OperationType::DATA_LOCAL ) [pure virtual]`

Tear down data call connection based on network interface name associated with data call. This is an asynchronous API. If `telux::common::Status::SUCCESS` is returned, client provided callback will be invoked at later time with error code and DataCall object associated with requested call. Clients might receive additional notification for the final data call status. For details see [telux::data::DataCallResponseCb](#).

Note

This API will tear down data calls accross all IP families associated with the provided network interface. If application starts data call on IPV4V6 then it's expected to stop both IPv4 and IPv6 ip families when this API is called.

Parameters

in	<i>interfaceName</i>	network interface name associated with data call to be torn down
out	<i>callback</i>	Optional callback to get the response of stop data call
in	<i>operationType</i>	Optional telux::data::OperationType

Returns

Immediate status of [stopDataCall\(\)](#) request sent i.e. success or suitable status code. The client receives asynchronous notifications indicating the data call tear-down.

5.8.2.1.2.8 `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 IDataConnectionListener object that processes the notification

Returns

Status of registerListener success or suitable status code

5.8.2.1.2.9 `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 IDataConnectionListener object that needs to be removed

Returns

Status of deregisterListener success or suitable status code

5.8.2.1.2.10 virtual int telux::data::IDataConnectionManager::getSlotId () [pure virtual]

Get associated slot id for the Data Connection Manager.

Returns

SlotId

5.8.2.1.2.11 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>	telux::data::OperationType
out	<i>callback</i>	Callback with list of supported data calls

Note

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

5.8.2.2 class telux::data::IDataCall

Represents single established data call on the device.

Public member functions

- virtual const std::string & [getInterfaceName](#) ()=0
- virtual [DataBearerTechnology](#) [getCurrentBearerTech](#) ()=0
- virtual [DataCallEndReason](#) [getDataCallEndReason](#) ()=0
- virtual [DataCallStatus](#) [getDataCallStatus](#) ()=0
- virtual [IpFamilyInfo](#) [getIpv4Info](#) ()=0
- virtual [IpFamilyInfo](#) [getIpv6Info](#) ()=0
- virtual [TechPreference](#) [getTechPreference](#) ()=0
- virtual std::list< [IpAddrInfo](#) > [getIpAddressInfo](#) ()=0

- virtual [IpFamilyType](#) getIpFamilyType ()=0
- virtual int [getProfileId](#) ()=0
- virtual [OperationType](#) getOperationType ()=0
- virtual [telux::common::Status](#) requestTrafficFlowTemplate ([IpFamilyType](#) ipFamilyType, [TrafficFlowTemplateCb](#) callback)=0
- virtual [telux::common::Status](#) requestDataCallStatistics ([StatisticsResponseCb](#) callback=nullptr)=0
- virtual [telux::common::Status](#) resetDataCallStatistics ([telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [~IDataCall](#) ()

5.8.2.2.1 Constructors and Destructors

5.8.2.2.1.1 virtual [telux::data::IDataCall::~IDataCall](#) () [[virtual](#)]

Destructor for [IDataCall](#)

5.8.2.2.2 Member Function Documentation

5.8.2.2.2.1 virtual const std::string& [telux::data::IDataCall::getInterfaceName](#) () [[pure virtual](#)]

Get interface name for the data call associated.

Returns

Interface Name.

5.8.2.2.2.2 virtual [DataBearerTechnology](#) [telux::data::IDataCall::getCurrentBearerTech](#) () [[pure virtual](#)]

Get the bearer technology on which earlier data call was brought up like LTE, WCDMA and etc. This is synchronous API called by client to get bearer technology corresponding to data call.

Returns

[DataBearerTechnology](#)

5.8.2.2.2.3 virtual [DataCallEndReason](#) [telux::data::IDataCall::getDataCallEndReason](#) () [[pure virtual](#)]

Get failure reason for the data call.

Returns

[DataCallFailReason](#).

5.8.2.2.2.4 virtual DataCallStatus telux::data::IDataCall::getDataCallStatus () [pure virtual]

Get data call status like connected, disconnected and IP address changes.

Returns

[DataCallStatus](#).

5.8.2.2.2.5 virtual IpFamilyInfo telux::data::IDataCall::getIpv4Info () [pure virtual]

Get IPv4 Family info like connected, disconnected and IP address changes.

Returns

[IpFamilyInfo](#).

5.8.2.2.2.6 virtual IpFamilyInfo telux::data::IDataCall::getIpv6Info () [pure virtual]

Get IPv6 Family info like connected, disconnected and IP address changes.

Returns

[IpFamilyInfo](#).

5.8.2.2.2.7 virtual TechPreference telux::data::IDataCall::getTechPreference () [pure virtual]

Get the technology on which the call was brought up.

Returns

[TechPreference](#).

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

Get list of IP address information.

Returns

List of IP address details.

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

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

Returns

[IpFamilyType](#).

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

Get Profile Id

Returns

Profile Identifier.

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

Get data operation used for the DataCall.

Returns

[OperationType](#)

5.8.2.2.2.12 virtual telux::common::Status telux::data::IDataCall::requestTrafficFlowTemplate (IpFamilyType *ipFamilyType*, TrafficFlowTemplateCb *callback*) [pure virtual]

Get the current installed QOS Traffic flow template information.

Parameters

in	<i>ipFamilyType</i>	- IP Family type IpFamilyType . TFT's are installed per IP Family.
in	<i>callback</i>	- callback function to get the result of API.

Returns

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

Note

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

5.8.2.2.2.13 virtual telux::common::Status telux::data::IDataCall::requestDataCallStatistics (StatisticsResponseCb *callback = nullptr*) [pure virtual]

Request the data transfer statistics for data call corresponding to specified profile identifier.

Parameters

in	<i>callback</i>	Optional callback to get the response of request Data Call Statistics

Returns

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

5.8.2.2.2.14 virtual telux::common::Status telux::data::IDataCall::resetDataCallStatistics (telux::common::ResponseCallback *callback* = nullptr) [pure virtual]

Reset data transfer statistics for data call corresponding to specified profile identifier.

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 connection listener object. Client needs to implement this interface to get access to data services notifications like onDataCall, onDataCallStatusChanged and onDataCallFailure.

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

Public member functions

- virtual void [onDataCallInfoChanged](#) (const std::shared_ptr< [IDataCall](#) > &dataCall)
- virtual void [onTrafficFlowTemplateChange](#) (const std::shared_ptr< [IDataCall](#) > &dataCall, const std::vector< std::shared_ptr< [TftChangeInfo](#) >> &tft)
- virtual void [onHwAccelerationChanged](#) (const [ServiceState](#) state)
- virtual [~IDataConnectionListener](#) ()

5.8.2.3.1 Constructors and Destructors

5.8.2.3.1.1 virtual telux::data::IDataConnectionListener::~~IDataConnectionListener () [virtual]

Destructor for [IDataConnectionListener](#)

5.8.2.3.2 Member Function Documentation

5.8.2.3.2.1 virtual void telux::data::IDataConnectionListener::onDataCallInfoChanged (const std::shared_ptr< [IDataCall](#) > & *dataCall*) [virtual]

This function is called when there is a change in the data call.

Parameters

in	<i>dataCall</i>	Pointer to IDataCall

5.8.2.3.2.2 `virtual void telux::data::IDataConnectionListener::onTrafficFlowTemplateChange ( const std::shared_ptr< IDataCall > & dataCall, const std::vector< std::shared_ptr< TftChangeInfo >> & tft ) [virtual]`

This function is called when the TFT’s parameters are changed for a packet data session.

Parameters

in	<i>dataCall</i>	Pointer to IDataCall
in	<i>tft</i>	vector of TftChangeInfo info TftChangeInfo

Note

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

5.8.2.3.2.3 `virtual void telux::data::IDataConnectionListener::onHwAccelerationChanged ( const ServiceState state ) [virtual]`

This function is called when a change occur in hardware acceleration service.

Parameters

in	<i>state</i>	New state of hardware Acceleration service (Active/Inactive)

5.8.2.4 struct telux::data::DataRestrictMode

Defines the supported powersave filtering mode and autoexit for the packet data session. DataRestrictFilter

Data fields

Type	Field	Description
DataRestrict-ModeType	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.
DataRestrict-ModeType	filterAutoExit	Disable or enable autoexit feature. When enabled, once an incoming packet matching the filter is received, filter mode will be disabled automatically and any packet will be allowed to be forwarded from modem to apps.

5.8.2.5 struct telux::data::PortInfo

Used to define the Port number and range (number of ports following port value) Ex- for ports ranging from 1000-3000 port = 1000 and range= 2000

for single port 5000 port = 5000 and range= 0

Data fields

Type	Field	Description
uint16_t	port	Port.
uint16_t	range	Range.

5.8.2.6 struct telux::data::ProfileParams

Profile Parameters used for profile creation, query and modify

Data fields

Type	Field	Description
string	profileName	Profile Name
string	apn	APN name
string	userName	APN user name (if any)
string	password	APN password (if any)
TechPreference	techPref	Technology preference, default is TechPreference::UNKNOWN
AuthProtocol-Type	authType	Authentication protocol type, default is AuthProtocolType::AUTH_NONE
IpFamilyType	ipFamilyType	Preferred IP family for the call, default is IpFamilyType::UNKNOWN
ApnTypes	apnTypes	APN Types ref

5.8.2.7 struct telux::data::DataCallStats

Data transfer statistics structure.

Data fields

Type	Field	Description
uint64_t	packetsTx	Number of packets transmitted
uint64_t	packetsRx	Number of packets received
uint64_t	bytesTx	Number of bytes transmitted
uint64_t	bytesRx	Number of bytes received
uint64_t	packets-DroppedTx	Number of transmit packets dropped

Type	Field	Description
uint64_t	packets-DroppedRx	Number of receive packets dropped

5.8.2.8 struct telux::data::IpAddrInfo

IP address information structure

Data fields

Type	Field	Description
string	ifAddress	Interface IP address.
unsigned int	ifMask	Subnet mask.
string	gwAddress	Gateway IP address.
unsigned int	gwMask	Subnet mask.
string	primaryDns-Address	Primary DNS address.
string	secondaryDns-Address	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
EndReason-Type	type	Data call terminated due to reason type, default is CE_UNKNOWN
union DataCall-EndReason	__unnamed__	

5.8.2.10 struct telux::data::LatencyConfig

Latency configuration

Data fields

Type	Field	Description
LatencyLevel	uplink	

5.8.2.11 struct telux::data::BackhaulInfo

Encapsulate backhaul configuration parameters

Data fields

Type	Field	Description
BackhaulType	backhaul	
SlotId	slotId	Backhaul type to apply configuration on.
int	profileId	Slot Id which has sim that contains profile id Needed only for WWAN backhaul

5.8.2.12 struct telux::data::VlanConfig

Structure for vlan configuration

Data fields

Type	Field	Description
InterfaceType	iface	PHY interfaces (i.e. ETH, ECM and RNDIS)
int16_t	vlanId	Vlan identifier (i.e 1-4094)
bool	isAccelerated	is acceleration allowed

5.8.2.13 struct telux::data::FlowDataRate

QOS Flow data min max rate bits per seconds

Data fields

Type	Field	Description
uint64_t	maxRate	QOS Flow maximum data rate
uint64_t	minRate	QOS Flow minimum data rate

5.8.2.14 struct telux::data::QosIPFlowInfo

QOS Flow IP info

Data fields

Type	Field	Description
QosIPFlow-Mask	mask	Valid parameters of QosIPFlowInfo ref
IpTrafficClass-Type	tfClass	IP Traffic class type IpFamilyType
FlowDataRate	dataRate	Flow data rate FlowDataRate

5.8.2.15 class telux::data::DataFactory

[DataFactory](#) is the central factory to create all data classes.

Public member functions

- std::shared_ptr
< [IDataConnectionManager](#) > [getDataConnectionManager](#) (int slotId=DEFAULT_SLOT_ID)
- std::shared_ptr
< [IDataProfileManager](#) > [getDataProfileManager](#) (int slotId=DEFAULT_SLOT_ID)
- std::shared_ptr
< [IDataFilterManager](#) > [getDataFilterManager](#) (int slotId=DEFAULT_SLOT_ID)
- std::shared_ptr
< [IServingSystemManager](#) > [getServingSystemManager](#) (SlotId slotId=DEFAULT_SLOT_ID)
- std::shared_ptr
< [telux::data::net::INatManager](#) > [getNatManager](#) (telux::data::OperationType oprType)
- std::shared_ptr
< [telux::data::net::IFirewallManager](#) > [getFirewallManager](#) (telux::data::OperationType oprType)
- std::shared_ptr
< [telux::data::net::IFirewallEntry](#) > [getNewFirewallEntry](#) (IpProtocol proto, [Direction](#) direction, [IpFamilyType](#) ipFamilyType)
- std::shared_ptr< [IipFilter](#) > [getNewIpFilter](#) (IpProtocol proto)
- std::shared_ptr
< [telux::data::net::IVlanManager](#) > [getVlanManager](#) (telux::data::OperationType oprType)
- std::shared_ptr
< [telux::data::net::ISocksManager](#) > [getSocksManager](#) (telux::data::OperationType oprType)
- std::shared_ptr
< [telux::data::net::IBridgeManager](#) > [getBridgeManager](#) ()
- std::shared_ptr
< [telux::data::net::IL2tpManager](#) > [getL2tpManager](#) ()
- virtual std::shared_ptr
< [telux::data::IDataSettingsManager](#) > [getDataSettingsManager](#) (telux::data::OperationType oprType)

Static Public Member Functions

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

5.8.2.15.1 Member Function Documentation**5.8.2.15.1.1 static DataFactory& telux::data::DataFactory::getInstance () [static]**

Get Data Factory instance.

5.8.2.15.1.2 `std::shared_ptr<IDataConnectionManager> telux::data::DataFactory::getDataConnectionManager ( int slotId = DEFAULT_SLOT_ID )`

Get Data Connection Manager

Parameters

in	slotId	Unique identifier for the SIM slot

Returns

instance of [IDataConnectionManager](#)

5.8.2.15.1.3 `std::shared_ptr<IDataProfileManager> telux::data::DataFactory::getDataProfileManager ( int slotId = DEFAULT_SLOT_ID )`

Get Data Profile Manager

Parameters

in	slotId	Unique identifier for the SIM slot

Returns

instance of [IDataProfileManager](#)

5.8.2.15.1.4 `std::shared_ptr<IDataFilterManager> telux::data::DataFactory::getDataFilterManager ( int slotId = DEFAULT_SLOT_ID )`

Get Data Filter Manager instance

Parameters

in	slotId	Unique identifier for the SIM slot

Returns

instance of [IDataFilterManager](#).

Note

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

5.8.2.15.1.5 `std::shared_ptr<IServingSystemManager> telux::data::DataFactory::getServingSystemManager ( SlotId slotId = DEFAULT_SLOT_ID )`

Get Serving System Manager

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

Returns

instance of [IServingSystemManager](#)

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

Get Network Address Translation(NAT) Manager

Parameters

<i>in</i>	<i>oprType</i>	Required operation type telux::data::OperationType

Returns

instance of [INatManager](#)

Note

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

5.8.2.15.1.7 `std::shared_ptr<telux::data::net::IFirewallManager> telux::data::DataFactory::getFirewall-Manager ( telux::data::OperationType oprType )`

Get Firewall Manager

Parameters

<i>in</i>	<i>oprType</i>	Required operation type telux::data::OperationType

Returns

instance of [IFirewallManager](#)

Note

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

5.8.2.15.1.8 `std::shared_ptr<telux::data::net::IFirewallEntry> telux::data::DataFactory::getNew-FirewallEntry ( IpProtocol proto, Direction direction, IpFamilyType ipFamilyType )`

Get Firewall entry based on IP protocol and set respective filter (i.e. TCP or UDP)

in	<i>proto</i>	telux::data::IpProtocol
in	<i>direction</i>	telux::data::Direction
in	<i>ipFamilyType</i>	Identifies IP family type telux::data::IpFamilyType

Returns

instance of IFirewallEntry

Note

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

5.8.2.15.1.9 std::shared_ptr<IpFilter> telux::data::DataFactory::getNewIpFilter (IpProtocol *proto*)

Get [IpFilter](#) instance based on IP Protocol, This can be used in Firewall Manager and Data Filter Manager

Parameters

in	<i>proto</i>	telux::data::IpProtocol 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 [IpFilter](#) based on IpProtocol filter (i.e TCP, UDP)

Note

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

5.8.2.15.1.10 std::shared_ptr<telux::data::net::IVlanManager> telux::data::DataFactory::getVlanManager (telux::data::OperationType *oprType*)

Get VLAN Manager

Parameters

in	<i>oprType</i>	Required operation type telux::data::OperationType

Returns

instance of IVlanManager

Note

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

5.8.2.15.1.11 `std::shared_ptr<telux::data::net::ISocksManager> telux::data::DataFactory::getSocksManager ( telux::data::OperationType oprType )`

Get Socks Manager

Parameters

<code>in</code>	<code><i>oprType</i></code>	Required operation type telux::data::OperationType

Returns

instance of ISocksManager

Note

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

5.8.2.15.1.12 `std::shared_ptr<telux::data::net::IBridgeManager> telux::data::DataFactory::getBridgeManager ( )`

Get Software Bridge Manager

Returns

instance of IBridgeManager

Note

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

5.8.2.15.1.13 `std::shared_ptr<telux::data::net::IL2tpManager> telux::data::DataFactory::getL2tpManager ( )`

Get L2TP Manager

Returns

instance of IL2tpManager

Note

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

5.8.2.15.1.14 `virtual std::shared_ptr<telux::data::IDataSettingsManager> telux::data::DataFactory::get-DataSettingsManager ( telux::data::OperationType oprType ) [virtual]`

Gets the data settings manager instance.

Parameters

<code>in</code>	<code>oprType</code>	Required operation type telux::data::OperationType

Returns

[IDataSettingsManager](#) instance

5.8.2.16 class telux::data::IDataFilterListener

Listener class for listening to filtering mode notifications, like Data filtering mode change. Client need to implement these methods. The methods in listener can be invoked from multiple threads. So the client needs to make sure that the implementation is thread-safe.

Note

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

Public member functions

- virtual void [onDataRestrictModeChange](#) ([DataRestrictMode](#) mode)
- virtual [~IDataFilterListener](#) ()

5.8.2.16.1 Constructors and Destructors

5.8.2.16.1.1 `virtual telux::data::IDataFilterListener::~IDataFilterListener ( ) [virtual]`

Destructor of [IDataFilterListener](#)

5.8.2.16.2 Member Function Documentation

5.8.2.16.2.1 `virtual void telux::data::IDataFilterListener::onDataRestrictModeChange ( DataRestrict-Mode mode ) [virtual]`

This function is called when the data filtering mode is changed for the packet data session.

in	state	the current data filter mode

Note

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

5.8.2.17 class telux::data::IDataFilterManager

[IDataFilterManager](#) class provides interface to enable/disable the data restrict filters and register for data restrict filter. The filtering can be done at any time. One such use case is to do it when we want the AP to suspend so that we are not waking up the AP due to spurious incoming messages. Also to make sure the DataRestrict mode is enabled.

In contrary to when DataRestrict mode is disabled, modem will forward all the incoming data packets to AP and might wake up AP unnecessarily.

Note

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

Public member functions

- virtual bool [isReady](#) ()=0
- virtual std::future< bool > [onReady](#) ()=0
- virtual [telux::common::Status registerListener](#) (std::weak_ptr< [IDataFilterListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak_ptr< [IDataFilterListener](#) > listener)=0
- virtual [telux::common::Status setDataRestrictMode](#) ([DataRestrictMode](#) mode, [telux::common::ResponseCallback](#) callback=nullptr, int profileId=[PROFILE_ID_MAX](#), [IpFamilyType](#) ipFamilyType=[IpFamilyType::UNKNOWN](#))=0
- virtual [telux::common::Status requestDataRestrictMode](#) (std::string ifaceName, [DataRestrictModeCb](#) callback)=0
- virtual [telux::common::Status addDataRestrictFilter](#) (std::shared_ptr< [IipFilter](#) > &filter, [telux::common::ResponseCallback](#) callback=nullptr, int profileId=[PROFILE_ID_MAX](#), [IpFamilyType](#) ipFamilyType=[IpFamilyType::UNKNOWN](#))=0
- virtual [telux::common::Status removeAllDataRestrictFilters](#) ([telux::common::ResponseCallback](#) callback=nullptr, int profileId=[PROFILE_ID_MAX](#), [IpFamilyType](#) ipFamilyType=[IpFamilyType::UNKNOWN](#))=0
- virtual int [getSlotId](#) ()=0
- virtual [~IDataFilterManager](#) ()

5.8.2.17.1 Constructors and Destructors

5.8.2.17.1.1 **virtual telux::data::IDataFilterManager::~IDataFilterManager () [virtual]**

Destructor of [IDataFilterManager](#)

5.8.2.17.2 Member Function Documentation

5.8.2.17.2.1 **virtual bool telux::data::IDataFilterManager::isReady () [pure virtual]**

Checks the status of Data Filter Service and if the other APIs are ready for use, and returns the result.

Returns

True if the services are ready otherwise false.

Note

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

5.8.2.17.2.2 **virtual std::future<bool> telux::data::IDataFilterManager::onReady () [pure virtual]**

Wait for Data Filter Service to be ready.

Returns

A future that caller can wait on to be notified when Data Filter Service are ready.

Note

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

5.8.2.17.2.3 **virtual telux::common::Status telux::data::IDataFilterManager::registerListener (std::weak_ptr< IDataFilterListener > *listener*) [pure virtual]**

Register a listener for powersave filtering mode notifications.

Parameters

in	<i>listener</i>	- Pointer of IDataFilterListener object that processes the notification

Returns

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

Note

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

5.8.2.17.2.4 `virtual telux::common::Status telux::data::IDataFilterManager::deregisterListener ( std::weak_ptr< IDataFilterListener > listener ) [pure virtual]`

Remove a previously registered listener.

Parameters

in	<i>listener</i>	- Previously registered IDataFilterListener that needs to be removed

Returns

Status of deregisterListener, success or suitable status code

Note

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

5.8.2.17.2.5 `virtual telux::common::Status telux::data::IDataFilterManager::setDataRestrictMode ( DataRestrictMode mode, telux::common::ResponseCallback callback = nullptr, int profileId = PROFILE_ID_MAX, IpFamilyType ipFamilyType = IpFamilyType::UNKNOWN ) [pure virtual]`

Changes the Data Powersave filter mode and auto exit feature.

This API enables or disables the powersave filtering mode of the packet data session..

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.

Note

Currently, this API does not support enabling data restrict filters per profile. This API will only accept the default value (PROFILE_ID_MAX). Any other value will result in an error.

in	<i>ipFamilyType</i>	- Optional IP Family type IpFamilyType . If user does not specify the ip family type, then the API applies to all the currently running data connection. If user wants to apply the changes to any specific data connection, then its ip family type can be specified as input.

Note

Currently, this API does not support enabling data restrict filters per IP Family type. This API will only accept the default value (IpFamilyType::UNKNOWN). Any other value will result in an error.

Returns

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

Note

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

5.8.2.17.2.6 `virtual telux::common::Status telux::data::IDataFilterManager::requestDataRestrictMode ( std::string ifaceName, DataRestrictModeCb callback ) [pure virtual]`

Get the current Data Powersave filter mode

Parameters

in	<i>ifaceName</i>	- Interface name for data connection. Note: Some platforms support filter mode that applies to all PDNs (i.e. global) and some platforms support filter mode per PDN. On platforms that support global filter mode, this parameter will be ignored.
in	<i>callback</i>	- callback function to get the result of API.

Returns

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

Note

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

5.8.2.17.2.7 `virtual telux::common::Status telux::data::IDataFilterManager::addDataRestrictFilter ( std::shared_ptr< llpFilter > & filter, telux::common::ResponseCallback callback = nullptr, int profileId = PROFILE_ID_MAX, IpFamilyType ipFamilyType = IpFamilyType::UNKNOWN ) [pure virtual]`

This API adds a filter rules for a packet data session to achieve power savings. In case when DataRestrict mode is enabled and AP is in suspended state, Modem will filter all the incoming data packet and route them to AP only if filter rules added via addDataRestrictFilter API matches the criteria, else they are

queued at Modem itself and not forwarded to AP, until filter mode is disabled.

Parameters

in	<i>filter</i>	- Filter rule.
in	<i>callback</i>	- Optional callback to get the response.
in	<i>profileId</i>	- Optional Profile ID for data connection. If user does not specify the profile id, then the API applies to all the currently running data connection. If user wants to apply the changes to any specific data connection, then its profile id can be specified as input.

Note

Currently, this API does not support enabling data restrict filters per profile. This API will only accept the default value (PROFILE_ID_MAX). Any other value will result in an error.

Parameters

in	<i>ipFamilyType</i>	- Optional IP Family type IpFamilyType . If user does not specify the ip family type, then the API applies to all the currently running data connection. If user wants to apply the changes to any specific data connection, then its ip family type can be specified as input.

Note

Currently, this API does not support enabling data restrict filters per IP Family type. This API will only accept the default value (IpFamilyType::UNKNOWN). Any other value will result in an error.

Returns

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

Note

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

5.8.2.17.2.8 `virtual telux::common::Status telux::data::IDataFilterManager::removeAllDataRestrictFilters ( telux::common::ResponseCallback callback = nullptr, int profileId = PROFILE_ID_MAX, IpFamilyType ipFamilyType = IpFamilyType::UNKNOWN ) [pure virtual]`

This API removes all the previous added powersave filter for a packet data session

Parameters

in	<i>callback</i>	- Optional callback to get the response.
in	<i>profileId</i>	- Optional Profile ID for data connection. If user does not specify the profile id, then the API applies to all the currently running data connection. If user wants to apply the changes to any specific data connection, then its profile id can be specified as input.

Note

Currently, this API does not support enabling data restrict filters per profile. This API will only accept the default value (PROFILE_ID_MAX). Any other value will result in an error.

Parameters

in	<i>ipFamilyType</i>	- Optional IP Family type IpFamilyType . If user does not specify the ip family type, then the API applies to all the currently running data connection. If user wants to apply the changes to any specific data connection, then its ip family type can be specified as input.

Note

Currently, this API does not support enabling data restrict filters per IP Family type. This API will only accept the default value (IpFamilyType::UNKNOWN). Any other value will result in an error.

Returns

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

Note

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

5.8.2.17.2.9 virtual int telux::data::IDataFilterManager::getSlotId () [pure virtual]

Get associated slot id for the Data Filter Manager.

Returns

SlotId

5.8.2.18 class telux::data::DataProfile

[DataProfile](#) class represents single data profile on the modem.

Public member functions

- [DataProfile](#) (int id, const std::string &name, const std::string &apn, const std::string &username, const std::string &password, [IpFamilyType](#) ipFamilyType, [TechPreference](#) techPref, [AuthProtocolType](#) authType, [ApnTypes](#) apnTypes)
- int [getId](#) ()
- std::string [getName](#) ()
- std::string [getApn](#) ()
- std::string [getUserName](#) ()
- std::string [getPassword](#) ()
- [TechPreference](#) [getTechPreference](#) ()
- [AuthProtocolType](#) [getAuthProtocolType](#) ()
- [IpFamilyType](#) [getIpFamilyType](#) ()
- [ApnTypes](#) [getApnTypes](#) ()
- std::string [toString](#) ()

Static Public Attributes

- static constexpr int [PROFILE_ID_INVALID](#) = -1

5.8.2.18.1 Constructors and Destructors

5.8.2.18.1.1 `telux::data::DataProfile::DataProfile ( int id, const std::string & name, const std::string & apn, const std::string & username, const std::string & password, IpFamilyType ipFamilyType, TechPreference techPref, AuthProtocolType authType, ApnTypes apnTypes )`

5.8.2.18.2 Member Function Documentation

5.8.2.18.2.1 `int telux::data::DataProfile::getId ( )`

Get profile identifier.

Returns

profile id

5.8.2.18.2.2 `std::string telux::data::DataProfile::getName ( )`

Get profile name.

Returns

profile name

5.8.2.18.2.3 std::string telux::data::DataProfile::getApn ()

Get Access Point Name (APN) name.

Returns

APN name

5.8.2.18.2.4 std::string telux::data::DataProfile::getUserName ()

Get profile user name.

Returns

user name

5.8.2.18.2.5 std::string telux::data::DataProfile::getPassword ()

Get profile password.

Returns

profile password

5.8.2.18.2.6 TechPreference telux::data::DataProfile::getTechPreference ()

Get technology preference.

Returns

TechPreference [TechPreference](#)

5.8.2.18.2.7 AuthProtocolType telux::data::DataProfile::getAuthProtocolType ()

Get authentication preference.

Returns

AuthProtocolType [AuthProtocolType](#)

5.8.2.18.2.8 IpFamilyType telux::data::DataProfile::getIpFamilyType ()

Get IP Family type.

Returns

IpFamilyType [IpFamilyType](#)

5.8.2.18.2.9 `ApnTypes` `telux::data::DataProfile::getApnTypes ( )`

Get Apn type mask.

Returns

`ApnTypes` [ApnTypes](#)

5.8.2.18.2.10 `std::string` `telux::data::DataProfile::toString ( )`

Get the text related informative representation of this object.

Returns

String containing informative string.

5.8.2.18.3 Field Documentation**5.8.2.18.3.1 `constexpr int` `telux::data::DataProfile::PROFILE_ID_INVALID = -1` `[static]`****5.8.2.19 `class` `telux::data::IDataProfileListener`**

Listener class for getting profile change notification.

The methods in the listener can be invoked from multiple threads. It is client's responsibility to make sure the implementation is thread safe.

Public member functions

- virtual void [onServiceStatusChange](#) (`telux::common::ServiceStatus` status, `SlotId` slotId)
- virtual void [onProfileUpdate](#) (int profileId, [TechPreference](#) techPreference, [ProfileChangeEvent](#) event)
- virtual [~IDataProfileListener](#) ()

5.8.2.19.1 Constructors and Destructors**5.8.2.19.1.1 `virtual` `telux::data::IDataProfileListener::~~IDataProfileListener ( )` `[virtual]`**

Destructor of [IDataProfileListener](#)

5.8.2.19.2 Member Function Documentation**5.8.2.19.2.1 `virtual` void `telux::data::IDataProfileListener::onServiceStatusChange ( telux::common::ServiceStatus status, SlotId slotId )` `[virtual]`**

This function is called when service status changes.

in	<i>status</i>	- ServiceStatus
in	<i>slotId</i>	- SlotId

5.8.2.19.2.2 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.

5.8.2.20 class telux::data::IDataProfileManager

[IDataProfileManager](#) is a primary interface for profile management.

Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status requestProfileList](#) (std::shared_ptr< [IDataProfileListCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status createProfile](#) (const [ProfileParams](#) &profileParams, std::shared_ptr< [IDataCreateProfileCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status deleteProfile](#) (uint8_t profileId, [TechPreference](#) techPreference, std::shared_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status modifyProfile](#) (uint8_t profileId, const [ProfileParams](#) &profileParams, std::shared_ptr< [telux::common::ICommandResponseCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status queryProfile](#) (const [ProfileParams](#) &profileParams, std::shared_ptr< [IDataProfileListCallback](#) > callback=nullptr)=0
- virtual [telux::common::Status requestProfile](#) (uint8_t profileId, [TechPreference](#) techPreference, std::shared_ptr< [IDataProfileCallback](#) > callback=nullptr)=0
- virtual int [getSlotId](#) ()=0
- virtual [telux::common::Status registerListener](#) (std::weak_ptr< [telux::data::IDataProfileListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak_ptr< [telux::data::IDataProfileListener](#) > listener)=0
- virtual [~IDataProfileManager](#) ()

5.8.2.20.1 Constructors and Destructors

5.8.2.20.1.1 `virtual telux::data::IDataProfileManager::~IDataProfileManager ( ) [virtual]`

Destructor for [IDataProfileManager](#)

5.8.2.20.2 Member Function Documentation

5.8.2.20.2.1 `virtual bool telux::data::IDataProfileManager::isSubsystemReady ( ) [pure virtual]`

Checks if the data profile manager is ready.

Returns

True if data profile subsystem is ready for service otherwise false.

5.8.2.20.2.2 `virtual std::future<bool> telux::data::IDataProfileManager::onSubsystemReady ( ) [pure virtual]`

Waits for data profile subsystem to be ready.

Returns

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

5.8.2.20.2.3 `virtual telux::common::Status telux::data::IDataProfileManager::requestProfileList ( std::shared_ptr< IDataProfileListCallback > callback = nullptr ) [pure virtual]`

Request list of profiles supported by the device.

Parameters

<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.

5.8.2.20.2.4 `virtual telux::common::Status telux::data::IDataProfileManager::createProfile ( const ProfileParams & profileParams, std::shared_ptr< IDataCreateProfileCallback > callback = nullptr ) [pure virtual]`

Create profile based on data profile params.

Parameters

<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.

5.8.2.20.2.5 `virtual telux::common::Status telux::data::IDataProfileManager::deleteProfile ( uint8_t profileId, TechPreference techPreference, std::shared_ptr< telux::common::ICommand-ResponseCallback > callback = nullptr ) [pure virtual]`

Delete profile corresponding to profile identifier.

The deletion of a profile does not affect profile index assignments.

Parameters

in	<i>profileId</i>	Profile identifier
in	<i>techPreference</i>	Technology Preference like 3GPP / 3GPP2
in	<i>callback</i>	Callback pointer to get the result of delete profile

Returns

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

5.8.2.20.2.6 `virtual telux::common::Status telux::data::IDataProfileManager::modifyProfile ( uint8_t profileId, const ProfileParams & profileParams, std::shared_ptr< telux::common::ICommandResponseCallback > callback = nullptr ) [pure virtual]`

Modify existing profile with new profile params.

Parameters

in	<i>profileId</i>	Profile identifier of profile to be modified
in	<i>profileParams</i>	New profileParams configuration passed for updating existing profile
in	<i>callback</i>	Callback pointer to get the result of modify profile

Returns

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

5.8.2.20.2.7 `virtual telux::common::Status telux::data::IDataProfileManager::queryProfile ( const ProfileParams & profileParams, std::shared_ptr< IDataProfileListCallback > callback = nullptr ) [pure virtual]`

Lookup modem profile/s based on given profile params.

Parameters

in	<i>profileParams</i>	ProfileParams configuration to be passed
in	<i>callback</i>	Callback pointer to get the result of query profile

Returns

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

5.8.2.20.2.8 `virtual telux::common::Status telux::data::IDataProfileManager::requestProfile ( uint8_t profileId, TechPreference techPreference, std::shared_ptr< IDataProfileCallback > callback = nullptr ) [pure virtual]`

Get data profile corresponding to profile identifier.

Parameters

in	<i>profileId</i>	Profile identifier
in	<i>techPreference</i>	Technology preference TechPreference
in	<i>callback</i>	Callback pointer to get the result of get profile by ID

Returns

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

5.8.2.20.2.9 `virtual int telux::data::IDataProfileManager::getSlotId ( ) [pure virtual]`

Get associated slot id for the Data Profile Manager.

Returns

SlotId

5.8.2.20.2.10 `virtual telux::common::Status telux::data::IDataProfileManager::registerListener ( std::weak_ptr< telux::data::IDataProfileListener > listener ) [pure virtual]`

Listen for create, delete and modify profile events.

Parameters

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

Returns

Status.

5.8.2.20.2.11 `virtual telux::common::Status telux::data::IDataProfileManager::deregisterListener ( std::weak_ptr< telux::data::IDataProfileListener > listener ) [pure virtual]`

De-register listener.

in	<i>listener</i>	- Listener to be de-registered.

Returns

Status.

5.8.2.21 class telux::data::IDataCreateProfileCallback

Interface for create profile callback object. Client needs to implement this interface to get single shot responses for command like create profile.

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

Public member functions

- virtual void [onResponse](#) (int profileId, [telux::common::ErrorCode](#) error)

5.8.2.21.1 Member Function Documentation

5.8.2.21.1 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 IDataProfileManager::requestProfile to get the data profile
in	<i>error</i>	telux::common::ErrorCode

5.8.2.22 class telux::data::IDataProfileListCallback

Interface for getting list of [DataProfile](#) using callback. Client needs to implement this interface to get single shot responses for commands like get profile list and query profile.

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

Public member functions

- virtual void [onProfileListResponse](#) (const std::vector< std::shared_ptr< [DataProfile](#) >> &profiles, [telux::common::ErrorCode](#) error)

5.8.2.22.1 Member Function Documentation

5.8.2.22.1.1 `virtual void telux::data::IDataProfileListCallback::onProfileListResponse ( const std::vector< std::shared_ptr< DataProfile >> & profiles, telux::common::ErrorCode error ) [virtual]`

This function is called with the response to requestProfileList API or queryProfile API.

Parameters

in	<i>profiles</i>	List of profiles supported by the device
in	<i>error</i>	telux::common::ErrorCode

5.8.2.23 class telux::data::IDataProfileCallback

Interface for getting [DataProfile](#) using callback. Client needs to implement this interface to get single shot responses for command like create profile.

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

Public member functions

- virtual void [onResponse](#) (const std::shared_ptr< [DataProfile](#) > &profile, [telux::common::ErrorCode](#) error)

5.8.2.23.1 Member Function Documentation

5.8.2.23.1.1 `virtual void telux::data::IDataProfileCallback::onResponse ( const std::shared_ptr< DataProfile > & profile, telux::common::ErrorCode error ) [virtual]`

This function is called with the response to [IDataProfileManager::requestProfile](#) API.

Parameters

in	<i>profile</i>	Response of data profile
in	<i>error</i>	telux::common::ErrorCode

5.8.2.24 class telux::data::IDataSettingsManager

The Data Settings Manager class provides an interface to the data subsystem settings.

Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status](#) [switchBackHaul](#) ([BackhaulInfo](#) source, [BackhaulInfo](#) dest, bool applyToAll=false, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::ErrorCode](#) [setLatencyConfig](#) ([LatencyConfig](#) latencyConfig)=0

- virtual `telux::common::ErrorCode getLatencyConfig (LatencyConfig &latencyConfig)=0`
- virtual `telux::common::Status registerListener (std::weak_ptr< IDataSettingsListener > listener)=0`
- virtual `telux::common::Status deregisterListener (std::weak_ptr< IDataSettingsListener > listener)=0`
- virtual `~IDataSettingsManager ()`

5.8.2.24.1 Constructors and Destructors

5.8.2.24.1.1 `virtual telux::data::IDataSettingsManager::~IDataSettingsManager ( ) [virtual]`

Destructor for `IDataSettingsManager`

5.8.2.24.2 Member Function Documentation

5.8.2.24.2.1 `virtual bool telux::data::IDataSettingsManager::isSubsystemReady ( ) [pure virtual]`

Checks if the data subsystem is ready.

Returns

if Settings Manager is ready, false otherwise.

5.8.2.24.2.2 `virtual std::future<bool> telux::data::IDataSettingsManager::onSubsystemReady ( ) [pure virtual]`

Waits for the data subsystem to be ready.

Returns

A future that caller can wait on to be notified when data settings manager is ready.

5.8.2.24.2.3 `virtual telux::common::Status telux::data::IDataSettingsManager::switchBackHaul ( BackhaulInfo source, BackhaulInfo dest, bool applyToAll = false, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Switch backhaul to be used by traffic. Provides the ability to re-route clients traffic from one backhaul to another. Clients must call this API for each backhaul switch. For instance, if the default bridge (bridge0) and the on-demand bridge (bridges created by VLANs) need to be re-routed to WLAN, this API must be called twice for the default profile ID and the on-demand profile ID.. If destination backhaul is WLAN (WLAN in Station Mode):

- Traffic associated with the default and on-demand bridges will be re-routed to WLAN backhaul.
- Client traffic can only be re-routed to WLAN backhaul if the station is connected to an external access point.
- VLANs mapped to WWAN backhaul will be automatically mapped to WLAN backhaul.

- Firewall and DMZ rules configured on WLAN backhaul (if configured before calling this API) will be automatically activated. If destination backhaul is WWAN:
 - Any VLAN profile ID mapping configured in the destination backhaul prior to calling this API will be applied automatically.
 - Any firewall or DMZ rule configured on WWAN backhaul before calling this API will be activated automatically.

Parameters

in	<i>source</i>	Backhaul telux::data::BackhaulInfo to re-route traffic from
in	<i>dest</i>	Backhaul telux::data::BackhaulType to re-route traffic to
in	<i>applyToAll</i>	Traffic on all source backhauls will be routed to dest backhauls if the source backhaul type is telux::data::BackhaulType::WWAN , traffic on all WWAN backhauls (default and on-demand) will be routed to dest backhaul. if dest backhaul type is telux::data::BackhaulType::WWAN traffic on source backhaul will be routed to WWAN backhauls (default and on-demand) based on vlan-backhaul binding set by telux::data::net::IVlanManager::bindToBackhaul
in	<i>callback</i>	Optional callback to get the response for switchBackHaul.

Returns

Status of switchBackHaul, i.e., success or applicable status code

Note

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

5.8.2.24.2.4 virtual [telux::common::ErrorCode](#) [telux::data::IDataSettingsManager::setLatencyConfig \(LatencyConfig latencyConfig \)](#) [pure virtual]

Sets the latency level for data traffic that has been marked as prioritized data. This only affects the latency level in the modem on the UE. This does not impact network level latency or QoS. This API allows clients to set the uplink latency level of data they deem as time critical compared to the rest of the data flowing in the system. If the latency level is set to [LatencyLevel::LOW](#), then the implementation will prioritize this data over other non-prioritized data flows. Configuration can be reset by setting [LatencyLevel::NORMAL](#) (no priority).

This is a global setting applicable to all WWAN profiles. Configuration set via this API is not persistent over reboot or sub-system restart (updated via [IDataSettingsListener::onServiceStatusChange](#)). After reboot or SSR, configuration will reset to default, i.e., [LatencyLevel::NORMAL](#) (no priority).

Parameters

in	<i>latencyConfig</i>	Latency/priority configuration

Returns

Immediate error code of setLatencyConfig, i.e., success or suitable error code.

Note

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

5.8.2.24.2.5 **virtual telux::common::ErrorCode telux::data::IDataSettingsManager::getLatencyConfig (LatencyConfig & *latencyConfig*) [pure virtual]**

Get latency level. This API can be used to get the current latency configuration. See [setLatencyConfig](#) for more information.

Parameters

out	<i>latencyConfig</i>	Latency/priority configuration

Returns

Immediate error code of getLatencyConfig, i.e., success or suitable error code.

Note

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

5.8.2.24.2.6 **virtual telux::common::Status telux::data::IDataSettingsManager::registerListener (std::weak_ptr< IDataSettingsListener > *listener*) [pure virtual]**

Registers a listener to receive data settings notifications.

Parameters

in	<i>listener</i>	Pointer to the IDataSettingsListener object to process received notifications.

Returns

Status of registerListener success or applicable status code

5.8.2.24.2.7 **virtual telux::common::Status telux::data::IDataSettingsManager::deregisterListener (std::weak_ptr< IDataSettingsListener > *listener*) [pure virtual]**

Removes a previously added listener.

<i>in</i>	<i>listener</i>	Point to the IDataSettingsListener object to remove.

Returns

Status of deregisterListener success or applicable status code

5.8.2.25 class telux::data::IDataSettingsListener

Interface for the Data...Clients need to implement this interface to access Data Settings service notifications like onServiceStatusChange.

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

Public member functions

- virtual void [onServiceStatusChange](#) ([telux::common::ServiceStatus](#) status)
- virtual [~IDataSettingsListener](#) ()

5.8.2.25.1 Constructors and Destructors

5.8.2.25.1.1 virtual telux::data::IDataSettingsListener::~~IDataSettingsListener () [virtual]

Destructor for [IDataSettingsListener](#)

5.8.2.25.2 Member Function Documentation

5.8.2.25.2.1 virtual void telux::data::IDataSettingsListener::onServiceStatusChange (telux::common::ServiceStatus *status*) [virtual]

This function is called when service status changes.

Parameters

<i>in</i>	<i>status</i>	- ServiceStatus

5.8.2.26 class telux::data::IServingSystemManager

Serving System Manager class provides APIs related to the serving system for data functionality. For example, ability to query or be notified about the state of the platform's WWAN PS data serving information.

Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status](#) [requestServiceStatus](#) ([RequestServiceStatusResponseCb](#) callback)=0

- virtual `telux::common::Status requestRoamingStatus (RequestRoamingStatusResponseCb callback)=0`
- virtual `telux::common::Status makeDormant (telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status registerListener (std::weak_ptr< IServingSystemListener > listener)=0`
- virtual `telux::common::Status deregisterListener (std::weak_ptr< IServingSystemListener > listener)=0`
- virtual `SlotId getSlotId ()=0`
- virtual `~IServingSystemManager ()`

5.8.2.26.1 Constructors and Destructors

5.8.2.26.1.1 `virtual telux::data::IServingSystemManager::~IServingSystemManager ( ) [virtual]`

Destructor of `IServingSystemManager`

5.8.2.26.2 Member Function Documentation

5.8.2.26.2.1 `virtual bool telux::data::IServingSystemManager::isSubsystemReady ( ) [pure virtual]`

Checks if the data subsystem is ready.

Returns

True if Data Connection Manager is ready for service, otherwise returns false.

5.8.2.26.2.2 `virtual std::future<bool> telux::data::IServingSystemManager::onSubsystemReady ( ) [pure virtual]`

Wait for data subsystem to be ready.

Returns

A future that caller can wait on to be notified when card manager is ready.

5.8.2.26.2.3 `virtual telux::common::Status telux::data::IServingSystemManager::requestServiceStatus ( RequestServiceStatusResponseCb callback ) [pure virtual]`

Queries the current serving network status

Parameters

in	<i>callback</i>	callback to get response for requestServiceStatus

Returns

Status of requestServiceStatus i.e. success or suitable status code. if requestServiceStatus returns failure, callback will not be invoked.

Note

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

5.8.2.26.2.4 **virtual telux::common::Status telux::data::IServingSystemManager::requestRoamingStatus (RequestRoamingStatusResponseCb *callback*) [pure virtual]**

Queries the current roaming status

Parameters

in	<i>callback</i>	callback to get response for requestRoamingStatus

Returns

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

Note

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

5.8.2.26.2.5 **virtual telux::common::Status telux::data::IServingSystemManager::makeDormant (telux::common::ResponseCallback *callback* = nullptr) [pure virtual]**

Request modem switch to dormant state: Certain network operations can only be performed when modem is in dormant state. This API provides an ability for clients to request modem to immediately transition to dormant state for such scenarios.

Clients must ensure no data calls are in process of bring up/tear down and there is no traffic on any active data calls when this API is called.

Parameters

in	<i>callback</i>	optional callback to get the response of makeDormancy

Returns

telux::common::ErrorCode::SUCCESS if request is honored by network.

telux::common::ErrorCode::INVALID_STATE is returned if:

- There is no active data calls
- Any Data calls is going through bring up/tear down
- There is data traffic on any active data calls If API fails, application is responsible for

re-attempting operation at later time once the above conditions are met.

Note

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

5.8.2.26.2.6 `virtual telux::common::Status telux::data::IServingSystemManager::registerListener ( std::weak_ptr< IServingSystemListener > listener ) [pure virtual]`

Register a listener for specific updates from serving system.

Parameters

in	<i>listener</i>	Pointer of IServingSystemListener object that processes the notification

Returns

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

5.8.2.26.2.7 `virtual telux::common::Status telux::data::IServingSystemManager::deregisterListener ( std::weak_ptr< IServingSystemListener > listener ) [pure virtual]`

Deregister the previously added listener.

Parameters

in	<i>listener</i>	Previously registered IServingSystemListener that needs to be removed

Returns

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

5.8.2.26.2.8 `virtual SlotId telux::data::IServingSystemManager::getSlotId ( ) [pure virtual]`

Get associated slot id for the Serving System Manager.

Returns

SlotId

5.8.2.27 class telux::data::IServingSystemListener

Listener class for data serving system change notification.

The listener method can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe.

Public member functions

- virtual void [onServiceStatusChange](#) ([telux::common::ServiceStatus](#) status)
- virtual void [onServiceStateChanged](#) ([ServiceStatus](#) status)
- virtual void [onRoamingStatusChanged](#) ([RoamingStatus](#) status)
- virtual [~IServingSystemListener](#) ()

5.8.2.27.1 Constructors and Destructors

5.8.2.27.1.1 virtual [telux::data::IServingSystemListener::~~IServingSystemListener](#) () [[virtual](#)]

Destructor of [IServingSystemListener](#)

5.8.2.27.2 Member Function Documentation

5.8.2.27.2.1 virtual void [telux::data::IServingSystemListener::onServiceStatusChange](#) ([telux::common::ServiceStatus](#) *status*) [[virtual](#)]

This function is called when service status changes.

Parameters

in	<i>status</i>	- ServiceStatus

5.8.2.27.2.2 virtual void [telux::data::IServingSystemListener::onServiceStateChanged](#) ([ServiceStatus](#) *status*) [[virtual](#)]

This function is called whenever service state is changed.

Parameters

in	<i>status</i>	ServiceStatus

5.8.2.27.2.3 virtual void [telux::data::IServingSystemListener::onRoamingStatusChanged](#) ([RoamingStatus](#) *status*) [[virtual](#)]

This function is called whenever roaming status is changed.

in	status	RoamingStatus

5.8.2.28 union telux::data::DataCallEndReason.__unnamed__

Data fields

Type	Field	Description
MobileIp-ReasonCode	IpCode	
InternalReason-Code	internalCode	
CallManager-ReasonCode	cmCode	
SpecReason-Code	specCode	
PPPReason-Code	pppCode	
EHRPD-ReasonCode	ehrpdcCode	
Ipv6Reason-Code	ipv6Code	
Handoff-ReasonCode	handOffCode	

5.8.3 Enumeration Type Documentation

5.8.3.1 enum telux::data::IpFamilyType

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

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

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

Defines the supported filtering mode of the packet data session. DataRestrictFilter

Enumerator

UNKNOWN
DISABLE
ENABLE

5.8.3.5 enum telux::data::ApnMaskType

Specifies APN types that can be set while creating or modifying a profile

Enumerator

APN_MASK_TYPE_DEFAULT APN type for default/internet traffic
APN_MASK_TYPE_IMS APN type for the IP multimedia subsystem
APN_MASK_TYPE_MMS APN type for the multimedia messaging service
APN_MASK_TYPE_DUN APN type for the dial up network
APN_MASK_TYPE_SUPL APN type for secure user plane location
APN_MASK_TYPE_HIPRI APN type for high priority mobile data
APN_MASK_TYPE_FOTA APN type for over the air administration
APN_MASK_TYPE_CBS APN type for carrier branded services
APN_MASK_TYPE_IA APN type for initial attach
APN_MASK_TYPE_EMERGENCY APN type for emergency
APN_MASK_TYPE_UT APN type for UT
APN_MASK_TYPE_MCX APN type for mission critical service

5.8.3.6 enum telux::data::DataCallStatus

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

5.8.3.7 enum telux::data::DataBearerTechnology

Bearer technology types (returned with getCurrentBearerTech).

Enumerator

UNKNOWN Unknown bearer.
CDMA_1X 1X technology.
EVDO_REV0 CDMA Rev 0.
EVDO_REVA CDMA Rev A.
EVDO_REVB CDMA Rev B.
EHRPD EHRPD.
FMC Fixed mobile convergence.
HRPD HRPD
BEARER_TECH_3GPP2_WLAN IWLAN
WCDMA WCDMA.
GPRS GPRS.
HSDPA HSDPA.
HSUPA HSUPA.
EDGE EDGE.
LTE LTE.
HSDPA_PLUS HSDPA+.
DC_HSDPA_PLUS DC HSDPA+.
HSPA HSPA
BEARER_TECH_64_QAM 64 QAM.
TDSCDMA TD-SCDMA.
GSM GSM
BEARER_TECH_3GPP_WLAN IWLAN
BEARER_TECH_5G 5G

5.8.3.8 enum telux::data::EndReasonType

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

5.8.3.9 enum telux::data::MobileIpReasonCode

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

5.8.3.10 enum telux::data::InternalReasonCode

Data call end/termination reason code for EndReasonType::CE_INTERNAL

Enumerator

CE_RETRY
CE_INTERNAL_ERROR
CE_CALL_ENDED
CE_INTERNAL_UNKNOWN_CAUSE_CODE
CE_UNKNOWN_CAUSE_CODE
CE_CLOSE_IN_PROGRESS
CE_NW_INITIATED_TERMINATION

CE_APP_PREEMPTED
CE_ERR_PDN_IPV4_CALL_DISALLOWED
CE_ERR_PDN_IPV4_CALL_THROTTLED
CE_ERR_PDN_IPV6_CALL_DISALLOWED
CE_ERR_PDN_IPV6_CALL_THROTTLED
CE_MODEM_RESTART
CE_PDP_PPP_NOT_SUPPORTED
CE_UNPREFERRED_RAT
CE_PHYS_LINK_CLOSE_IN_PROGRESS
CE_APN_PENDING_HANDOVER
CE_PROFILE_BEARER_INCOMPATIBLE
CE_MMGSDI_CARD_EVT
CE_LPM_OR_PWR_DOWN
CE_APN_DISABLED
CE_MPIT_EXPIRED
CE_IPV6_ADDR_TRANSFER_FAILED
CE_TRAT_SWAP_FAILED
CE_EHRPD_TO_HRPD_FALLBACK
CE_MANDATORY_APN_DISABLED
CE_MIP_CONFIG_FAILURE
CE_INTERNAL_PDN_INACTIVITY_TIMER_EXPIRED
CE_MAX_V4_CONNECTIONS
CE_MAX_V6_CONNECTIONS
CE_APN_MISMATCH
CE_IP_VERSION_MISMATCH
CE_DUN_CALL_DISALLOWED
CE_INVALID_PROFILE
CE_INTERNAL_EPC_NONEPC_TRANSITION
CE_INVALID_PROFILE_ID
CE_INTERNAL_CALL_ALREADY_PRESENT
CE_IFACE_IN_USE
CE_IP_PDP_MISMATCH
CE_APN_DISALLOWED_ON_ROAMING
CE_APN_PARAM_CHANGE
CE_IFACE_IN_USE_CFG_MATCH
CE_NULL_APN_DISALLOWED
CE_THERMAL_MITIGATION
CE_SUBS_ID_MISMATCH
CE_DATA_SETTINGS_DISABLED
CE_DATA_ROAMING_SETTINGS_DISABLED
CE_APN_FORMAT_INVALID
CE_DDS_CALL_ABORT
CE_VALIDATION_FAILURE
CE_PROFILES_NOT_COMPATIBLE
CE_NULL_RESOLVED_APN_NO_MATCH
CE_INVALID_APN_NAME

5.8.3.11 enum telux::data::CallManagerReasonCode

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_LRRRC_CONN_REL_NORMAL
CE_LRRRC_CONN_REL_RLF
CE_LRRRC_CONN_REL_CRE_FAILURE
CE_LRRRC_CONN_REL_OOS_DURING_CRE
CE_LRRRC_CONN_REL_ABORTED
CE_LRRRC_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

5.8.3.12 enum telux::data::SpecReasonCode

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

5.8.3.13 enum telux::data::PPPReasonCode

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

5.8.3.14 enum telux::data::EHRPDReasonCode

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

5.8.3.15 enum telux::data::Ipv6ReasonCode

Data call end/termination reason code for EndReasonType::CE_IPV6

Enumerator

CE_PREFIX_UNAVAILABLE
CE_IPV6_ERR_HRPD_IPV6_DISABLED
CE_IPV6_DISABLED

5.8.3.16 enum telux::data::HandoffReasonCode

Data call end/termination reason code for EndReasonType::CE_HANDOFF

Enumerator

CE_VCER_HANDOFF_PREF_SYS_BACK_TO_SRAT

5.8.3.17 enum telux::data::ProfileChangeEvent

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

5.8.3.18 enum telux::data::OperationType

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.

5.8.3.19 enum telux::data::Direction

Direction of firewall rule

Enumerator

UPLINK Uplink Direction
DOWNLINK Downlink Direction

5.8.3.20 enum telux::data::InterfaceType

Peripheral Interface type

Enumerator

UNKNOWN UNKNOWN interface
WLAN Wireless local area network station (WLAN STA)
ETH Ethernet (ETH)
ECM Ethernet control model (ECM)
RNDIS Remote network driver interface specification (RNDIS)
MHI Modem host interface (MHI)
ETH2 Ethernet network interface card (ETH NIC2)
AP_PRIMARY Primary WLAN access point
AP_SECONDARY Secondary WLAN access point
AP_TERTIARY Tertiary WLAN access point
AP_QUATERNARY Quaternary WLAN access point

5.8.3.21 enum telux::data::BackhaulType

Specifies backhaul types

Enumerator

ETH
USB Ethernet Backhaul
WLAN USB Backhaul
WWAN WLAN Backhaul
BLE WWAN Backhaul with default profile ID set by
telux::data::DataConnectionManager::setDefaultProfile
MAX_SUPPORTED Bluetooth Backhaul

5.8.3.22 enum telux::data::LatencyLevel

Latency Level Latency level of configured IP packets. LatencyLevel::NORMAL is a default setting that has no additional priority.

Enumerator

INVALID
NORMAL Not configured
LOW Default, no priority

5.8.3.23 enum telux::data::ServiceState

State of Service

Enumerator

INACTIVE Service is inactive
ACTIVE Service is Active

5.8.3.24 enum telux::data::QosFlowStateChangeEvent

QOS flow state change type

Enumerator

UNKNOWN UNKNOWN state
ACTIVATED Flow activated
MODIFIED Flow modified
DELETED Flow deleted

5.8.3.25 enum telux::data::IpTrafficClassType

QOS flow IP traffic class type

Enumerator

UNKNOWN UNKNOWN type

CONVERSATIONAL Conversational IP Traffic class
STREAMING Streaming IP Traffic class
INTERACTIVE Interactive IP Traffic class
BACKGROUND Background IP Traffic class

5.8.3.26 enum telux::data::QosIPFlowMaskType

Specifies QOS IP Flow parameter mask

Enumerator

MASK_IP_FLOW_NONE
MASK_IP_FLOW_TRF_CLASS No parameters set
MASK_IP_FLOW_DATA_RATE_MIN_MAX Traffic class

5.8.3.27 enum telux::data::QosFlowMaskType

Specifies QOS Flow parameter mask

Enumerator

MASK_FLOW_NONE
MASK_FLOW_TX_GRANTED No parameters set
MASK_FLOW_RX_GRANTED TX Granted flow set
MASK_FLOW_TX_FILTERS RX Granted flow set
MASK_FLOW_RX_FILTERS TX filters set

5.9 Subscription Management

This section contains APIs related to Subscription Management.

5.9.1 Data Structure Documentation

5.9.1.1 class telux::tel::ISubscription

Subscription returns information about network operator subscription details pertaining to a SIM card.

Public member functions

- virtual std::string [getCarrierName](#) ()=0
- virtual std::string [getIccId](#) ()=0
- virtual int [getMcc](#) ()=0
- virtual int [getMnc](#) ()=0
- virtual std::string [getMobileCountryCode](#) ()=0
- virtual std::string [getMobileNetworkCode](#) ()=0
- virtual std::string [getPhoneNumber](#) ()=0
- virtual int [getSlotId](#) ()=0
- virtual std::string [getImsi](#) ()=0
- virtual std::string [getGID1](#) ()=0
- virtual std::string [getGID2](#) ()=0
- virtual [~ISubscription](#) ()

5.9.1.1.1 Constructors and Destructors

5.9.1.1.1.1 virtual telux::tel::ISubscription::~ISubscription () [virtual]

5.9.1.1.2 Member Function Documentation

5.9.1.1.2.1 virtual std::string telux::tel::ISubscription::getCarrierName () [pure virtual]

Retrieves the name of the carrier on which this subscription is made.

Returns

Name of the carrier.

5.9.1.1.2.2 virtual std::string telux::tel::ISubscription::getIccId () [pure virtual]

Retrieves the SIM's ICCID (Integrated Chip ID) - i.e SIM Serial Number.

Returns

Integrated Chip Id.

5.9.1.1.2.3 virtual int telux::tel::ISubscription::getMcc () [pure virtual]

Retrieves the mobile country code of the carrier to which the phone is connected.

Returns

Mobile Country Code.

Deprecated Use [telux::tel::ISubscription::getMobileCountryCode\(\)](#) API instead

5.9.1.1.2.4 virtual int telux::tel::ISubscription::getMnc () [pure virtual]

Retrieves the mobile network code of the carrier to which phone is connected.

Returns

Mobile Network Code.

Deprecated Use [telux::tel::ISubscription::getMobileNetworkCode\(\)](#) API instead

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

Retrieves the mobile country code(MCC) of the carrier to which the phone is connected.

Returns

mcc.

Note

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

5.9.1.1.2.6 virtual std::string telux::tel::ISubscription::getMobileNetworkCode () [pure virtual]

Retrieves the mobile network code(MNC) of the carrier to which the phone is connected.

Returns

mnc.

Note

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

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

Retrieves the phone number for the SIM subscription.

Returns

PhoneNumber.

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

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

Returns

SIM slotId.

5.9.1.1.2.9 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.

5.9.1.1.2.10 virtual std::string telux::tel::ISubscription::getGID1 () [pure virtual]

Retrieves the GID1(group identifier level1) on the SIM. It represents identifier for particular SIM and ME associations. It can be used to identify a group of SIMs for a particular application. Defined in 3GPP Spec 131.102 section 4.2.10

Returns

GID1 content in hex format.

Note

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

5.9.1.1.2.11 virtual std::string telux::tel::ISubscription::getGID2 () [pure virtual]

Retrieves the GID2(group identifier level2) content on the SIM. It represents identifier for particular SIM and ME associations. It can be used to identify a group of SIMs for a particular application. Defined in 3GPP Spec 131.102 section 4.2.11

Returns

GID2 content in hex format.

Note

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

5.9.1.2 class telux::tel::ISubscriptionListener

A listener class for receiving device subscription information. The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

Public member functions

- virtual void [onSubscriptionInfoChanged](#) (std::shared_ptr< [ISubscription](#) > subscription)
- virtual void [onNumberOfSubscriptionsChanged](#) (int count)
- virtual [~ISubscriptionListener](#) ()

5.9.1.2.1 Constructors and Destructors

5.9.1.2.1.1 virtual telux::tel::ISubscriptionListener::~~ISubscriptionListener () [virtual]

5.9.1.2.2 Member Function Documentation

5.9.1.2.2.1 virtual void telux::tel::ISubscriptionListener::onSubscriptionInfoChanged (std::shared_ptr< [ISubscription](#) > *subscription*) [virtual]

This function is called whenever there is a change in Subscription details.

Parameters

in	<i>subscription</i>	Pointer to ISubscription 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.

<code>in</code>	<code>count</code>	count of subscription

5.9.1.3 class telux::tel::ISubscriptionManager

Public member functions

- virtual bool `isSubsystemReady` ()=0
- virtual std::future< bool > `onSubsystemReady` ()=0
- virtual std::shared_ptr
< `ISubscription` > `getSubscription` (int slotId=DEFAULT_SLOT_ID, telux::common::Status
*status=nullptr)=0
- virtual std::vector
< std::shared_ptr
< `ISubscription` > > `getAllSubscriptions` (telux::common::Status *status=nullptr)=0
- virtual telux::common::Status `registerListener` (std::weak_ptr< `ISubscriptionListener` > listener)=0
- virtual telux::common::Status `removeListener` (std::weak_ptr< `ISubscriptionListener` > listener)=0
- virtual `~ISubscriptionManager` ()

5.9.1.3.1 Constructors and Destructors

5.9.1.3.1.1 virtual telux::tel::ISubscriptionManager::~~ISubscriptionManager () [virtual]

5.9.1.3.2 Member Function Documentation

5.9.1.3.2.1 virtual bool telux::tel::ISubscriptionManager::isSubsystemReady () [pure virtual]

Checks the status of SubscriptionManager and returns the result.

Returns

If true then SubscriptionManager is ready for service.

5.9.1.3.2.2 virtual std::future<bool> telux::tel::ISubscriptionManager::onSubsystemReady () [pure virtual]

Wait for Subscription subsystem to be ready.

Returns

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

5.9.1.3.2.3 `virtual std::shared_ptr<ISubscription> telux::tel::ISubscriptionManager::getSubscription ( int slotId = DEFAULT_SLOT_ID, telux::common::Status * status = nullptr ) [pure virtual]`

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

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> > telux::tel::ISubscriptionManager::getAllSubscriptions ( telux::common::Status * status = nullptr ) [pure virtual]`

Get all the subscription details of the device.

Parameters

out	<i>status</i>	Status of getAllSubscriptions i.e. success or suitable status code.

Returns

list of [ISubscription](#) objects.

5.9.1.3.2.5 `virtual telux::common::Status telux::tel::ISubscriptionManager::registerListener ( std::weak_ptr< ISubscriptionListener > listener ) [pure virtual]`

Register a listener for Subscription events.

Parameters

in	<i>listener</i>	Pointer to ISubscriptionListener 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.

in	<i>listener</i>	Pointer to ISubscriptionListener 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
RatMask	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
InUseStatus	inUse	In-use status of network operator
RoamingStatus	roaming	Roaming status of network operator
Forbidden-Status	forbidden	Forbidden status of network operator
PreferredStatus	preferred	Preferred status of network operator

5.10.1.3 class telux::tel::INetworkSelectionManager

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

in	<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 when automatic registration is performed by the device. Higher priority networks should appear first in the list.

Parameters

in	<i>preferredNetworks-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 INetworkSelectionListener 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 INetworkSelectionListener 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.

<code>in</code>	<code>mode</code>	Network selection mode NetworkSelectionMode

5.10.2 Enumeration Type Documentation

5.10.2.1 enum telux::tel::RatType

Defines network RAT type for preferred networks. Each value represents corresponding bit for RatMask bitset.

Enumerator

UMTS UMTS

LTE LTE

LTE

GSM GSM

GSM

NR5G NR5G

5.10.2.2 enum telux::tel::NetworkSelectionMode

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

Defines in-use status of network operator

Enumerator

UNKNOWN Unknown

CURRENT_SERVING Current serving

AVAILABLE Available

5.10.2.4 enum telux::tel::RoamingStatus

Defines roaming status of network operator

Enumerator

UNKNOWN Unknown

HOME Home

ROAM Roaming

5.10.2.5 enum telux::tel::ForbiddenStatus

Defines forbidden status of network operator

Enumerator

UNKNOWN Unknown

FORBIDDEN Forbidden

NOT_FORBIDDEN Not forbidden

5.10.2.6 enum telux::tel::PreferredStatus

Defines preferred status of network operator

Enumerator

UNKNOWN Unknown

PREFERRED Preferred

NOT_PREFERRED Not preferred

5.11 Serving System

Serving System Manager class provides the interface to request and set service domain preference and radio access technology mode preference for searching and registering (CS/PS domain, RAT and operation mode)

5.11.1 Data Structure Documentation

5.11.1.1 struct telux::tel::DcStatus

Defines Dual Connectivity status

Data fields

Type	Field	Description
Endc-Availability	endc-Availability	ENDC availability
DcnrRestriction	dcnrRestriction	DCNR restriction

5.11.1.2 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::tel::DcStatus](#) [getDcStatus](#) ()=0
- virtual [telux::common::Status](#) [requestNetworkTime](#) ([NetworkTimeResponseCallback](#) callback)=0
- virtual [telux::common::Status](#) [registerListener](#) (std::weak_ptr< [IServingSystemListener](#) > listener)=0
- virtual [telux::common::Status](#) [deregisterListener](#) (std::weak_ptr< [IServingSystemListener](#) > listener)=0
- virtual [~IServingSystemManager](#) ()

5.11.1.2.1 Constructors and Destructors

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

Destructor of [IServingSystemManager](#)

5.11.1.2.2 Member Function Documentation

5.11.1.2.2.1 `virtual bool telux::tel::IServingSystemManager::isSubsystemReady ( ) [pure virtual]`

Checks the status of serving subsystem and returns the result.

Returns

True if serving subsystem is ready for service otherwise false.

5.11.1.2.2.2 `virtual std::future<bool> telux::tel::IServingSystemManager::onSubsystemReady ( ) [pure virtual]`

Wait for serving subsystem to be ready.

Returns

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

5.11.1.2.2.3 `virtual telux::common::Status telux::tel::IServingSystemManager::setRatPreference ( RatPreference ratPref, common::ResponseCallback callback = nullptr ) [pure virtual]`

Set the preferred radio access technology mode that the device should use to acquire service.

Parameters

in	<i>ratPref</i>	Radio access technology mode preference.
in	<i>callback</i>	Callback function to get the response of set RAT mode preference.

Returns

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

5.11.1.2.2.4 `virtual telux::common::Status telux::tel::IServingSystemManager::requestRatPreference ( RatPreferenceCallback callback ) [pure virtual]`

Request for preferred radio access technology mode.

in	<i>callback</i>	Callback function to get the response of request preferred RAT mode.

Returns

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

5.11.1.2.2.5 `virtual telux::common::Status telux::tel::IServingSystemManager::setServiceDomainPreference ( ServiceDomainPreference serviceDomain, common::ResponseCallback callback = nullptr ) [pure virtual]`

Initiate service domain preference like CS, PS or CS_PS and receive the response asynchronously.

Parameters

in	<i>serviceDomain</i>	ServiceDomainPreference .
in	<i>callback</i>	Callback function to get the response of set service domain preference request.

Returns

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

5.11.1.2.2.6 `virtual telux::common::Status telux::tel::IServingSystemManager::requestServiceDomainPreference ( ServiceDomainPreferenceCallback callback ) [pure virtual]`

Request for Service Domain Preference asynchronously.

Parameters

in	<i>callback</i>	Callback function to get the response of request service domain preference.

Returns

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

5.11.1.2.2.7 `virtual telux::tel::DcStatus telux::tel::IServingSystemManager::getDcStatus ( ) [pure virtual]`

Request for Dual Connectivity status on 5G NR.

Returns

[DcStatus](#)

Note

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

5.11.1.2.2.8 **virtual telux::common::Status telux::tel::IServingSystemManager::requestNetworkTime (NetworkTimeResponseCallback *callback*) [pure virtual]**

Get network time information asynchronously.

Parameters

in	<i>callback</i>	Callback function to get the response of get network time information request.

Returns

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

Note

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

5.11.1.2.2.9 **virtual telux::common::Status telux::tel::IServingSystemManager::registerListener (std::weak_ptr< IServingSystemListener > *listener*) [pure virtual]**

Register a listener for specific updates from serving system.

Parameters

in	<i>listener</i>	Pointer of IServingSystemListener object that processes the notification

Returns

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

5.11.1.2.2.10 **virtual telux::common::Status telux::tel::IServingSystemManager::deregisterListener (std::weak_ptr< IServingSystemListener > *listener*) [pure virtual]**

Deregister the previously added listener.

Parameters

in	<i>listener</i>	Previously registered IServingSystemListener that needs to be removed

Returns

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

5.11.1.3 class telux::tel::IServingSystemListener

Listener class for getting radio access technology mode preference change notification.

The listener method can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe.

Public member functions

- virtual void [onRatPreferenceChanged](#) ([RatPreference](#) preference)
- virtual void [onServiceDomainPreferenceChanged](#) ([ServiceDomainPreference](#) preference)
- virtual void [onDcStatusChanged](#) ([DcStatus](#) dcStatus)
- virtual void [onNetworkTimeChanged](#) ([NetworkTimeInfo](#) info)
- virtual [~IServingSystemListener](#) ()

5.11.1.3.1 Constructors and Destructors

5.11.1.3.1.1 virtual telux::tel::IServingSystemListener::~~IServingSystemListener () [virtual]

Destructor of [IServingSystemListener](#)

5.11.1.3.2 Member Function Documentation

5.11.1.3.2.1 virtual void telux::tel::IServingSystemListener::onRatPreferenceChanged ([RatPreference](#) *preference*) [virtual]

This function is called whenever RAT mode preference is changed.

Parameters

in	<i>preference</i>	RatPreference

5.11.1.3.2.2 virtual void telux::tel::IServingSystemListener::onServiceDomainPreferenceChanged ([ServiceDomainPreference](#) *preference*) [virtual]

This function is called whenever service domain preference is changed.

Parameters

in	<i>preference</i>	ServiceDomainPreference

5.11.1.3.2.3 **virtual void telux::tel::IServingSystemListener::onDcStatusChanged (DcStatus *dcStatus*) [virtual]**

This function is called whenever the Dual Connectivity status is changed on 5G NR.

Parameters

in	<i>dcStatus</i>	DcStatus

Note

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

5.11.1.3.2.4 **virtual void telux::tel::IServingSystemListener::onNetworkTimeChanged (NetworkTimeInfo *info*) [virtual]**

This function is called whenever network time information is changed.

Parameters

in	<i>info</i>	Network time information NetworkTimeInfo

Note

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

5.11.2 Enumeration Type Documentation

5.11.2.1 **enum telux::tel::ServiceDomainPreference**

Defines service domain preference

Enumerator

UNKNOWN
CS_ONLY Circuit-switched only
PS_ONLY Packet-switched only
CS_PS Circuit-switched and packet-switched

5.11.2.2 **enum telux::tel::RatPrefType**

Defines the radio access technology mode preference.

Enumerator

PREF_CDMA_1X CDMA_1X
PREF_CDMA_EVDO CDMA_EVDO
PREF_GSM GSM
PREF_WCDMA WCDMA

PREF_LTE LTE
PREF_TDSCDMA TDSCDMA
PREF_NR5G NR5G

5.11.2.3 enum telux::tel::EndcAvailability

Defines ENDC(E-UTRAN New Radio-Dual Connectivity) Availability status on 5G NR

Enumerator

UNKNOWN Status unknown
AVAILABLE ENDC is Available
UNAVAILABLE ENDC is not Available

5.11.2.4 enum telux::tel::DcnrRestriction

Defines DCNR(Dual Connectivity with NR) Restriction status on 5G NR

Enumerator

UNKNOWN Status unknown
RESTRICTED DCNR is Rescticted
UNRESTRICTED DCNR is not Restricted

5.12 Common

This section contains APIs related to Command Callbacks, Error Codes and [Version](#) information.

5.12.1 Data Structure Documentation

5.12.1.1 class `telux::common::ICommandCallback`

Base command callback class is responsible for single shot asynchronous callback. This callback will be invoked only once when the operation succeeds or fails.

Public member functions

- virtual [~ICommandCallback](#) ()

5.12.1.1.1 Constructors and Destructors

5.12.1.1.1 virtual `telux::common::ICommandCallback::~ICommandCallback ( )` [virtual]

5.12.1.2 class `telux::common::ICommandResponseCallback`

General command response callback for most of the requests, client needs to implement this interface to get single shot response.

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

Public member functions

- virtual void [commandResponse](#) ([ErrorCode](#) error)=0
- virtual [~ICommandResponseCallback](#) ()

5.12.1.2.1 Constructors and Destructors

5.12.1.2.1.1 virtual `telux::common::ICommandResponseCallback::~ICommandResponseCallback ( )` [virtual]

5.12.1.2.2 Member Function Documentation

5.12.1.2.2.1 virtual void `telux::common::ICommandResponseCallback::commandResponse ( ErrorCode error )` [pure virtual]

This function is called with the response to the command operation.

Parameters

in	<i>error</i>	- ErrorCode

5.12.1.3 class telux::common::DeviceConfig

Static Public Member Functions

- static bool [isMultiSimSupported](#) ()

5.12.1.3.1 Member Function Documentation

5.12.1.3.1.1 static bool telux::common::DeviceConfig::isMultiSimSupported () [static]

Check whether multi SIM support available.

Returns

bool to determine multi SIM support

5.12.1.4 class telux::common::Log

Static Public Member Functions

- template<typename... MessageArgs>
static void [logMessage](#) ([LogLevel](#) logLevel, const std::string &fileName, const std::string &lineNo, const int &component, MessageArgs...params)

5.12.1.4.1 Member Function Documentation

5.12.1.4.1.1 template<typename... MessageArgs> static void telux::common::Log::logMessage (LogLevel *logLevel*, const std::string & *fileName*, const std::string & *lineNo*, const int & *component*, MessageArgs... *params*) [static]

5.12.1.5 struct telux::common::SdkVersion

Structure of major, minor and patch version

Data fields

Type	Field	Description
int	major	Major Version : This number will be incremented whenever significant changes or features are introduced
int	minor	Minor Version : This number will be incremented when smaller features with some new APIs are introduced.
int	patch	Patch Version : If the release only contains bug fixes, but no API change then the patch version would be incremented.

5.12.1.6 class telux::common::Version

Provides version of SDK.

Static Public Member Functions

- static std::string [getReleaseName](#) ()
- static [SdkVersion](#) [getSdkVersion](#) ()

5.12.1.6.1 Member Function Documentation

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

Get the release name.

Returns

String contains release name

5.12.1.6.1.2 static SdkVersion telux::common::Version::getSdkVersion () [static]

Get the Telematics SDK version, for example: 01.00.00

Returns

[SdkVersion](#) structure of major, minor and patch version

5.12.2 Enumeration Type Documentation

5.12.2.1 enum telux::common::Status

Defines all the status codes that all Telematics SDK APIs can return

Enumerator

SUCCESS API processing is successful, returned parameters are valid

FAILED API processing failure.

NOCONNECTION Connection to Socket server has not been established

NOSUBSCRIPTION Subscription not available

INVALIDPARAM Input parameters are invalid

INVALIDSTATE Invalid State

NOTREADY Subsystem is not ready

NOTALLOWED Operation not allowed

NOTIMPLEMENTED Functionality not implemented

CONNECTIONLOST Connection to Socket server lost

EXPIRED Expired

ALREADY Already registered handler

NOSUCH No such object

NOTSUPPORTED Not supported on target platform

NOMEMORY Not sufficient memory to process the request

5.12.2.2 enum telux::common::ErrorCode

Generic Error code for each API responses

Enumerator

SUCCESS No error

RADIO_NOT_AVAILABLE If radio did not start or is resetting

GENERIC_FAILURE Generic Failure

PASSWORD_INCORRECT For PIN/PIN2 methods only

SIM_PIN2 Operation requires SIM PIN2 to be entered

SIM_PUK2 Operation requires SIM PIN2 to be entered

REQUEST_NOT_SUPPORTED Not Supported request

CANCELLED Cancelled

OP_NOT_ALLOWED_DURING_VOICE_CALL Data operation are not allowed during voice call on a Class C GPRS device

OP_NOT_ALLOWED_BEFORE_REG_TO_NW Data operation are not allowed before device registers in network

SMS_SEND_FAIL_RETRY Fail to send SMS and need retry

SIM_ABSENT Fail to set the location where CDMA subscription shall be retrieved because of SIM or RUIM are absent

SUBSCRIPTION_NOT_AVAILABLE Fail to find CDMA subscription from specified location

MODE_NOT_SUPPORTED Hardware does not support preferred network type

FDN_CHECK_FAILURE Command failed because recipient is not on FDN list

ILLEGAL_SIM_OR_ME Network selection failed due to illegal SIM or ME

MISSING_RESOURCE No logical channel available

NO_SUCH_ELEMENT Application not found on SIM

DIAL_MODIFIED_TO USSD DIAL request modified to USSD

DIAL_MODIFIED_TO_SS DIAL request modified to SS

DIAL_MODIFIED_TO_DIAL DIAL request modified to DIAL with different data

USSD_MODIFIED_TO_DIAL USSD request modified to DIAL

USSD_MODIFIED_TO_SS USSD request modified to SS

USSD_MODIFIED_TO_USSD USSD request modified to different USSD request

SS_MODIFIED_TO_DIAL SS request modified to DIAL

SS_MODIFIED_TO_USSD SS request modified to USSD

SUBSCRIPTION_NOT_SUPPORTED Subscription not supported

SS_MODIFIED_TO_SS SS request modified to different SS request

LCE_NOT_SUPPORTED LCE service not supported

NO_MEMORY Not sufficient memory to process the request

INTERNAL_ERR Hit unexpected vendor internal error scenario

SYSTEM_ERR Hit platform or system error

MODEM_ERR Hit unexpected modem error

INVALID_STATE Unexpected request for the current state

NO_RESOURCES Not sufficient resource to process the request

SIM_ERR Received error from SIM card

INVALID_ARGUMENTS Received invalid arguments in request

INVALID_SIM_STATE Cannot process the request in current SIM state

INVALID_MODEM_STATE Cannot process the request in current Modem state

INVALID_CALL_ID Received invalid call id in request

NO_SMS_TO_ACK ACK received when there is no SMS to ack

NETWORK_ERR Received error from network

REQUEST_RATE_LIMITED Operation denied due to overly-frequent requests

SIM_BUSY SIM is busy

SIM_FULL The target EF is full

NETWORK_REJECT Request is rejected by network

OPERATION_NOT_ALLOWED Not allowed the request now

EMPTY_RECORD The request record is empty

INVALID_SMS_FORMAT Invalid SMS format

ENCODING_ERR Message not encoded properly

INVALID_SMSC_ADDRESS SMSC address specified is invalid

NO_SUCH_ENTRY No such entry present to perform the request

NETWORK_NOT_READY Network is not ready to perform the request

NOT_PROVISIONED Device does not have this value provisioned

NO_SUBSCRIPTION Device does not have subscription

NO_NETWORK_FOUND Network cannot be found

DEVICE_IN_USE Operation cannot be performed because the device is currently in use

ABORTED Operation aborted

INCOMPATIBLE_STATE Operation cannot be performed because the device is in incompatible state

NO_EFFECT Given request had to no effect

DEVICE_NOT_READY Device not ready

MISSING_ARGUMENTS Missing one or more arguments

PIN_PERM_BLOCKED PIN is permanently blocked. The SIM is unusable.

PIN_BLOCKED PIN is blocked. Unblock operation must be issued.

MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission

INTERNAL Internal error

CLIENT_IDS_EXHAUSTED Client IDs exhausted

UNABORTABLE_TRANSACTION The specified transaction could not be aborted

INVALID_CLIENT_ID Could not find client's request

NO_THRESHOLDS No thresholds specified in enable signal strength

INVALID_HANDLE Invalid client handle was received

INVALID_PROFILE Invalid profile index specified

INVALID_PINID PIN in the request is invalid.

INCORRECT_PIN PIN in the request is incorrect.

CALL_FAILED Call origination failed in the lower layers

OUT_OF_CALL Request issued when packet data session disconnected

MISSING_ARG TLV was missing in the request.

ARG_TOO_LONG Path in the request was too long.

INVALID_TX_ID The transaction ID supplied in the request does not match any pending transaction
i.e. either the transaction was not received or it is already executed by the device

OP_NETWORK_UNSUPPORTED Selected operation is not supported by the network

OP_DEVICE_UNSUPPORTED Operation is not supported by device or SIM card

NO_FREE_PROFILE Maximum number of profiles are stored in the device and there is no more storage available to create a new profile

INVALID_PDP_TYPE PDP type specified is not supported

INVALID_TECH_PREF Invalid technology preference

INVALID_PROFILE_TYPE Invalid profile type is specified

INVALID_SERVICE_TYPE Invalid service type

INVALID_REGISTER_ACTION Invalid register action value specified in request

INVALID_PS_ATTACH_ACTION Invalid PS attach action value specified in request

AUTHENTICATION_FAILED Authentication error.

SIM_NOT_INITIALIZED PIN is not yet initialized because the SIM initialization has not finished. Try the PIN operation later.

MAX_QOS_REQUESTS_IN_USE Maximum QoS requests in use

INCORRECT_FLOW_FILTER Incorrect flow filter

NETWORK_QOS_UNAWARE Network QoS unaware

INVALID_ID Invalid call ID was sent in the request

REQUESTED_NUM_UNSUPPORTED Requested message ID is not supported by the currently running software

INTERFACE_NOT_FOUND Cannot retrieve the FMC interface

FLOW_SUSPENDED Flow suspended

INVALID_DATA_FORMAT Invalid data format

GENERAL General error

UNKNOWN Unknown error

INVALID_ARG Parameters passed as input were invalid

INVALID_INDEX MIP profile index is not within the valid range

NO_ENTRY No message exists at the specified memory storage designation

DEVICE_STORAGE_FULL Memory storage specified in the request is full

CAUSE_CODE There was an error in the request

MESSAGE_NOT_SENT Message could not be sent

MESSAGE_DELIVERY_FAILURE Message could not be delivered

INVALID_MESSAGE_ID Message ID specified for the message is invalid

ENCODING Message is not encoded properly

AUTHENTICATION_LOCK Maximum number of authentication failures has been reached

INVALID_TRANSITION Selected operating mode transition from the current operating mode is invalid

NOT_A_MCAST_IFACE Not a MCAST interface

MAX_MCAST_REQUESTS_IN_USE MCAST request in use

INVALID_MCAST_HANDLE An invalid MCAST handle

INVALID_IP_FAMILY_PREF IP family preference is invalid

SESSION_INACTIVE Session inactive

SESSION_INVALID Session not valid

SESSION_OWNERSHIP Session ownership error

INSUFFICIENT_RESOURCES Response is longer than the maximum supported size

DISABLED Disabled

INVALID_OPERATION Device is not expecting the request.

INVALID_QMI_CMD Invalid QMI command

TPDU_TYPE Message in memory contains a TPDU type that cannot be read

SMSC_ADDR SMSC address specified is invalid

INFO_UNAVAILABLE Information is not available

SEGMENT_TOO_LONG PRL segment size is too large

SEGMENT_ORDER PRL segment order is incorrect

BUNDLING_NOT_SUPPORTED Bundling not supported

OP_PARTIAL_FAILURE Some personalization codes were set but an error prevented

POLICY_MISMATCH Network policy does not match a valid NAT

SIM_FILE_NOT_FOUND File is not present on the card.

EXTENDED_INTERNAL Error from the the DS profile module, the extended error

ACCESS_DENIED Access to the requested file is denied. This can occur when there is an attempt to access a PIN-protected file.

HARDWARE_RESTRICTED Selected operating mode is invalid with the current wireless disable setting

ACK_NOT_SENT ACK could not be sent

INJECT_TIMEOUT Inject timeout

FDN_RESTRICT FDN restriction

SUPS_FAILURE_CAUSE Indicates supplementary services failure information;

NO_RADIO Radio is not available

NOT_SUPPORTED Operation is not supported

CARD_CALL_CONTROL_FAILED SIM/R-UIM call control failed

NETWORK_ABORTED Operation was released abruptly by the network

MSG_BLOCKED Message blocked

INVALID_SESSION_TYPE Invalid session type

INVALID_PB_TYPE Invalid Phone Book type

NO_SIM Action is being performed on a SIM that is not initialized.

PB_NOT_READY Phone Book not ready

PIN_RESTRICTION PIN restriction

PIN2_RESTRICTION PIN2 restriction

PUK_RESTRICTION PUK restriction

PUK2_RESTRICTION PUK2 restriction

PB_ACCESS_RESTRICTED Phone Book access restricted

PB_DELETE_IN_PROG Phone Book delete in progress

PB_TEXT_TOO_LONG Phone Book text too long

PB_NUMBER_TOO_LONG Phone Book number too long

PB_HIDDEN_KEY_RESTRICTION Phone Book hidden key restriction

PB_NOT_AVAILABLE Phone Book not available

DEVICE_MEMORY_ERROR Device memory error

NO_PERMISSION No permission

TOO_SOON Too soon

TIME_NOT_ACQUIRED Time not acquired

OP_IN_PROGRESS Operation is in progress

DS_PROFILE_REG_RESULT_FAIL General failure

DS_PROFILE_REG_RESULT_ERR_INVAL_HNDL Request contains an invalid profile handle

DS_PROFILE_REG_RESULT_ERR_INVAL_OP Invalid operation was requested

DS_PROFILE_REG_RESULT_ERR_INVAL_PROFILE_TYPE Request contains an invalid technology type

DS_PROFILE_REG_RESULT_ERR_INVAL_PROFILE_NUM Request contains an invalid profile number

DS_PROFILE_REG_RESULT_ERR_INVAL_IDENT Request contains an invalid profile identifier

DS_PROFILE_REG_RESULT_ERR_INVAL 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_INVAL_SUBS_ID Request contains an invalid subscription identifier

DS_PROFILE_REG_INVALID_PROFILE_FAMILY Request contains an invalid profile family
DS_PROFILE_REG_PROFILE_VERSION_MISMATCH [Version](#) mismatch
REG_RESULT_ERR_OUT_OF_MEMORY Out of memory
DS_PROFILE_REG_RESULT_ERR_FILE_ACCESS File access error
DS_PROFILE_REG_RESULT_ERR_EOF End of field
REG_RESULT_ERR_VALID_FLAG_NOT_SET A valid flag is not set
REG_RESULT_ERR_OUT_OF_PROFILES Out of profiles
REG_RESULT_NO_EMERGENCY_PDN_SUPPORT No emergency PDN support
DS_PROFILE_3GPP_INVALID_PROFILE_FAMILY Request contains an invalid 3GPP profile family
DS_PROFILE_3GPP_ACCESS_ERR Error was encountered while accessing the 3GPP profiles
DS_PROFILE_3GPP_CONTEXT_NOT_DEFINED Specified 3GPP profile does not have a valid context
DS_PROFILE_3GPP_VALID_FLAG_NOT_SET Specified 3GPP profile is marked invalid
DS_PROFILE_3GPP_READ_ONLY_FLAG_SET Specified 3GPP profile is marked read-only
DS_PROFILE_3GPP_ERR_OUT_OF_PROFILES Creation of a new 3GPP profile failed because the limit of 16 profiles has already been reached
DS_PROFILE_3GPP2_ERR_INVALID_IDENT_FOR_PROFILE Invalid profile identifier was received as part of the 3GPP2 profile modification request
DS_PROFILE_3GPP2_ERR_OUT_OF_PROFILE Creation of a new 3GPP2 profile failed because the limit has already been reached
INTERNAL_ERROR Internal error
SERVICE_ERROR Service error
TIMEOUT_ERROR Timeout error
EXTENDED_ERROR Extended error
PORT_NOT_OPEN_ERROR Port not open
MEMCOPY_ERROR Memory copy error
INVALID_TRANSACTION Invalid transaction
ALLOCATION_FAILURE Allocation failure
TRANSPORT_ERROR Transport error
PARAM_ERROR Parameter error
INVALID_CLIENT Invalid client
FRAMEWORK_NOT_READY Framework not ready
INVALID_SIGNAL Invalid signal
TRANSPORT_BUSY_ERROR Transport busy error
SUBSYSTEM_UNAVAILABLE Underlying service currently unavailable
OPERATION_TIMEOUT Timeout error

5.12.2.3 enum telux::common::ServiceStatus

Service status.

Enumerator

SERVICE_UNAVAILABLE
SERVICE_AVAILABLE
SERVICE_FAILED

5.12.2.4 enum telux::common::LogLevel

Indicates supported logging levels.

Enumerator**LEVEL_NONE****LEVEL_PERF** Prints messages with nanoseconds precision timestamp**LEVEL_ERROR** Prints perf and error messages only**LEVEL_WARNING** Prints perf, error and warning messages**LEVEL_INFO** Prints perf, errors, warning and information messages**LEVEL_DEBUG** Full logging including debug messages

5.13 C-V2X

This section contains APIs related to Cellular-V2X operation.

5.13.1 Data Structure Documentation

5.13.1.1 class `telux::cv2x::ICv2xConfigListener`

Listeners for `ICv2xConfig` must implement this interface.

Public member functions

- virtual void `onConfigChanged` (const `ConfigEventInfo` &info)
- virtual `~ICv2xConfigListener` ()

5.13.1.1.1 Constructors and Destructors

5.13.1.1.1.1 virtual `telux::cv2x::ICv2xConfigListener::~~ICv2xConfigListener` () [`virtual`]

Destructor for `ICv2xConfigListener`

5.13.1.1.2 Member Function Documentation

5.13.1.1.2.1 virtual void `telux::cv2x::ICv2xConfigListener::onConfigChanged` (const `ConfigEventInfo` & *info*) [`virtual`]

Called when CV2X configuration has changed in the below scenarios:

1. The specified configuration source has expired.
2. The active configuration source has changed to the specified configuration source type due to the expiration of the configuration source being used.
3. The specified configuration source has been updated.

Parameters

<i>in</i>	<i>info</i>	- Information of CV2X configuration event.

5.13.1.2 class `telux::cv2x::ICv2xConfig`

`Cv2xConfig` provide operations to update or request `cv2x` configuration.

Public member functions

- virtual `~ICv2xConfig` ()
- virtual bool `isReady` ()=0
- virtual std::future< bool > `onReady` ()=0

- virtual [telux::common::Status updateConfiguration](#) (const std::string &configFilePath, [telux::common::ResponseCallback](#) cb)=0
- virtual [telux::common::Status retrieveConfiguration](#) (const std::string &configFilePath, [telux::common::ResponseCallback](#) cb)=0
- virtual [telux::common::Status registerListener](#) (std::weak_ptr< [ICv2xConfigListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak_ptr< [ICv2xConfigListener](#) > listener)=0

5.13.1.2.1 Constructors and Destructors

5.13.1.2.1.1 virtual [telux::cv2x::ICv2xConfig::~~ICv2xConfig](#) () [**virtual**]

5.13.1.2.2 Member Function Documentation

5.13.1.2.2.1 virtual bool [telux::cv2x::ICv2xConfig::isReady](#) () [**pure virtual**]

Checks if the Cv2x Config Manager is ready.

Returns

True if Cv2x Config is ready for service, otherwise returns false.

5.13.1.2.2.2 virtual std::future<bool> [telux::cv2x::ICv2xConfig::onReady](#) () [**pure virtual**]

Wait for Cv2x Config to be ready.

Returns

A future that caller can wait on to be notified when Cv2x Radio Manager is ready.

5.13.1.2.2.3 virtual [telux::common::Status telux::cv2x::ICv2xConfig::updateConfiguration](#) (const std::string & *configFilePath*, [telux::common::ResponseCallback](#) *cb*) [**pure virtual**]

Updates CV2X configuration. Requires CV2X TX/RX radio status be Inactive. If CV2X radio status is Active or Suspended, call [ICv2xRadioManager::stopCv2x](#) before trying to update configuration. The functionality of V2X configuration expiration is supported by adding an expiration leaf to the V2X configuration file passed in. When the active configuration expires, the system fallbacks to a lower priority V2X configuration [ConfigSourceType](#) if existed. If the V2X status is active, it changes to suspended when the active V2X configuration expires and then changes to active after the system fallbacks to a lower priority V2X configuration or changes to inactive if no V2X configuration is available.

Parameters

in	<i>configFilePath</i>	- Path to config file. This is the fully qualified file path including the name of the file.
in	<i>cb</i>	- Callback that is invoked when the send is complete. This may be null.

5.13.1.2.2.4 `virtual telux::common::Status telux::cv2x::ICv2xConfig::retrieveConfiguration ( const std::string & configFilePath, telux::common::ResponseCallback cb ) [pure virtual]`

Retrieve active CV2X configuration. The calling application should have write access to the path specified by . And if the v2x configuration retrieval request succeed, the file specified by will be created and filled with the configuration contents. Otherwise, no file will be created.

Parameters

in	<i>configFilePath</i>	- Path to config file. This is the fully qualified file path including the name of the file.
in	<i>cb</i>	- Callback that is invoked when the configuration retrieval is complete. This may be null.

5.13.1.2.2.5 `virtual telux::common::Status telux::cv2x::ICv2xConfig::registerListener ( std::weak_ptr< ICv2xConfigListener > listener ) [pure virtual]`

Registers a listener for this [ICv2xConfig](#).

Parameters

in	<i>listener</i>	- Listener that implements ICv2xConfigListener interface.

5.13.1.2.2.6 `virtual telux::common::Status telux::cv2x::ICv2xConfig::deregisterListener ( std::weak_ptr< ICv2xConfigListener > listener ) [pure virtual]`

Deregisters a listener for this [ICv2xConfig](#).

Parameters

in	<i>listener</i>	- Previously registered ICv2xConfigListener that is to be deregistered.

5.13.1.3 `class telux::cv2x::Cv2xFactory`

[Cv2xFactory](#) is the factory that creates the Cv2x Radio.

Public member functions

- `std::shared_ptr< ICv2xRadioManager > getCv2xRadioManager ()`
- `std::shared_ptr< ICv2xConfig > getCv2xConfig ()`
- `std::shared_ptr< ICv2xThrottleManager > getCv2xThrottleManager ()`

Static Public Member Functions

- static [Cv2xFactory](#) & [getInstance](#) ()

5.13.1.3.1 Member Function Documentation

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

Get [Cv2xFactory](#) instance

Returns

Reference to [Cv2xFactory](#) singleton.

5.13.1.3.1.2 std::shared_ptr<ICv2xRadioManager> [telux::cv2x::Cv2xFactory::getCv2xRadioManager](#) ()

Get [Cv2xRadioManager](#) instance.

Returns

shared pointer to [Cv2x Radio Manager](#) upon success. nullptr otherwise.

5.13.1.3.1.3 std::shared_ptr<ICv2xConfig> [telux::cv2x::Cv2xFactory::getCv2xConfig](#) ()

Get [Cv2xConfig](#) instance.

Returns

shared pointer to [Cv2x Config](#) upon success. nullptr otherwise.

5.13.1.3.1.4 std::shared_ptr<ICv2xThrottleManager> [telux::cv2x::Cv2xFactory::getCv2xThrottleManager](#) ()

Get [Cv2xThrottleManager](#) instance.

Returns

shared pointer to [Cv2x ThrottleManager](#) upon success. nullptr otherwise.

5.13.1.4 class [telux::cv2x::ICv2xRadio](#)

This is class encapsulates a [Cv2xRadio](#) interface.

Returned from [getCv2xRadio](#) in [Cv2xFactory](#)

Public member functions

- virtual `Cv2xRadioCapabilities` `getCapabilities ()` const =0
- virtual bool `isReady ()` const =0
- virtual bool `isInitialized ()` const =0
- virtual std::future
 < `telux::common::Status` > `onReady ()`=0
- virtual `telux::common::Status` `registerListener` (std::weak_ptr< `ICv2xRadioListener` > listener)=0
- virtual `telux::common::Status` `deregisterListener` (std::weak_ptr< `ICv2xRadioListener` > listener)=0
- virtual `telux::common::Status` `createRxSubscription` (`TrafficIpType` ipType, uint16_t port,
 `CreateRxSubscriptionCallback` cb, std::shared_ptr< std::vector< uint32_t >> idList=nullptr)=0
- virtual `telux::common::Status` `createTxSpsFlow` (`TrafficIpType` ipType, uint32_t serviceId, const
 `SpsFlowInfo` &spsInfo, uint16_t spsSrcPort, bool eventSrcPortValid, uint16_t eventSrcPort,
 `CreateTxSpsFlowCallback` cb)=0
- virtual `telux::common::Status` `createTxEventFlow` (`TrafficIpType` ipType, uint32_t serviceId, uint16_t
 eventSrcPort, `CreateTxEventFlowCallback` cb)=0
- virtual `telux::common::Status` `createTxEventFlow` (`TrafficIpType` ipType, uint32_t serviceId, const
 `EventFlowInfo` &flowInfo, uint16_t eventSrcPort, `CreateTxEventFlowCallback` cb)=0
- virtual `telux::common::Status` `closeRxSubscription` (std::shared_ptr< `ICv2xRxSubscription` > rxSub,
 `CloseRxSubscriptionCallback` cb)=0
- virtual `telux::common::Status` `closeTxFlow` (std::shared_ptr< `ICv2xTxFlow` > txFlow,
 `CloseTxFlowCallback` cb)=0
- virtual `telux::common::Status` `changeSpsFlowInfo` (std::shared_ptr< `ICv2xTxFlow` > txFlow, const
 `SpsFlowInfo` &spsInfo, `ChangeSpsFlowInfoCallback` cb)=0
- virtual `telux::common::Status` `requestSpsFlowInfo` (std::shared_ptr< `ICv2xTxFlow` > txFlow,
 `RequestSpsFlowInfoCallback` cb)=0
- virtual `telux::common::Status` `changeEventFlowInfo` (std::shared_ptr< `ICv2xTxFlow` > txFlow,
 const `EventFlowInfo` &flowInfo, `ChangeEventFlowInfoCallback` cb)=0
- virtual `telux::common::Status` `requestCapabilities` (`RequestCapabilitiesCallback` cb)=0
- virtual `telux::common::Status` `requestDataSessionSettings` (`RequestDataSessionSettingsCallback`
 cb)=0
- virtual `telux::common::Status` `updateSrcL2Info` (`UpdateSrcL2InfoCallback` cb)=0
- virtual `telux::common::Status` `updateTrustedUEList` (const `TrustedUEInfoList` &infoList,
 `UpdateTrustedUEListCallback` cb)=0
- virtual `~ICv2xRadio ()`
- virtual std::string `getIfaceNameFromIpType` (`TrafficIpType` ipType)=0
- virtual `telux::common::Status` `createCv2xTcpSocket` (const `EventFlowInfo` &eventInfo, const
 `SocketInfo` &sockInfo, `CreateTcpSocketCallback` cb)=0
- virtual `telux::common::Status` `closeCv2xTcpSocket` (std::shared_ptr< `ICv2xTxRxSocket` > sock,

[CloseTcpSocketCallback](#) cb)=0

- virtual [telux::common::Status registerTxStatusReportListener](#) (uint16_t port, std::weak_ptr<[ICv2xTxStatusReportListener](#) > listener, [telux::common::ResponseCallback](#) cb)=0
- virtual [telux::common::Status deregisterTxStatusReportListener](#) (uint16_t port, [telux::common::ResponseCallback](#) cb)=0

5.13.1.4.1 Constructors and Destructors

5.13.1.4.1.1 **virtual** [telux::cv2x::ICv2xRadio::~~ICv2xRadio](#) () [**virtual**]

Destructor for [ICv2xRadio](#)

5.13.1.4.2 Member Function Documentation

5.13.1.4.2.1 **virtual** [Cv2xRadioCapabilities](#) [telux::cv2x::ICv2xRadio::getCapabilities](#) () const [**pure virtual**]

Get the capabilities of this Cv2xRadio.

Returns

[Cv2xRadioCapabilities](#) - Contains capabilities of this Cv2xRadio.

Deprecated Use [requestCapabilities\(\)](#) API

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

Returns true if the radio interface was successfully initialized.

Returns

True if ready. False otherwise.

5.13.1.4.2.3 **virtual bool** [telux::cv2x::ICv2xRadio::isInitialized](#) () const [**pure virtual**]

Returns true if the radio interface has completed initialization.

Returns

True if initialized. False otherwise.

5.13.1.4.2.4 `virtual std::future<telux::common::Status> telux::cv2x::ICv2xRadio::onReady ( ) [pure virtual]`

Returns a future that indicated if the radio interface is ready or if radio failed to initialize.

Returns

SUCCESS if Cv2xRadio initialization was successful. Otherwise it returns an Error Code.

5.13.1.4.2.5 `virtual telux::common::Status telux::cv2x::ICv2xRadio::registerListener ( std::weak_ptr<ICv2xRadioListener > listener ) [pure virtual]`

Registers a listener for this Cv2xRadio.

Parameters

in	<i>listener</i>	- Listener that implements Cv2xRadioListener interface.

5.13.1.4.2.6 `virtual telux::common::Status telux::cv2x::ICv2xRadio::deregisterListener ( std::weak_ptr<ICv2xRadioListener > listener ) [pure virtual]`

Deregisters a listener from this Cv2xRadio.

Parameters

in	<i>listener</i>	- Previously registered Cv2xRadioListener that is to be deregistered.

5.13.1.4.2.7 `virtual telux::common::Status telux::cv2x::ICv2xRadio::createRxSubscription ( 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()`.

5.13.1.4.2.8 `virtual telux::common::Status telux::cv2x::ICv2xRadio::createTxSpsFlow ( TrafficIpType ipType, uint32_t serviceId, const SpsFlowInfo & spsInfo, uint16_t spsSrcPort, bool eventSrcPortValid, uint16_t eventSrcPort, CreateTxSpsFlowCallback cb ) [pure virtual]`

Creates a Tx SPS flow with the specified IP type, serviceId, and other parameters specified in reservation. Additionally, an option event flow will be created with the same IP type and serviceId. A Tx socket will be created and initialized for the SPS flow. A Tx socket will be created and initialized for the event flow if the optional event flow is specified.

Parameters

in	<i>ipType</i>	- IP traffic type (IP or NON-IP)
in	<i>serviceId</i>	- ID used for transmissions that will be mapped to an L2 destination address. Variable length 4-byte PSID or ITS_AID, or another service ID.
in	<i>spsInfo</i>	- SPS reservation parameters.
in	<i>spsPort</i>	- Requested source port number for the bandwidth reserved SPS transmissions.
in	<i>eventSrcPortValid</i>	- True if an optional event flow is desired. If this field is left false, the event flow will not be created.
in	<i>eventSrcPort</i>	- Requested source port number for the optional event flow.
in	<i>cb</i>	- Callback function that is invoked when socket creation is complete. This must not be null.

This caller is expected to identify two unused local port numbers

to use for binding: one for the event-driven flow and one for the SPS flow.

Returns

SUCCESS upon success. Error status otherwise.

5.13.1.4.2.9 `virtual telux::common::Status telux::cv2x::ICv2xRadio::createTxEventFlow ( TrafficIpType ipType, uint32_t serviceId, uint16_t eventSrcPort, CreateTxEventFlowCallback cb ) [pure virtual]`

Creates an event flow. An associated Tx socket will be created and initialized.

Parameters

in	<i>ipType</i>	- IP traffic type (IP or NON-IP)
in	<i>serviceId</i>	- ID used for transmissions that will be mapped to an L2 destination address. Variable length 4-byte PSID or ITS_AID, or another service ID.

in	<i>eventSrcPort</i>	- Local port number to which the socket is bound. Used for transmissions of this ID.
in	<i>cb</i>	- Callback function that is invoked when socket creation is complete. This must not be null.

Detailed description This function is used only for TX when no periodicity is

available for the application type. If your transmit data periodicity is known, use [createTxSpsFlow\(\)](#) instead.

These even-driven sockets pay attention to the QoS parameters in

the IP socket.

Returns

SUCCESS upon success. Error status otherwise.

5.13.1.4.2.10 `virtual telux::common::Status telux::cv2x::lCv2xRadio::createTxEventFlow ( TrafficIpType ipType, uint32_t serviceId, const EventFlowInfo & flowInfo, uint16_t eventSrcPort, CreateTxEventFlowCallback cb ) [pure virtual]`

Creates an event flow. An associated Tx socket will be created and initialized.

Parameters

in	<i>ipType</i>	- IP traffic type (IP or NON-IP)
in	<i>serviceId</i>	- ID used for transmissions that will be mapped to an L2 destination address. Variable length 4-byte PSID or ITS_AID, or another service ID.
in	<i>flowInfo</i>	- Flow configuration parameters
in	<i>eventSrcPort</i>	- Local port number to which the socket is bound. Used for transmissions of this ID.
in	<i>cb</i>	- Callback function that is invoked when socket creation is complete. This must not be null.

Detailed description This function is used only for TX when no periodicity is

available for the application type. If your transmit data periodicity is known, use [createTxSpsFlow\(\)](#) instead.

These even-driven sockets pay attention to the QoS parameters in

the IP socket.

Returns

SUCCESS upon success. Error status otherwise.

5.13.1.4.2.11 `virtual telux::common::Status telux::cv2x::ICv2xRadio::closeRxSubscription ( std::shared_ptr< ICv2xRxSubscription > rxSub, CloseRxSubscriptionCallback cb ) [pure virtual]`

Closes the RxSubscription and frees resources (such as the Rx socket) associated with it.

Parameters

in	<i>rxSub</i>	- RxSubscription to close
in	<i>cb</i>	- Callback that is invoked when socket close is complete. This may be null.

Returns

SUCCESS if no error occurred.

5.13.1.4.2.12 `virtual telux::common::Status telux::cv2x::ICv2xRadio::closeTxFlow ( std::shared_ptr< ICv2xTxFlow > txFlow, CloseTxFlowCallback cb ) [pure virtual]`

Closes the TxFlow and frees resources associated with it (such as reserved SPS bandwidth contracts and sockets). This function works on both SPS and event flows.

Parameters

in	<i>txFlow</i>	- Tx (SPS or event) flow to close.
in	<i>cb</i>	- Callback that is invoked when Tx flow close is complete. This may be null.

Returns

SUCCESS if no error occurred.

5.13.1.4.2.13 `virtual telux::common::Status telux::cv2x::ICv2xRadio::changeSpsFlowInfo ( std::shared_ptr< ICv2xTxFlow > txFlow, const SpsFlowInfo & spsInfo, ChangeSpsFlowInfoCallback cb ) [pure virtual]`

Request to change TX SPS Flow reservation parameters.

Parameters

in	<i>txFlow</i>	- Tx SPS flow
in	<i>spsInfo</i>	- Desired SPS reservation parameters
in	<i>cb</i>	- Callback that is invoked upon reservation change. This may be null.

Detailed description

This function does not update reservation priority

Returns

SUCCESS if no error occurred.

5.13.1.4.2.14 `virtual telux::common::Status telux::cv2x::ICv2xRadio::requestSpsFlowInfo ( std::shared_ptr< ICv2xTxFlow > txFlow, RequestSpsFlowInfoCallback cb ) [pure virtual]`

Request SPS flow info.

Parameters

in	<i>sock</i>	- Tx SPS flow
in	<i>cb</i>	- Callback that will be invoked and returns the SPS info. Must not be null.

Returns

SUCCESS if no error occurred.

5.13.1.4.2.15 `virtual telux::common::Status telux::cv2x::ICv2xRadio::changeEventFlowInfo ( std::shared_ptr< ICv2xTxFlow > txFlow, const EventFlowInfo & flowInfo, ChangeEventFlowInfoCallback cb ) [pure virtual]`

Request to change TX Event Flow reservation parameters.

Parameters

in	<i>txFlow</i>	- Tx Event flow
in	<i>flowInfo</i>	- Desired Event flow parameters
in	<i>cb</i>	- Callback that is invoked upon parameter change. This may be null.

Returns

SUCCESS if no error occurred.

5.13.1.4.2.16 `virtual telux::common::Status telux::cv2x::ICv2xRadio::requestCapabilities ( RequestCapabilitiesCallback cb ) [pure virtual]`

Request modem Cv2x capability information.

Parameters

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

Returns

SUCCESS if no error occurred.

5.13.1.4.2.17 **virtual telux::common::Status telux::cv2x::lCv2xRadio::requestDataSessionSettings (RequestDataSessionSettingsCallback *cb*) [pure virtual]**

Request data session settings currently in use.

Parameters

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

Returns

SUCCESS if no error occurred.

5.13.1.4.2.18 **virtual telux::common::Status telux::cv2x::lCv2xRadio::updateSrcL2Info (UpdateSrcL2InfoCallback *cb*) [pure virtual]**

Requests modem to change L2 info.

Parameters

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

Returns

SUCCESS if no error occurred.

5.13.1.4.2.19 **virtual telux::common::Status telux::cv2x::lCv2xRadio::updateTrustedUEList (const TrustedUEInfoList & *infoList*, UpdateTrustedUEListCallback *cb*) [pure virtual]**

Send request to modem to update the list of malicious UE source IDs and trusted UE source IDs with corresponding confidence information.

Parameters

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

Returns

SUCCESS if no error occurred.

5.13.1.4.2.20 **virtual std::string telux::cv2x::lCv2xRadio::getInterfaceNameFromIpType (TrafficIpType *ipType*) [pure virtual]**

Get interface name based on ipType.

<i>ipType</i>	- IP traffic type (IP or NON-IP)

Returns

Interface name as a string

Note

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

5.13.1.4.2.21 virtual telux::common::Status telux::cv2x::lCv2xRadio::createCv2xTcpSocket (const EventFlowInfo & *eventInfo*, const SocketInfo & *sockInfo*, CreateTcpSocketCallback *cb*) [pure virtual]

Creates a CV2X TCP socket with specified event flow information and TCP socket information. The TCP socket will be created and bound to the IPv6 address of local IP interface with specified source port. Additionally, this API also registers a Tx event flow and subscribes Rx with specified service ID. If the created socket is expected to work as TCP client mode, the caller must connect the created socket to a destination using connect() and then use the socket for send() and recv() on successful connection. If the created socket is expected to work as TCP server mode, the caller must mark this socket as a listening socket using listen() and accept connections received from this listening socket using accept(), and then use the accepted sockets returned from accept() for send() or recv().

Parameters

in	<i>eventInfo</i>	- Information for the Event flow.
in	<i>sockInfo</i>	- Information for the TCP socket.
in	<i>cb</i>	- Callback function that is invoked when socket creation is complete. This must not be null.

The caller is expected to identify an unused local port number as the source

port number in structure [SocketInfo](#) to use for binding.

The caller must release the created socket and associated resources with

[closeCv2xTcpSocket](#). Additionally, if the created socket is marked as a listening socket, the caller must close all the accepted sockets returned by accept() using close() first, and then release the listening socket and associated resources by calling [closeCv2xTcpSocket](#).

Returns

SUCCESS upon success. Error status otherwise.

5.13.1.4.2.22 `virtual telux::common::Status telux::cv2x::ICv2xRadio::closeCv2xTcpSocket ( std::shared_ptr< ICv2xTxRxSocket > sock, CloseTcpSocketCallback cb ) [pure virtual]`

Closes the CV2X TCP socket and frees resources associated with it (such as registered event Tx flow and subscribed Rx service ID and created TCP socket).

Parameters

in	<i>sock</i>	- CV2X TCP socket to close.
in	<i>cb</i>	- Callback that is invoked when CV2X TCP socket close is complete. This may be null.

Returns

SUCCESS if no error occurred.

5.13.1.4.2.23 `virtual telux::common::Status telux::cv2x::ICv2xRadio::registerTxStatusReport-Listener ( uint16_t port, std::weak_ptr< ICv2xTxStatusReportListener > listener, telux::common::ResponseCallback cb ) [pure virtual]`

Registers a listener for Tx status report.

Parameters

in	<i>port</i>	- Set this value to the port number of registered Tx Flow if user wants to receive Tx status report associated with its own Tx flow. If user wants to receive Tx status report associated with all Tx flows in system, set this value to 0.
in	<i>listener</i>	- Listener that implements ICv2xTxStatusReportListener interface.
in	<i>cb</i>	- Callback that is invoked when the registration of CV2X Tx status report is complete.

5.13.1.4.2.24 `virtual telux::common::Status telux::cv2x::ICv2xRadio::deregisterTxStatusReportListener ( uint16_t port, telux::common::ResponseCallback cb ) [pure virtual]`

Deregisters a listener for Tx status report.

Parameters

in	<i>port</i>	- Port number of previously registered ICv2xTxStatusReport-Listener that is to be deregistered. If the listener is registered with port number 0, set this value to 0 to deregister the listener.
in	<i>cb</i>	- Callback that is invoked when the deregistration of CV2X Tx status report is complete.

5.13.1.5 class telux::cv2x::ICv2xRadioListener

Listeners for Cv2xRadio must implement this interface.

Public member functions

- virtual void [onStatusChanged](#) (Cv2xStatus status)
- virtual void [onStatusChanged](#) (Cv2xStatusEx status)
- virtual void [onL2AddrChanged](#) (uint32_t newL2Address)
- virtual void [onSpsOffsetChanged](#) (int spsId, [MacDetails](#) details)
- virtual void [onSpsSchedulingChanged](#) (const [SpsSchedulingInfo](#) &schedulingInfo)
- virtual void [onCapabilitiesChanged](#) (const [Cv2xRadioCapabilities](#) &capabilities)
- virtual [~ICv2xRadioListener](#) ()

5.13.1.5.1 Constructors and Destructors

5.13.1.5.1.1 virtual telux::cv2x::ICv2xRadioListener::~ICv2xRadioListener () [virtual]

Destructor for [ICv2xRadioListener](#)

5.13.1.5.2 Member Function Documentation

5.13.1.5.2.1 virtual void telux::cv2x::ICv2xRadioListener::onStatusChanged (Cv2xStatus *status*)
[virtual]

Called when the status of the CV2X radio has changed.

Parameters

in	<i>status</i>	- CV2X radio status.

Deprecated use onStatusChanged in Cv2xListener

5.13.1.5.2.2 virtual void telux::cv2x::ICv2xRadioListener::onStatusChanged (Cv2xStatusEx *status*)
[virtual]

Called when the status of the CV2X radio has changed.

Parameters

in	<i>status</i>	- CV2X radio status.

Deprecated use onStatusChanged in Cv2xListener

5.13.1.5.2.3 virtual void telux::cv2x::ICv2xRadioListener::onL2AddrChanged (uint32_t *newL2Address*) [virtual]

Called when the L2 Address has changed.

Parameters

in	<i>newL2Address</i>	- The new L2 address.

5.13.1.5.2.4 virtual void telux::cv2x::ICv2xRadioListener::onSpsOffsetChanged (int *spsId*, MacDetails *details*) [virtual]

Called when SPS offset has changed.

Parameters

in	<i>spsId</i>	- SPS Id of the SPS flow
in	<i>details</i>	- new SPS MAC PHY details.

Deprecated use onSpsSchedulingChanged

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

Called when SPS scheduling has changed.

Parameters

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

5.13.1.5.2.6 virtual void telux::cv2x::ICv2xRadioListener::onCapabilitiesChanged (const Cv2xRadioCapabilities & *capabilities*) [virtual]

Called when Cv2x radio capabilities have changed.

Parameters

in	<i>capabilities</i>	- Capabilities of the CV2X radio .

5.13.1.6 class telux::cv2x::ICv2xRadioManager

Cv2xRadioManager manages instances of Cv2xRadio.

Public member functions

- virtual bool [isReady](#) ()=0
- virtual std::future< bool > [onReady](#) ()=0

- virtual std::shared_ptr
< ICv2xRadio > getCv2xRadio (TrafficCategory category)=0
- virtual telux::common::Status startCv2x (StartCv2xCallback cb)=0
- virtual telux::common::Status stopCv2x (StopCv2xCallback cb)=0
- virtual telux::common::Status requestCv2xStatus (RequestCv2xStatusCallback cb)=0
- virtual telux::common::Status requestCv2xStatus (RequestCv2xStatusCallbackEx cb)=0
- virtual telux::common::Status registerListener (std::weak_ptr< ICv2xListener > listener)=0
- virtual telux::common::Status deregisterListener (std::weak_ptr< ICv2xListener > listener)=0
- virtual telux::common::Status updateConfiguration (const std::string &configFilePath, UpdateConfigurationCallback cb)=0
- virtual ~ICv2xRadioManager ()

5.13.1.6.1 Constructors and Destructors

5.13.1.6.1.1 virtual telux::cv2x::ICv2xRadioManager::~~ICv2xRadioManager () [virtual]

5.13.1.6.2 Member Function Documentation

5.13.1.6.2.1 virtual bool telux::cv2x::ICv2xRadioManager::isReady () [pure virtual]

Checks if the Cv2x Radio Manager is ready.

Returns

True if Cv2x Radio Manager is ready for service, otherwise returns false.

5.13.1.6.2.2 virtual std::future<bool> telux::cv2x::ICv2xRadioManager::onReady () [pure virtual]

Wait for Cv2x Radio Manager to be ready.

Returns

A future that caller can wait on to be notified when Cv2x Radio Manager is ready.

5.13.1.6.2.3 virtual std::shared_ptr<ICv2xRadio> telux::cv2x::ICv2xRadioManager::getCv2xRadio (TrafficCategory category) [pure virtual]

Get Cv2xRadio instance

Parameters

in	category	- Specifies the category of the client application. This field is currently unused.

Returns

Reference to Cv2xRadio interface that corresponds to the Cv2x Traffic Category specified.

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

Put modem into CV2X mode.

Parameters

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

Returns

SUCCESS on success. Error status otherwise.

5.13.1.6.2.5 `virtual telux::common::Status telux::cv2x::ICv2xRadioManager::stopCv2x ( StopCv2x-Callback cb ) [pure virtual]`

Take modem out of CV2X mode

Parameters

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

Returns

SUCCESS on success. Error status otherwise.

5.13.1.6.2.6 `virtual telux::common::Status telux::cv2x::ICv2xRadioManager::requestCv2xStatus ( RequestCv2xStatusCallback cb ) [pure virtual]`

request CV2X status from modem

Parameters

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

Returns

SUCCESS on success. Error status otherwise.

Deprecated use requestCv2xStatus(RequestCv2xCalbackEx)

5.13.1.6.2.7 `virtual telux::common::Status telux::cv2x::ICv2xRadioManager::requestCv2xStatus ( RequestCv2xStatusCallbackEx cb ) [pure virtual]`

request CV2X status from modem

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

Returns

SUCCESS on success. Error status otherwise.

5.13.1.6.2.8 `virtual telux::common::Status telux::cv2x::ICv2xRadioManager::registerListener ( std::weak_ptr< ICv2xListener > listener ) [pure virtual]`

Registers a listener for this manager.

Parameters

<i>in</i>	<i>listener</i>	- Listener that implements Cv2xListener interface.

5.13.1.6.2.9 `virtual telux::common::Status telux::cv2x::ICv2xRadioManager::deregisterListener ( std::weak_ptr< ICv2xListener > listener ) [pure virtual]`

Deregisters a Cv2xListener for this manager.

Parameters

<i>in</i>	<i>listener</i>	- Previously registered CvListener that is to be deregistered.

5.13.1.6.2.10 `virtual telux::common::Status telux::cv2x::ICv2xRadioManager::updateConfiguration ( const std::string & configFilePath, UpdateConfigurationCallback cb ) [pure virtual]`

Updates CV2X configuration. Requires CV2X TX/RX radio status be Inactive. If CV2X radio status is Active or Suspended, call [stopCv2x](#) before updateConfiguration.

Parameters

<i>in</i>	<i>configFilePath</i>	- Path to config file.
<i>in</i>	<i>cb</i>	- Callback that is invoked when the send is complete. This may be null.

Deprecated Use [ICv2xConfig](#) instead

5.13.1.7 struct telux::cv2x::SocketInfo

Encapsulates parameters of a CV2X socket.

Used in createCv2xTcpSocket.

Data fields

Type	Field	Description
uint32_t	serviceId	V2X service ID bound to the socket.
uint16_t	localPort	Local port number of the socket used for binding.

5.13.1.8 struct telux::cv2x::Cv2xStatus

Encapsulates status of CV2X radio.

Used in Cv2xRadioManager:requestV2xStatus and Cv2xRadioListener.

Data fields

Type	Field	Description
Cv2xStatus-Type	rxStatus	RX status
Cv2xStatus-Type	txStatus	TX status
Cv2xCause-Type	rxCause	RX cause of failure
Cv2xCause-Type	txCause	TX cause of failure
uint8_t	cbrValue	Channel Busy Ratio
bool	cbrValueValid	CBR value is valid

5.13.1.9 struct telux::cv2x::Cv2xPoolStatus

Encapsulates status for single pool.

Used in [Cv2xStatusEx](#).

Data fields

Type	Field	Description
uint8_t	poolId	pool ID
Cv2xStatus	status	status

5.13.1.10 struct telux::cv2x::Cv2xStatusEx

Encapsulates status of CV2X radio and per pool status.

Used in Cv2xRadioManager:requestV2xStatus and Cv2xRadioListener.

Data fields

Type	Field	Description
Cv2xStatus	status	Overall Cv2x status
vector< Cv2x-PoolStatus >	poolStatus	Multi pool status vector
bool	time-Uncertainty-Valid	Time uncertainty value is valid
float	time-Uncertainty	Time uncertainty value in milleseconds

5.13.1.11 struct telux::cv2x::TxPoolIdInfo

Contains minimum and maximum frequency for a given TX pool ID. Multiple TX Pools allow the same radio and overall frequency range to be shared for multiple types of traffic like V2V and V2X. Each pool ID and frequency range corresponds to a certain type of traffic.

Used in [Cv2xRadioCapabilities](#)

Data fields

Type	Field	Description
uint8_t	poolId	TX pool ID.
uint16_t	minFreq	Minimum frequency in MHz.
uint16_t	maxFreq	Maximum frequency in MHz.

5.13.1.12 struct telux::cv2x::EventFlowInfo

Contains event flow configuration parameters.

Used in createTxEventFlow

Data fields

Type	Field	Description
bool	autoRetrans-EnabledValid	Set to true if autoRetransEnabled field is specified. If false, the system will use the default setting.
bool	autoRetrans-Enabled	Used to enable automatic-retransmissions.
bool	peakTxPower-Valid	Set to true if peakTxPower is used. If false, the system will use the default setting.
int32_t	peakTxPower	Max Tx power setting in dBm.
bool	mcsIndexValid	Set to true if mcsIndex is used. If false, the system will use its default setting.
uint8_t	mcsIndex	Modulation and Coding Scheme Index to use.
bool	txPoolIdValid	Set to true if txPoolId is used. If false, the system will use its default setting.
uint8_t	txPoolId	Transmission Pool ID.

5.13.1.13 struct telux::cv2x::SpsFlowInfo

Used to request the QoS bandwidth contract, implemented in PC5 3GPP V2X radio as a *Semi Persistent Flow* (SPS).

The underlying radio providing the interface might support periodicities of various granularity in 100ms integer multiples (e.g. 200ms, 300ms).

Used in txSpsCreateAndBindSock and changeSpsFlowInfo

Data fields

Type	Field	Description
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.

5.13.1.14 struct telux::cv2x::Cv2xRadioCapabilities

Contains capabilities of the Cv2xRadio.

Used in requestCapabilities and onCapabilitiesChanged

Data fields

Type	Field	Description
uint32_t	linkIpMtuBytes	Maximum data payload length (in bytes) of a packet supported by the IP Radio interface.
uint32_t	linkNonIpMtu-Bytes	Maximum data payload length (in bytes) of a packet supported by the non-IP Radio interface.
Radio-Concurrency-Mode	maxSupported-Concurrency	Indicates whether this interface supports concurrent WWAN with V2X (PC5).
uint16_t	nonIpTx-PayloadOffset-Bytes	Byte offset in a non-IP Tx packet before the actual payload begins.

Type	Field	Description
uint16_t	nonIpRx-PayloadOffset-Bytes	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	maxNumAuto-Retransmissions	Least frequent bandwidth periodicity that is supported. Above this value, use event-driven periodic messages of a period larger than this value.
uint8_t	layer2Mac-AddressSize	Size of the L2 MAC address. Different Radio Access Technologies have different-sized L2 MAC addresses: 802.11 has 6 bytes, whereas 3GPP PC5 has only 3 bytes. Because a randomized MAC address comes from an HSM with good pseudo random entropy, higher layers must know how many bytes of the MAC address to generate.
bitset< 8 >	priorities-Supported	Bit set of different priority levels supported by this Cv2xRadio. Refer to Priority
uint16_t	maxNumSps-Flows	Maximum number of supported SPS reservations.
uint16_t	maxNumNon-SpsFlows	Maximum number of supported event flows (non-SPS ports).
int32_t	maxTxPower	Maximum supported transmission power.
int32_t	minTxPower	Minimum supported transmission power.
vector< Tx-PoolIdInfo >	txPoolIds-Supported	Vector of supported transmission pool IDs.

5.13.1.15 struct telux::cv2x::MacDetails

Contains MAC information that is reported from the actual MAC SPS in the radio. The offsets can periodically change on any given transmission report.

Data fields

Type	Field	Description
uint32_t	periodicityIn-UseNs	Actual transmission interval period (in nanoseconds) scheduled relative to 1PP 0:00.00 time
uint16_t	currently-Reserved-PeriodicBytes	Actual number of bytes currently reserved at the MAC layer. This number can be slightly larger than original request.
uint32_t	txReservation-OffsetNs	Actual offset, from a 1PPS pulse and TX flow periodicity, that the MAC selected and is using for the transmit reservation. If the data goes to the radio with enough time, it can be transmitted on the medium in the next immediately scheduled slot.

5.13.1.16 struct telux::cv2x::SpsSchedulingInfo

Contains SPS packet scheduling information that is reported from the radio.

Used in onSpsSchedulingChanged

Data fields

Type	Field	Description
uint8_t	spsId	SPS ID
uint64_t	utcTime	Absolute UTC start time of next selected grant in nanoseconds.
uint32_t	periodicity	Periodicity of the grant in milliseconds.

5.13.1.17 struct telux::cv2x::TrustedUEInfo

Contains time confidence, position confidence, and propagation delay for a trusted UE.

Used in [TrustedUEInfo](#)

Data fields

Type	Field	Description
uint32_t	sourceL2Id	Trusted Source L2 ID
float	time- Uncertainty	Time uncertainty value in milliseconds.
uint16_t	time- Confidence- Level	Deprecated Use timeUncertainty Time confidence level. Range from 0 to 127 with 0 being invalid/unavailable and 127 being the most confident.
uint16_t	position- Confidence- Level	Position confidence level. Range from 0 to 127 with 0 being invalid/unavailable and 127 being the most confident.
uint32_t	propagation- Delay	Propagation delay in microseconds.

5.13.1.18 struct telux::cv2x::TrustedUEInfoList

Contains list of malicious UE source L2 IDs. Contains list of trusted UE source L2 IDs and associated confidence values.

Used in updateTrustedUEList

Data fields

Type	Field	Description
bool	maliciousIds- Valid	Malicious remote UE sources are valid.
vector< uint32- _t >	maliciousIds	Malicious remote UE source L2 IDs.

Type	Field	Description
bool	trustedUEs-Valid	Trusted remote UE sources are valid.
vector< TrustedUEInfo >	trustedUEs	Trusted remote UE sources.

5.13.1.19 struct telux::cv2x::IPv6Address

Contains IPv6 address.

Used in [DataSessionSettings](#)

Data fields

Type	Field	Description
uint8_t	addr[16]	

5.13.1.20 struct telux::cv2x::DataSessionSettings

Contains packet data session settings.

Used in requestDataSessionSettings

Data fields

Type	Field	Description
bool	mtuValid	Set to true if mtu is valid.
uint32_t	mtu	MTU size.
bool	ipv6AddrValid	Set to true if ipv6 address is valid.
IPv6Address	ipv6Addr	IPv6 address.

5.13.1.21 struct telux::cv2x::ConfigEventInfo

Information about any update to a CV2X config file.

Used in onConfigFileChanged

Data fields

Type	Field	Description
ConfigSource-Type	source	The type of the V2X config file.
ConfigEvent	event	V2X config event.

5.13.1.22 struct telux::cv2x::RFTxInfo

Tx status per Tx chain and Tx power per Tx antenna for a specific transport block.

Used in [TxStatusReport](#)

Data fields

Type	Field	Description
RFTxStatus	status	Fault detection status for a specific Tx chain.
int32_t	power	The target Tx power after MPR/AMPR reduction for a specific Tx antenna in dBm*10 format. Invalid value is -700, it means the corresponding antenna is not being used for the transmission of this transport block.

5.13.1.23 struct telux::cv2x::TxStatusReport

Information on Tx status of a V2X transport block that is reported from low layer.

1. A V2X packet might trigger multiple reports because of the segmentaion and re-Tx in low layer.
2. If a transport block is dropped in low layer, no report will be triggered for that transport block.
3. The power in the array of rfInfo is the target Tx power value in dBm*10 after MPR/AMPR reduction for a specific Tx antenna. The status in the array of rfInfo is the fault detection status for a specific Tx chain.
 - In CDD mode, two antennas are being used for a specific transport block, both rfInfo[0].power and rfInfo[1].power are valid (not -700), rfInfo[i].status is reflecting the status of Tx chain/Tx antenna i.
 - In TXD mode, data transmission switches between two antennas/chains and only one antenna/chain is being used for a specific transport block, the Tx antenna being used has valid power (not -700) in the array of rfInfo, rfInfo[i].status is reflecting the status of Tx chain i or the status of the Tx antenna i whose power is valid (not -700) in the array of rfInfo. Used in onTxStatusReport

Data fields

Type	Field	Description
RFTxInfo	rfInfo[MAX_-ANTENNAS_-SUPPORTED]	Tx status per Tx chain and Tx power per Tx antenna.
uint8_t	numRb	Number of resource blocks used for the transport block.
uint8_t	startRb	Start resource block index used for the transport block.
uint8_t	mcs	Modulation and coding scheme used for the transport block that is defined in 3GPP TS 36.213.
uint8_t	segNum	Total number of segments of a V2X packet.
SegmentType	segType	Segment type of the transport block.
TxType	txType	Indication of new Tx or re-Tx of the transport block.
uint16_t	otaTiming	OTA timing in format of system frame number*10 + subframe number.

Type	Field	Description
uint16_t	port	Port number that can be used to link the report to a specific Tx flow which has the same source port number.

5.13.1.24 class telux::cv2x::ICv2xRxSubscription

This class encapsulates a Cv2xRadio Rx Subscription. It contains the Rx socket associated with the subscription from which client applications can read data. This class is referenced in Cv2xRadio::createRxSubscription and Cv2xRadio::closeRxSubscription.

Public member functions

- virtual uint32_t [getSubscriptionId](#) () const =0
- virtual [TrafficIpType](#) [getIpType](#) () const =0
- virtual int [getSock](#) () const =0
- virtual struct sockaddr_in6 [getSockAddr](#) () const =0
- virtual uint16_t [getPortNum](#) () const =0
- virtual std::shared_ptr
< std::vector< uint32_t > > [getServiceIDList](#) () const =0
- virtual void [setServiceIDList](#) (const std::shared_ptr< std::vector< uint32_t > > idList)=0
- virtual [~ICv2xRxSubscription](#) ()

5.13.1.24.1 Constructors and Destructors

5.13.1.24.1.1 virtual telux::cv2x::ICv2xRxSubscription::~ICv2xRxSubscription () [virtual]

5.13.1.24.2 Member Function Documentation

5.13.1.24.2.1 virtual uint32_t telux::cv2x::ICv2xRxSubscription::getSubscriptionId () const [pure virtual]

Accessor for Rx subscription ID

Returns

subscription ID

5.13.1.24.2.2 virtual TrafficIpType telux::cv2x::ICv2xRxSubscription::getIpType () const [pure virtual]

Accessor for IP traffic type

Returns

The Rx subscriptions's IP traffic type (IP or NON-IP)

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

Accessor for the socket file descriptor

Returns

The Rx subscriptions's socket fd.

5.13.1.24.2.4 `virtual struct sockaddr_in6 telux::cv2x::ICv2xRxSubscription::getSockAddr ( ) const [read], [pure virtual]`

Accessor for the socket address description

Returns

The Rx subscriptions's socket address

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

Accessor for the subscriptions's port number

Returns

The Rx subscriptions's port num

5.13.1.24.2.6 `virtual std::shared_ptr<std::vector<uint32_t> > telux::cv2x::ICv2xRxSubscription::getServiceIDList ( ) const [pure virtual]`

Get subscriptions's service ID list

Returns

The Rx subscriptions's service ID list

5.13.1.24.2.7 `virtual void telux::cv2x::ICv2xRxSubscription::setServiceIDList ( const std::shared_ptr<std::vector< uint32_t >> idList ) [pure virtual]`

Set subscriptions's service ID list

Parameters

in	<i>idList</i>	- the subscriptions's service ID list

5.13.1.25 class telux::cv2x::ICv2xTxFlow

This class encapsulates a Cv2xRadio Tx flows. It contains the Tx socket associated with the flow through which client applications can send data. This class is referenced in Cv2xRadio::createTxSpsFlow, Cv2xRadio::createTxEventFlow, and Cv2xRadio::closeTxFlow

Public member functions

- virtual uint32_t [getFlowId](#) () const =0
- virtual [TrafficIpType](#) [getIpType](#) () const =0
- virtual uint32_t [getServiceId](#) () const =0
- virtual int [getSock](#) () const =0
- virtual struct sockaddr_in6 [getSockAddr](#) () const =0
- virtual uint16_t [getPortNum](#) () const =0
- virtual [~ICv2xTxFlow](#) ()

5.13.1.25.1 Constructors and Destructors

5.13.1.25.1.1 virtual telux::cv2x::ICv2xTxFlow::~~ICv2xTxFlow () [virtual]

5.13.1.25.2 Member Function Documentation

5.13.1.25.2.1 virtual uint32_t telux::cv2x::ICv2xTxFlow::getFlowId () const [pure virtual]

Accessor for flow ID. The flow ID should be unique within a process but will not be unique between processes.

Returns

flow ID

5.13.1.25.2.2 virtual TrafficIpType telux::cv2x::ICv2xTxFlow::getIpType () const [pure virtual]

Accessor for IP traffic type

Returns

The flow's IP traffic type (IP or NON-IP)

5.13.1.25.2.3 virtual uint32_t telux::cv2x::ICv2xTxFlow::getServiceId () const [pure virtual]

Accessor for service ID

Returns

The flow's Service ID.

5.13.1.25.2.4 virtual int telux::cv2x::ICv2xTxFlow::getSock () const [pure virtual]

Accessor for the socket file descriptor

Returns

The flow's socket fd.

5.13.1.25.2.5 virtual struct sockaddr_in6 telux::cv2x::ICv2xTxFlow::getSockAddr () const [read], [pure virtual]

Accessor for the socket address description

Returns

The flow's socket address

5.13.1.25.2.6 virtual uint16_t telux::cv2x::ICv2xTxFlow::getPortNum () const [pure virtual]

Accessor for the flow's source port number

Returns

The flow's source port num

5.13.1.26 class telux::cv2x::ICv2xTxRxSocket

This is class encapsulates a Cv2xRadio socket for both Tx and Rx. It contains the socket through which client applications can send and receive data. This class is referenced in Cv2xRadio::createCv2xTcpSocket and Cv2xRadio::closeCv2xTcpSocket.

Public member functions

- virtual uint32_t [getId](#) () const =0
- virtual uint32_t [getServiceId](#) () const =0
- virtual int [getSocket](#) () const =0
- virtual struct sockaddr_in6 [getSocketAddr](#) () const =0
- virtual uint16_t [getPortNum](#) () const =0
- virtual [~ICv2xTxRxSocket](#) ()

5.13.1.26.1 Constructors and Destructors**5.13.1.26.1.1 virtual telux::cv2x::ICv2xTxRxSocket::~~ICv2xTxRxSocket () [virtual]**

5.13.1.26.2 Member Function Documentation

5.13.1.26.2.1 `virtual uint32_t telux::cv2x::ICv2xTxRxSocket::getId ( ) const [pure virtual]`

Accessor for Cv2xRadio socket ID. The socket ID should be unique within a process but will not be unique between processes.

Returns

Cv2xRadio socket ID

5.13.1.26.2.2 `virtual uint32_t telux::cv2x::ICv2xTxRxSocket::getServiceId ( ) const [pure virtual]`

Accessor for service ID

Returns

The Service ID bound to the socket.

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

Accessor for the socket file descriptor

Returns

The socket fd.

5.13.1.26.2.4 `virtual struct sockaddr_in6 telux::cv2x::ICv2xTxRxSocket::getSocketAddr ( ) const [read], [pure virtual]`

Accessor for the socket address description

Returns

The socket address

5.13.1.26.2.5 `virtual uint16_t telux::cv2x::ICv2xTxRxSocket::getPortNum ( ) const [pure virtual]`

Accessor for the local port number bound to the socket

Returns

The local port number

5.13.2 Enumeration Type Documentation

5.13.2.1 enum telux::cv2x::TrafficCategory

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

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

Defines possible values for cause of CV2X radio failure.

Used in [Cv2xStatus](#)

Enumerator

TIMING V2X timing is not valid
CONFIG No valid V2X configuration
UE_MODE V2X is not supported in current UE mode
GEOPOLYGON V2X is not supported in current UE location
THERMAL Device's temperature is high and is in thermal mitigation mode
THERMAL_ECALL Device is in an emergency call and the device's temperature has crossed a threshold resulting in thermal mitigation
GEOPOLYGON_SWITCH V2X stack is suspended due to geopolygon switch
SENSING V2X stack is suspended due to sensing
LPM V2X is not supported under Low Power Mode
UNKNOWN Cause is unknown

5.13.2.4 enum telux::cv2x::TrafficIpType

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

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

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

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

Range of supported periodicities in milliseconds.

Used in [Cv2xRadioCapabilities](#) and [SpsFlowInfo](#)

: enum class not going to be supported in future releases. Clients should stop using this. Once a class has been marked as Deprecated, the class could be removed in future releases.

Enumerator

PERIODICITY_10MS
PERIODICITY_20MS
PERIODICITY_50MS
PERIODICITY_100MS
PERIODICITY_UNKNOWN

5.13.2.9 enum telux::cv2x::ConfigSourceType

V2X configuration source types listed in ascending order of priority. The system always uses the V2X configuration with the highest priority if multiple V2X configuration sources exist.

Used in [ConfigEventInfo](#)

Enumerator

UNKNOWN V2X config file source is unknown
PRECONFIG V2X config file source is preconfig
SIM_CARD V2X config file source is SIM card
OMA_DM V2X config file source is OMA-DM

5.13.2.10 enum telux::cv2x::ConfigEvent

Defines possible values for the events relevant to CV2X config file.

Used in [ConfigEventInfo](#)

Enumerator

CHANGED V2X config file is changed
EXPIRED V2X config file is expired

5.13.2.11 enum telux::cv2x::RFTxStatus

Fault detection for Tx chain that including PA and front end.

Used in [RFTxInfo](#)

Enumerator

INACTIVE The Tx chain is not working.
OPERATIONAL The Tx chain is operational.
FAULT Fault detected on the Tx chain.

5.13.2.12 enum telux::cv2x::SegmentType

Defines possible values for the segment type of a transport block.

Used in [TxStatusReport](#)

Enumerator

FIRST V2X packet is segmented, it's the first transport block.

LAST V2X packet is segmented, it's the last transport block.

MIDDLE V2X packet is segmented, it's a transport block between first and last.

ONLY_ONE V2X packet is not segmented, it's the only one transport block.

5.13.2.13 enum telux::cv2x::TxType

Defines new Tx or re-Tx type relevant to a transport block.

Used in [TxStatusReport](#)

Enumerator

NEW_TX New Tx of the V2X transport block.

RE_TX Re-Tx of the V2X transport block.

5.13.3 Variable Documentation

5.13.3.1 constexpr uint8_t telux::cv2x::MAX_ANTENNAS_SUPPORTED = 2u

Defines Maximum number of antennas that is supported.

Used in [TxStatusReport](#)

5.14 Audio

This section contains APIs related to Audio Stream operation.

5.14.1 Data Structure Documentation

5.14.1.1 struct telux::audio::FormatParams

Frame format common parameters

5.14.1.2 struct telux::audio::AmrwbpParams

Frame format codec specific parameters

Data Fields

- uint32_t [bitWidth](#)
- [AmrwbpFrameFormat](#) [frameFormat](#)

5.14.1.2.1 Field Documentation

5.14.1.2.1.1 uint32_t telux::audio::AmrwbpParams::bitWidth

Bitwidth of Stream, Typical Values <16/24>.

5.14.1.2.1.2 AmrwbpFrameFormat telux::audio::AmrwbpParams::frameFormat

5.14.1.3 struct telux::audio::StreamConfig

Common Stream configuration parameters

Data fields

Type	Field	Description
StreamType	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>
ChannelType-Mask	channelType-Mask	
AudioFormat	format	
vector< DeviceType >	deviceTypes	
vector< Direc- tion >	voicePaths	Represent voice path direction for in call audio. TX for Uplink and RX for Downlink.>
FormatParams *	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>
ChannelType-Mask	mask	parameter for configuration of channel type
AudioFormat	format	Represents audio format
FormatParams *	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
ChannelType	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< ChannelVolume >	volume	
Stream-Direction	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

Type	Field	Description
Stream-Direction	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
DtmfLowFreq	lowFreq	
DtmfHighFreq	highFreq	
Stream-Direction	direction	

5.14.1.10 class telux::audio::AudioFactory

[AudioFactory](#) allows creation of audio manager.

Public member functions

- std::shared_ptr< [IAudioManager](#) > [getAudioManager](#) ()

Static Public Member Functions

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

5.14.1.10.1 Member Function Documentation

5.14.1.10.1.1 static AudioFactory& telux::audio::AudioFactory::getInstance () [static]

Get Audio Factory instance.

5.14.1.10.1.2 `std::shared_ptr<IAudioManager> telux::audio::AudioFactory::getAudioManager ( )`

Get instance of audio manager.

Returns

[IAudioManager](#) pointer.

5.14.1.11 `class telux::audio::IVoiceListener`

Listener class for getting notifications related to DTMF tone detection. The client needs to implement these methods as briefly as possible and avoid blocking calls in it. The methods in this class can be invoked from multiple different threads. Client needs to make sure that the implementation is thread-safe.

Public member functions

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

5.14.1.11.1 Constructors and Destructors

5.14.1.11.1.1 `virtual telux::audio::IVoiceListener::~~IVoiceListener ( ) [virtual]`

Destructor of [IVoiceListener](#)

5.14.1.11.2 Member Function Documentation

5.14.1.11.2.1 `virtual void telux::audio::IVoiceListener::onDtmfToneDetection ( DtmfTone dtmfTone ) [virtual]`

This function is called when a DTMF tone is detected in the voice stream

Parameters

in	<i>dtmfTone</i>	DTMF tone properties

5.14.1.12 `class telux::audio::IPlayListener`

Public member functions

- virtual void [onReadyForWrite](#) ()
- virtual void [onPlayStopped](#) ()
- virtual [~IPlayListener](#) ()

5.14.1.12.1 Constructors and Destructors

5.14.1.12.1.1 `virtual telux::audio::IPlayListener::~~IPlayListener ( ) [virtual]`

Destructor of [IPlayListener](#)

5.14.1.12.2 Member Function Documentation

5.14.1.12.2.1 `virtual void telux::audio::IPlayListener::onReadyForWrite ( ) [virtual]`

This function is called when pipeline is ready to accept new buffer. It is applicable only for compressed audio format type where a client can write and queue buffers for playback.

Note

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

5.14.1.12.2.2 `virtual void telux::audio::IPlayListener::onPlayStopped ( ) [virtual]`

This function is called when stopAudio() is called with StopType::STOP_AFTER_PLAY. It indicates that all the buffers that were present in the pipeline have been played.

Note

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

5.14.1.13 class telux::audio::ITranscodeListener

Public member functions

- virtual void [onReadyForWrite](#) ()
- virtual [~ITranscodeListener](#) ()

5.14.1.13.1 Constructors and Destructors

5.14.1.13.1.1 `virtual telux::audio::ITranscodeListener::~~ITranscodeListener ( ) [virtual]`

Destructor of [ITranscodeListener](#)

5.14.1.13.2 Member Function Documentation

5.14.1.13.2.1 `virtual void telux::audio::ITranscodeListener::onReadyForWrite ( ) [virtual]`

This function is called when pipeline is ready to accept new buffer. It is applicable only for compressed audio format type where a client can write and queue buffers for transcoding.

Note

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

5.14.1.14 class telux::audio::IAudioListener**Public member functions**

- virtual [~IAudioListener](#) ()

5.14.1.14.1 Constructors and Destructors

5.14.1.14.1.1 virtual telux::audio::IAudioListener::~~IAudioListener () [virtual]

Destructor of [IAudioListener](#)

5.14.1.15 class telux::audio::IAudioBuffer

Stream Buffer manages the buffer to be used for read and write operations on Audio Streams. For write operations, applications should request a stream buffer, populate it with the data and then pass it to the write operation and set the dataSize that is to be written to the stream. Similarly for read operations, the application should request a stream buffer and use that in the read operation. At the end of the read, the stream buffer will contain the data read. Once an operation (read/write) has completed, the stream buffer could be reused for a subsequent read/write operation, provided [reset\(\)](#) API called on stream buffer between subsequent calls.

Public member functions

- virtual size_t [getMinSize](#) ()=0
- virtual size_t [getMaxSize](#) ()=0
- virtual uint8_t * [getRawBuffer](#) ()=0
- virtual uint32_t [getDataSize](#) ()=0
- virtual void [setDataSize](#) (uint32_t size)=0
- virtual [telux::common::Status reset](#) ()=0
- virtual [~IAudioBuffer](#) ()

5.14.1.15.1 Constructors and Destructors

5.14.1.15.1.1 virtual telux::audio::IAudioBuffer::~~IAudioBuffer () [virtual]

5.14.1.15.2 Member Function Documentation

5.14.1.15.2.1 virtual size_t telux::audio::IAudioBuffer::getMinSize () [pure virtual]

Returns the minimum size (in bytes) of data that caller needs to read/write before calling a read/write operation on the stream.

Returns

minimum size

5.14.1.15.2.2 virtual size_t telux::audio::IAudioBuffer::getMaxSize () [pure virtual]

Returns the maximum size (in bytes) that the buffer can hold.

Returns

maximum size

5.14.1.15.2.3 virtual uint8_t* telux::audio::IAudioBuffer::getRawBuffer () [pure virtual]

Gets the raw buffer that [IStreamBuffer](#) manages. Application should write in between(include) of [getMinSize\(\)](#) to [getMaxSize\(\)](#) number of bytes in this buffer. Application is not responsible to free the raw buffer. It will be free'd when the [IStreamBuffer](#) is destroyed.

Returns

raw buffer

5.14.1.15.2.4 virtual uint32_t telux::audio::IAudioBuffer::getDataSize () [pure virtual]

Gets the size (in bytes) of valid data present in the buffer.

Returns

size of valid data in the buffer

5.14.1.15.2.5 virtual void telux::audio::IAudioBuffer::setDataSize (uint32_t size) [pure virtual]

Sets the size (in bytes) of valid data present in the buffer.

Parameters

<i>size</i>	size of valid data in the buffer

5.14.1.15.2.6 virtual telux::common::Status telux::audio::IAudioBuffer::reset () [pure virtual]

Reset all state and data of the buffer. This is to be called when reusing the same buffer for multiple operations.

Returns

status Status of the operation

5.14.1.16 class telux::audio::IStreamBuffer

Public member functions

- virtual [~IStreamBuffer](#) ()

5.14.1.16.1 Constructors and Destructors

5.14.1.16.1.1 virtual telux::audio::IStreamBuffer::~IStreamBuffer () [virtual]

5.14.1.17 class telux::audio::IAudioManager

Audio Manager is a primary interface for audio operations. It provide APIs to manage Streams (like voice, play, record etc) and sound cards.

Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status](#) [getDevices](#) ([GetDevicesResponseCb](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [getStreamTypes](#) ([GetStreamTypesResponseCb](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [createStream](#) ([StreamConfig](#) streamConfig, [CreateStreamResponseCb](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [createTranscoder](#) ([FormatInfo](#) input, [FormatInfo](#) output, [CreateTranscoderResponseCb](#) callback)=0
- virtual [telux::common::Status](#) [deleteStream](#) (std::shared_ptr< [IAudioStream](#) > stream, [DeleteStreamResponseCb](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [registerListener](#) (std::weak_ptr< [IAudioListener](#) > listener)=0
- virtual [telux::common::Status](#) [deRegisterListener](#) (std::weak_ptr< [IAudioListener](#) > listener)=0
- virtual [~IAudioManager](#) ()

5.14.1.17.1 Constructors and Destructors

5.14.1.17.1.1 virtual telux::audio::IAudioManager::~IAudioManager () [virtual]

5.14.1.17.2 Member Function Documentation

5.14.1.17.2.1 virtual bool telux::audio::IAudioManager::isSubsystemReady () [pure virtual]

Checks the status of audio subsystems and returns the result.

Returns

If true that means AudioManager is ready for performing audio operations.

5.14.1.17.2.2 `virtual std::future<bool> telux::audio::IAudioManager::onSubsystemReady ( ) [pure virtual]`

Wait for Audio subsystem to be ready.

Returns

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

5.14.1.17.2.3 `virtual telux::common::Status telux::audio::IAudioManager::getDevices ( GetDevices-ResponseCb callback = nullptr ) [pure virtual]`

Get the list of supported audio devices, which are currently supported in the audio subsystem

Parameters

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

Returns

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

5.14.1.17.2.4 `virtual telux::common::Status telux::audio::IAudioManager::getStreamTypes ( GetStreamTypesResponseCb callback = nullptr ) [pure virtual]`

Get the list of supported audio streams types, which are currently supported in the audio subsystem

Parameters

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

Returns

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

5.14.1.17.2.5 `virtual telux::common::Status telux::audio::IAudioManager::createStream ( StreamConfig streamConfig, CreateStreamResponseCb callback = nullptr ) [pure virtual]`

Creates the stream for audio operation

Parameters

in	<i>streamConfig</i>	stream configuration.
in	<i>callback</i>	callback pointer to get the response of createStream.

Returns

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

5.14.1.17.2.6 `virtual telux::common::Status telux::audio::IAudioManager::createTranscoder ( FormatInfo input, FormatInfo output, CreateTranscoderResponseCb callback ) [pure virtual]`

Creates an instance of transcoder that can be used for transcoding operations. Each instance returned can be used for single transcoding operation. The instance can not be used for multiple transcoding operation.

Parameters

in	<i>input</i>	configuration of input buffers that needs to be transcoded.
in	<i>output</i>	configuration of transcoded output buffers.
in	<i>callback</i>	callback pointer to get the response of createTranscoder.

Returns

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

Note

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

5.14.1.17.2.7 `virtual telux::common::Status telux::audio::IAudioManager::deleteStream ( std::shared_ptr< IAudioStream > stream, DeleteStreamResponseCb callback = nullptr ) [pure virtual]`

Deletes the specified stream which was created before

Parameters

in	<i>stream</i>	reference to stream to be deleted.
in	<i>callback</i>	callback pointer to get the response of deleteStream.

Returns

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

5.14.1.17.2.8 `virtual telux::common::Status telux::audio::IAudioManager::registerListener ( std::weak_ptr< IAudioListener > listener ) [pure virtual]`

Register a listener to get notified when service status changes.

Parameters

in	<i>listener</i>	Pointer of IServiceListener object that processes the notification

Returns

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

Note

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

5.14.1.17.2.9 **virtual telux::common::Status telux::audio::IAudioManager::deRegisterListener (std::weak_ptr< IAudioListener > *listener*) [pure virtual]**

Remove a previously registered listener.

Parameters

in	<i>listener</i>	Previously registered IServiceListener that needs to be removed

Returns

Status of deRegisterListener, success or suitable status code

Note

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

5.14.1.18 **class telux::audio::IAudioDevice**

Audio device and it's characteristics like Direction (Sink or Source), type.

Public member functions

- virtual [DeviceType](#) [getType](#) ()=0
- virtual [DeviceDirection](#) [getDirection](#) ()=0
- virtual [~IAudioDevice](#) ()

5.14.1.18.1 **Constructors and Destructors**

5.14.1.18.1.1 **virtual telux::audio::IAudioDevice::~IAudioDevice () [virtual]**

5.14.1.18.2 **Member Function Documentation**

5.14.1.18.2.1 virtual DeviceType telux::audio::IAudioDevice::getType () [pure virtual]

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

Returns

DeviceType

5.14.1.18.2.2 virtual DeviceDirection telux::audio::IAudioDevice::getDirection () [pure virtual]

Provide direction of device whether is Sink for audio data (RX i.e. speaker, etc) or Source for audio data (TX i.e. mic, etc)

Returns

DeviceDirection

5.14.1.19 class telux::audio::IAudioStream

[IAudioStream](#) represents single audio stream with base properties.

Public member functions

- virtual [StreamType](#) getType ()=0
- virtual [telux::common::Status](#) setDevice (std::vector< [DeviceType](#) > devices, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) getDevice ([GetStreamDeviceResponseCb](#) callback=nullptr)=0
- virtual [telux::common::Status](#) setVolume ([StreamVolume](#) volume, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) getVolume ([StreamDirection](#) dir, [GetStreamVolumeResponseCb](#) callback=nullptr)=0
- virtual [telux::common::Status](#) setMute ([StreamMute](#) mute, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) getMute ([StreamDirection](#) dir, [GetStreamMuteResponseCb](#) callback=nullptr)=0
- virtual [~IAudioStream](#) ()

5.14.1.19.1 Constructors and Destructors

5.14.1.19.1.1 virtual telux::audio::IAudioStream::~IAudioStream () [virtual]

5.14.1.19.2 Member Function Documentation

5.14.1.19.2.1 virtual StreamType telux::audio::IAudioStream::getType () [pure virtual]

Get the stream type like VOICE, PLAY, CAPTURE

Returns

StreamType

5.14.1.19.2.2 virtual telux::common::Status telux::audio::IAudioStream::setDevice (std::vector< DeviceType > *devices*, telux::common::ResponseCallback *callback* = *nullptr*) [pure virtual]

Set Device of audio stream

Parameters

in	<i>devices</i>	Devices list.
in	<i>callback</i>	callback to get the response of setDevice.

Returns

Status of the request i.e. success or suitable status code.

5.14.1.19.2.3 virtual telux::common::Status telux::audio::IAudioStream::getDevice (GetStreamDevice-ResponseCb *callback* = *nullptr*) [pure virtual]

Get Device of audio stream

Parameters

in	<i>callback</i>	callback to get the response of getDevice

Returns

Status of the request i.e. success or suitable status code.

5.14.1.19.2.4 virtual telux::common::Status telux::audio::IAudioStream::setVolume (StreamVolume *volume*, telux::common::ResponseCallback *callback* = *nullptr*) [pure virtual]

Set Volume of audio stream

Parameters

in	<i>volume</i>	volume setting per channel for direction.
in	<i>callback</i>	callback to get the response of setVolume.

Returns

Status of the request i.e. success or suitable status code.

5.14.1.19.2.5 virtual telux::common::Status telux::audio::IAudioStream::getVolume (StreamDirection *dir*, GetStreamVolumeResponseCb *callback* = nullptr) [pure virtual]

Get Volume of audio stream

Parameters

in	<i>dir</i>	Stream Direction to query volume details.
in	<i>callback</i>	callback to get the response of getVolume.

Returns

Status of the request i.e. success or suitable status code.

5.14.1.19.2.6 virtual telux::common::Status telux::audio::IAudioStream::setMute (StreamMute *mute*, telux::common::ResponseCallback *callback* = nullptr) [pure virtual]

Set Mute of audio stream

Parameters

in	<i>mute</i>	mute setting for direction.
in	<i>callback</i>	callback to know the status of the request.

Returns

Status of the request i.e. success or suitable status code.

5.14.1.19.2.7 virtual telux::common::Status telux::audio::IAudioStream::getMute (StreamDirection *dir*, GetStreamMuteResponseCb *callback* = nullptr) [pure virtual]

Get Mute of audio stream

Parameters

in	<i>dir</i>	Stream Direction to query mute details.
in	<i>callback</i>	callback to get the response of getMute.

Returns

Status of the request i.e. success or suitable status code.

5.14.1.20 class telux::audio::IAudioVoiceStream

[IAudioVoiceStream](#) represents single voice stream.

Public member functions

- virtual `telux::common::Status startAudio (telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status stopAudio (telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status playDtmfTone (DtmfTone dtmfTone, uint16_t duration, uint16_t gain, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status stopDtmfTone (StreamDirection direction, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status registerListener (std::weak_ptr< IVoiceListener > listener, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status deRegisterListener (std::weak_ptr< IVoiceListener > listener)=0`
- virtual `~IAudioVoiceStream ()`

5.14.1.20.1 Constructors and Destructors

5.14.1.20.1.1 virtual `telux::audio::IAudioVoiceStream::~IAudioVoiceStream ( ) [virtual]`

5.14.1.20.2 Member Function Documentation

5.14.1.20.2.1 virtual `telux::common::Status telux::audio::IAudioVoiceStream::startAudio (telux::common::ResponseCallback callback = nullptr) [pure virtual]`

Starts audio stream

Parameters

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

Returns

Status of the request i.e. success or suitable status code.

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

Stops audio stream

Parameters

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

Returns

Status of the request i.e. success or suitable status code.

5.14.1.20.2.3 `virtual telux::common::Status telux::audio::IAudioVoiceStream::playDtmfTone ( DtmfTone dtmfTone, uint16_t duration, uint16_t gain, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Plays in-band DTMF tone on the active voice stream

Parameters

in	<i>dtmfTone</i>	DTMF tone properties [in] duration Duration (in milliseconds) for which the tone needs to be played. The constant infiniteDtmfDuration(=0xFFFF) represents infinite duration. [in] gain DTMF tone gain [in] callback callback to get the response of playDtmfTone.

Returns

Status of the request i.e. success or suitable status code.

5.14.1.20.2.4 `virtual telux::common::Status telux::audio::IAudioVoiceStream::stopDtmfTone ( StreamDirection direction, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Stops the DTMF tone which is being played (i.e duration not expired) on the active voice stream

Parameters

in	<i>direction</i>	Direction associated with the DTMF tone @ [in] callback callback to get the response of stopDtmfTone.

Returns

Status of the request i.e. success or suitable status code.

5.14.1.20.2.5 `virtual telux::common::Status telux::audio::IAudioVoiceStream::registerListener ( std::weak_ptr< IVoiceListener > listener, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Register a listener to get notified when a DTMF tone is detected in the active voice stream

Parameters

in	<i>listener</i>	Pointer of IVoiceListener object that processes the notification [in] callback callback to get the response of registerListener

Returns

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

5.14.1.20.2.6 `virtual telux::common::Status telux::audio::IAudioVoiceStream::deRegisterListener ( std::weak_ptr< IVoiceListener > listener ) [pure virtual]`

Remove a previously registered listener.

Parameters

in	<i>listener</i>	Previously registered IVoiceListener that needs to be removed

Returns

Status of deRegisterListener, success or suitable status code

5.14.1.21 class telux::audio::IAudioPlayStream

[IAudioPlayStream](#) represents single audio playback stream.

Public member functions

- virtual `std::shared_ptr< IStreamBuffer > getStreamBuffer ()=0`
- virtual `telux::common::Status write (std::shared_ptr< IStreamBuffer > buffer, WriteResponseCb callback=nullptr)=0`
- virtual `telux::common::Status stopAudio (StopType stopType, telux::common::ResponseCallback callback=nullptr)=0`
- virtual `telux::common::Status registerListener (std::weak_ptr< IPlayListener > listener)=0`
- virtual `telux::common::Status deRegisterListener (std::weak_ptr< IPlayListener > listener)=0`
- virtual `~IAudioPlayStream ()`

5.14.1.21.1 Constructors and Destructors

5.14.1.21.1.1 `virtual telux::audio::IAudioPlayStream::~IAudioPlayStream ( ) [virtual]`

5.14.1.21.2 Member Function Documentation

5.14.1.21.2.1 `virtual std::shared_ptr<IStreamBuffer> telux::audio::IAudioPlayStream::getStreamBuffer ( ) [pure virtual]`

Get an Audio [StreamBuffer](#) to be used for playback operations

Returns

an Audio Buffer or a nullptr in case of error

5.14.1.21.2.2 `virtual telux::common::Status telux::audio::IAudioPlayStream::write ( std::shared_ptr< IStreamBuffer > buffer, WriteResponseCb callback = nullptr ) [pure virtual]`

Write Samples to audio stream. First write starts playback operation.

Write in case of compressed audio format maintains a pipeline, if the callback returns with same number of bytes written as requested and no error occurred, user can send next buffer. If the number of bytes returned are not equal to the requested write size, then need to resend the buffer again from the leftover offset after waiting for the () event.

Parameters

in	<i>buffer</i>	stream buffer for write.
in	<i>callback</i>	callback to get the response of write.

Returns

Status of the request i.e. success or suitable status code.

5.14.1.21.2.3 `virtual telux::common::Status telux::audio::IAudioPlayStream::stopAudio ( StopType stopType, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

This API is to be used to stop playback. It is applicable only for compressed audio format playback.

Parameters

in	<i>callback</i>	callback to get the response of stopAudio.
in	<i>stopType</i>	it specifies type of stop for stopping audio playback.

Returns

Status of the request i.e. success or suitable status code.

Note

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

5.14.1.21.2.4 `virtual telux::common::Status telux::audio::IAudioPlayStream::registerListener ( std::weak_ptr< IPlayListener > listener ) [pure virtual]`

Register a listener to get notified for events of Play Stream

Parameters

in	<i>listener</i>	Pointer of IPlayListener 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.

5.14.1.21.2.5 **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 IPlayListener that needs to be removed

Returns

Status of deRegisterListener, success or suitable status code

Note

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

5.14.1.22 **class telux::audio::IAudioCaptureStream**

[IAudioCaptureStream](#) represents single audio capture stream.

Public member functions

- virtual std::shared_ptr< [IStreamBuffer](#) > [getStreamBuffer](#) ()=0
- virtual [telux::common::Status read](#) (std::shared_ptr< [IStreamBuffer](#) > buffer, uint32_t bytesToRead, [ReadResponseCb](#) callback=nullptr)=0
- virtual [~IAudioCaptureStream](#) ()

5.14.1.22.1 **Constructors and Destructors**

5.14.1.22.1.1 **virtual telux::audio::IAudioCaptureStream::~~IAudioCaptureStream () [virtual]**

5.14.1.22.2 **Member Function Documentation**

5.14.1.22.2.1 **virtual std::shared_ptr<IStreamBuffer> telux::audio::IAudioCaptureStream::getStreamBuffer () [pure virtual]**

Get an Audio Stream Buffer to be used for capture operations

Returns

an Audio Buffer or nullptr in case of failure

5.14.1.22.2.2 `virtual telux::common::Status telux::audio::IAudioCaptureStream::read ( std::shared_ptr< IStreamBuffer > buffer, uint32_t bytesToRead, ReadResponseCb callback = nullptr ) [pure virtual]`

Read Samples from audio stream. First read starts capture operation.

Parameters

in	<i>buffer</i>	stream buffer for read.
in	<i>bytesToRead</i>	specifying how many bytes to be read from stream.
in	<i>callback</i>	callback to get the response of read.

Returns

Status of the request i.e. success or suitable status code.

5.14.1.23 class telux::audio::IAudioLoopbackStream

[IAudioLoopbackStream](#) represents audio loopback stream.

Public member functions

- virtual [telux::common::Status startLoopback](#) ([telux::common::ResponseCallback](#) *callback*=nullptr)=0
- virtual [telux::common::Status stopLoopback](#) ([telux::common::ResponseCallback](#) *callback*=nullptr)=0
- virtual [~IAudioLoopbackStream](#) ()

5.14.1.23.1 Constructors and Destructors

5.14.1.23.1.1 `virtual telux::audio::IAudioLoopbackStream::~~IAudioLoopbackStream ( ) [virtual]`

5.14.1.23.2 Member Function Documentation

5.14.1.23.2.1 `virtual telux::common::Status telux::audio::IAudioLoopbackStream::startLoopback ( telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Start loopback between source and sink devices

Parameters

in	<i>callback</i>	callback to get the response of start loopback.

Returns

Status of the request i.e. success or suitable status code.

5.14.1.23.2.2 virtual telux::common::Status telux::audio::IAudioLoopbackStream::stopLoopback (telux::common::ResponseCallback *callback* = nullptr) [pure virtual]

Stop loopback between source and sink devices

Parameters

in	<i>callback</i>	callback to get the response of stop loopback.

Returns

Status of the request i.e. success or suitable status code.

5.14.1.24 class telux::audio::IAudioToneGeneratorStream

[IAudioToneGeneratorStream](#) represents tone generator stream.

Public member functions

- virtual [telux::common::Status](#) [playTone](#) (std::vector< uint16_t > *freq*, uint16_t *duration*, uint16_t *gain*, [telux::common::ResponseCallback](#) *callback*=nullptr)=0
- virtual [telux::common::Status](#) [stopTone](#) ([telux::common::ResponseCallback](#) *callback*=nullptr)=0
- virtual [~IAudioToneGeneratorStream](#) ()

5.14.1.24.1 Constructors and Destructors

5.14.1.24.1.1 virtual telux::audio::IAudioToneGeneratorStream::~~IAudioToneGeneratorStream () [virtual]

5.14.1.24.2 Member Function Documentation

5.14.1.24.2.1 virtual telux::common::Status telux::audio::IAudioToneGeneratorStream::playTone (std::vector< uint16_t > *freq*, uint16_t *duration*, uint16_t *gain*, [telux::common::ResponseCallback](#) *callback* = nullptr) [pure virtual]

Play a tone on sink devices. As the duration expires, the generated tone terminates automatically.

Parameters

in	<i>freq</i>	Accepts the composition of frequencies (in Hz) to be played such as single tone or dual tone. Any additional frequencies provided will be ignored.
in	<i>duration</i>	Duration (in milliseconds) for which the tone needs to be played. The constant infiniteToneDuration(=0xFFFF) represents infinite duration.
in	<i>gain</i>	Tone Gain.
in	<i>callback</i>	callback to get the response of play tone.

Returns

Status of the request i.e. success or suitable status code.

5.14.1.24.2.2 `virtual telux::common::Status telux::audio::IAudioToneGeneratorStream::stopTone (telux::common::ResponseCallback callback = nullptr) [pure virtual]`

Stops the tone which is being played (i.e duration not expired) on the active Tone generator stream.

Parameters

<i>in</i>	<i>callback</i>	callback to get the response of stop tone.

Returns

Status of the request i.e. success or suitable status code.

5.14.1.25 class telux::audio::ITranscoder

[ITranscoder](#) is used to convert one audio format to another audio format using the transcoding operation.

Public member functions

- virtual std::shared_ptr< [IAudioBuffer](#) > [getWriteBuffer](#) ()=0
- virtual std::shared_ptr< [IAudioBuffer](#) > [getReadBuffer](#) ()=0
- virtual [telux::common::Status](#) [write](#) (std::shared_ptr< [IAudioBuffer](#) > buffer, uint32_t isLastBuffer, [TranscoderWriteResponseCb](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [tearDown](#) ([telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [read](#) (std::shared_ptr< [IAudioBuffer](#) > buffer, uint32_t bytesToRead, [TranscoderReadResponseCb](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [registerListener](#) (std::weak_ptr< [ITranscodeListener](#) > listener)=0
- virtual [telux::common::Status](#) [deRegisterListener](#) (std::weak_ptr< [ITranscodeListener](#) > listener)=0
- virtual [~ITranscoder](#) ()

5.14.1.25.1 Constructors and Destructors

5.14.1.25.1.1 `virtual telux::audio::ITranscoder::~~ITranscoder ( ) [virtual]`

Destructor for [ITranscoder](#)

5.14.1.25.2 Member Function Documentation

5.14.1.25.2.1 `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.

5.14.1.25.2.2 `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.

5.14.1.25.2.3 `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.

5.14.1.25.2.4 **virtual telux::common::Status telux::audio::ITranscoder::tearDown (telux::common::ResponseCallback *callback* = *nullptr*) [pure virtual]**

It is mandatory to call this API after the end of a transcode operation or to abort a transcode operation. After this API call the [ITranscoder](#) object is no longer usable.

Parameters

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

Returns

Status of the request i.e. success or suitable status code.

Note

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

5.14.1.25.2.5 **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.

5.14.1.25.2.6 `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 ITranscodeListener 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.

5.14.1.25.2.7 `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 ITranscodeListener that needs to be removed.

Returns

Status of deRegisterListener, success or suitable status code.

Note

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

5.14.2 Enumeration Type Documentation

5.14.2.1 `enum telux::audio::DeviceType`

Represent type of device like SPEAKER, MIC, etc.

Enumerator

DEVICE_TYPE_NONE
DEVICE_TYPE_SPEAKER
DEVICE_TYPE_MIC
DEVICE_TYPE_OUT_VIRTUAL

5.14.2.2 enum telux::audio::DeviceDirection

Represent Device Direction RX (Sink), Tx (Source)

Enumerator

NONE

RX

TX

5.14.2.3 enum telux::audio::Direction

Represent Voice Direction RX (Sink), Tx (Source)

Enumerator

RX

TX

5.14.2.4 enum telux::audio::StreamType

Represent Stream Type

Enumerator

NONE

VOICE_CALL Voice Call, Provides Audio Session for an active Voice

PLAY Plaback, 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

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

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

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

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

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

Represents type of stop for compressed audio format playback. Audio playback can be stopped in two ways force stop and after playing all buffers in the pipeline.

Enumerator

FORCE_STOP Stop Playing Immediately and clear buffer pipeline

STOP_AFTER_PLAY Stop Play once after all buffers in pipeline are played

5.15 Thermal Management

This section contains APIs related to Thermal Management such as read list of thermal zones, cooling devices and binding info.

This section contains APIs related to Thermal Shutdown Management such as set/get thermal auto-shutdown mode, receive notifications on every auto-shutdown update.

5.15.1 Data Structure Documentation

5.15.1.1 class telux::therm::ThermalFactory

[ThermalFactory](#) allows creation of thermal manager.

Public member functions

- `std::shared_ptr< IThermalManager > getThermalManager ()`
- `std::shared_ptr< IThermalShutdownManager > getThermalShutdownManager ()`

Static Public Member Functions

- static `ThermalFactory & getInstance ()`

5.15.1.1.1 Member Function Documentation

5.15.1.1.1.1 static `ThermalFactory& telux::therm::ThermalFactory::getInstance ( ) [static]`

Get Thermal Factory instance.

5.15.1.1.1.2 `std::shared_ptr<IThermalManager> telux::therm::ThermalFactory::getThermalManager ( )`

Get thermal manager instance to get list of thermal zones (sensors) and cooling devices supported by the device

Returns

Pointer of [IThermalManager](#) object.

5.15.1.1.1.3 `std::shared_ptr<IThermalShutdownManager> telux::therm::ThermalFactory::getThermalShutdownManager ( )`

Get thermal shutdown manager instance to control automatic thermal shutdown and get relevant notifications

Returns

Pointer of [IThermalShutdownManager](#) object.

5.15.1.2 struct telux::therm::BoundCoolingDevice

Defines the trip points to which cooling device is bound.

Data fields

Type	Field	Description
int	coolingDevice-Id	Cooling device Id associated with trip points
vector< shared_ptr< ITripPoint > >	bindingInfo	List of trippoints bound to the cooling device

5.15.1.3 class telux::therm::IThermalManager

[IThermalManager](#) provides interface to get thermal zone and cooling device information.

Public member functions

- virtual std::vector< std::shared_ptr< [IThermalZone](#) > > [getThermalZones](#) ()=0
- virtual std::vector< std::shared_ptr< [ICoolingDevice](#) > > [getCoolingDevices](#) ()=0
- virtual std::shared_ptr< [IThermalZone](#) > [getThermalZone](#) (int thermalZoneId)=0
- virtual std::shared_ptr< [ICoolingDevice](#) > [getCoolingDevice](#) (int coolingDeviceId)=0
- virtual [~IThermalManager](#) ()

5.15.1.3.1 Constructors and Destructors

5.15.1.3.1.1 virtual telux::therm::IThermalManager::~IThermalManager () [[virtual](#)]

Destructor of [IThermalManager](#)

5.15.1.3.2 Member Function Documentation

5.15.1.3.2.1 virtual std::vector< std::shared_ptr< [IThermalZone](#) > > telux::therm::IThermalManager::getThermalZones () [[pure virtual](#)]

Retrieves the list of thermal zone info like type, temperature and trip points.

Returns

List of thermal zones.

5.15.1.3.2.2 `virtual std::vector<std::shared_ptr<ICoolingDevice> > telux::therm::IThermalManager::getCoolingDevices ( ) [pure virtual]`

Retrieves the list of thermal cooling device info like type, maximum throttle state and currently requested throttle state.

Returns

List of cooling devices.

5.15.1.3.2.3 `virtual std::shared_ptr<IThermalZone> telux::therm::IThermalManager::getThermalZone ( int thermalZoneId ) [pure virtual]`

Retrieves the thermal zone details like temperature, type and trip point info for the given thermal zone identifier.

Parameters

in	<i>thermalZoneId</i>	Thermal zone identifier

Returns

Pointer to thermal zone.

5.15.1.3.2.4 `virtual std::shared_ptr<ICoolingDevice> telux::therm::IThermalManager::getCoolingDevice ( int coolingDeviceId ) [pure virtual]`

Retrieves the cooling device details like type of the device, maximum cooling level and current cooling level for the given cooling device identifier.

Parameters

in	<i>coolingDeviceId</i>	Cooling device identifier

Returns

Pointer to cooling device.

5.15.1.4 `class telux::therm::ITripPoint`

[ITripPoint](#) provides interface to get trip point type, trip point temperature and hysteresis value for that trip point.

Public member functions

- virtual [TripType](#) `getType ()=0`
- virtual int `getThresholdTemp ()=0`
- virtual int `getHysteresis ()=0`

- virtual [~ITripPoint](#) ()

5.15.1.4.1 Constructors and Destructors

5.15.1.4.1.1 virtual telux::therm::ITripPoint::~ITripPoint () [virtual]

Destructor of [ITripPoint](#)

5.15.1.4.2 Member Function Documentation

5.15.1.4.2.1 virtual TripType telux::therm::ITripPoint::getType () [pure virtual]

Retrieves trip point type.

Returns

Type of trip point if available else return UNKNOWN.

- [TripType](#)

5.15.1.4.2.2 virtual int telux::therm::ITripPoint::getThresholdTemp () [pure virtual]

Retrieves the temperature above which certain trip point will be fired.

- Units: MilliDegree Celsius

Returns

Threshold temperature

5.15.1.4.2.3 virtual int telux::therm::ITripPoint::getHysteresis () [pure virtual]

Retrieves hysteresis value that is the difference between current temperature of the device and the temperature above which certain trip point will be fired. Units: MilliDegree Celsius

Returns

Hysteresis value

5.15.1.5 class telux::therm::IThermalZone

[IThermalZone](#) provides interface to get type of the sensor, the current temperature reading, trip points and the cooling devices binded etc.

Public member functions

- virtual int [getId](#) ()=0
- virtual std::string [getDescription](#) ()=0

- virtual int `getCurrentTemp` ()=0
- virtual int `getPassiveTemp` ()=0
- virtual std::vector
`< std::shared_ptr< ITripPoint > > getTripPoints` ()=0
- virtual std::vector
`< BoundCoolingDevice > getBoundCoolingDevices` ()=0
- virtual `~IThermalZone` ()

5.15.1.5.1 Constructors and Destructors

5.15.1.5.1.1 `virtual telux::therm::IThermalZone::~~IThermalZone ( ) [virtual]`

Destructor of `IThermalZone`

5.15.1.5.2 Member Function Documentation

5.15.1.5.2.1 `virtual int telux::therm::IThermalZone::getId ( ) [pure virtual]`

Retrieves the identifier for thermal zone.

Returns

Identifier for thermal zone

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

Retrieves the type of sensor.

Returns

Sensor type

5.15.1.5.2.3 `virtual int telux::therm::IThermalZone::getCurrentTemp ( ) [pure virtual]`

Retrieves the current temperature of the device. Units: MilliDegree Celsius

Returns

Current temperature

5.15.1.5.2.4 virtual int telux::therm::IThermalZone::getPassiveTemp () [pure virtual]

Retrieves the temperature of passive trip point for the zone. Default value is 0. Valid values: 0 (disabled) or greater than 1000 (enabled), Units: MilliDegree Celsius

Returns

Temperature of passive trip point

5.15.1.5.2.5 virtual std::vector<std::shared_ptr<ITripPoint> > telux::therm::IThermalZone::getTripPoints () [pure virtual]

Retrieves trip point information like trip type, trip temperature and hysteresis.

Returns

Trip point info list

5.15.1.5.2.6 virtual std::vector<BoundCoolingDevice> telux::therm::IThermalZone::getBoundCoolingDevices () [pure virtual]

Retrieves the list of cooling device and the associated trip points bound to cooling device in given thermal zone.

Returns

List of bound cooling device for the given thermal zone.

5.15.1.6 class telux::therm::ICoolingDevice

[ICoolingDevice](#) provides interface to get type of the cooling device, the maximum throttle state and the currently requested throttle state of the cooling device.

Public member functions

- virtual int [getId](#) ()=0
- virtual std::string [getDescription](#) ()=0
- virtual int [getMaxCoolingLevel](#) ()=0
- virtual int [getCurrentCoolingLevel](#) ()=0
- virtual [~ICoolingDevice](#) ()

5.15.1.6.1 Constructors and Destructors**5.15.1.6.1.1 virtual telux::therm::ICoolingDevice::~~ICoolingDevice () [virtual]**

Destructor of [ICoolingDevice](#)

5.15.1.6.2 Member Function Documentation

5.15.1.6.2.1 `virtual int telux::therm::ICoolingDevice::getId ( ) [pure virtual]`

Retrieves the identifier of the thermal cooling device.

Returns

Cooling device identifier

5.15.1.6.2.2 `virtual std::string telux::therm::ICoolingDevice::getDescription ( ) [pure virtual]`

Retrieves the type of the cooling device.

Returns

Cooling device type

5.15.1.6.2.3 `virtual int telux::therm::ICoolingDevice::getMaxCoolingLevel ( ) [pure virtual]`

Retrieves the maximum cooling level of the cooling device.

Returns

Maximum cooling level of the thermal cooling device

5.15.1.6.2.4 `virtual int telux::therm::ICoolingDevice::getCurrentCoolingLevel ( ) [pure virtual]`

Retrieves the current cooling level of the cooling device. This value can be between 0 and max cooling level. Max cooling level is different for different cooling devices like fan, processor etc.

Returns

Current cooling level of the thermal cooling device

5.15.1.7 `class telux::therm::IThermalShutdownListener`

Listener class for getting notifications when automatic thermal shutdown mode is enabled/ disabled or will be enabled imminently. The client needs to implement these methods as briefly as possible and avoid blocking calls in it. The methods in this class can be invoked from multiple different threads. Client needs to make sure that the implementation is thread-safe.

Public member functions

- virtual void `onShutdownEnabled ( )`
- virtual void `onShutdownDisabled ( )`
- virtual void `onImminentShutdownEnablement (uint32_t imminentDuration)`

- virtual [~IThermalShutdownListener](#) ()

5.15.1.7.1 Constructors and Destructors

5.15.1.7.1.1 virtual [telux::therm::IThermalShutdownListener::~~IThermalShutdownListener](#) ()
[virtual]

Destructor of [IThermalShutdownListener](#)

5.15.1.7.2 Member Function Documentation

5.15.1.7.2.1 virtual void [telux::therm::IThermalShutdownListener::onShutdownEnabled](#) ()
[virtual]

This function is called when the automatic shutdown mode changes to ENABLE

5.15.1.7.2.2 virtual void [telux::therm::IThermalShutdownListener::onShutdownDisabled](#) ()
[virtual]

This function is called when the automatic shutdown mode changes to DISABLE

5.15.1.7.2.3 virtual void [telux::therm::IThermalShutdownListener::onImminentShutdownEnablement](#) ([uint32_t imminentDuration](#)) [virtual]

This function is called when the automatic shutdown mode is about to change to ENABLE. Clients that want to keep the shutdown mode disabled, needs to set it accordingly with in the imminentDuration time. If disabled successfully within imminentDuration time, the system timer for auto-enablement will be reset.

Parameters

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

5.15.1.8 class [telux::therm::IThermalShutdownManager](#)

[IThermalShutdownManager](#) class provides interface to enable/disable automatic thermal shutdown. Additionally it facilitates to register for notifications when the automatic shutdown mode changes.

Public member functions

- virtual bool [isReady](#) ()=0
- virtual std::future< bool > [onReady](#) ()=0
- virtual [telux::common::Status registerListener](#) (std::weak_ptr< [IThermalShutdownListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak_ptr< [IThermalShutdownListener](#) > listener)=0

- virtual `telux::common::Status setAutoShutdownMode (AutoShutdownMode mode, telux::common::ResponseCallback callback=nullptr, uint32_t timeout=DEFAULT_TIMEOUT)=0`
- virtual `telux::common::Status getAutoShutdownMode (GetAutoShutdownModeResponseCb callback)=0`
- virtual `~IThermalShutdownManager ()`

5.15.1.8.1 Constructors and Destructors

5.15.1.8.1.1 `virtual telux::therm::IThermalShutdownManager::~IThermalShutdownManager ( ) [virtual]`

Destructor of `IThermalShutdownManager`

5.15.1.8.2 Member Function Documentation

5.15.1.8.2.1 `virtual bool telux::therm::IThermalShutdownManager::isReady ( ) [pure virtual]`

Checks the status of thermal shutdown management service and if the other APIs are ready for use and returns the result.

Returns

True if the services are ready otherwise false.

5.15.1.8.2.2 `virtual std::future<bool> telux::therm::IThermalShutdownManager::onReady ( ) [pure virtual]`

Wait for thermal shutdown management service to be ready.

Returns

A future that caller can wait on to be notified when thermal shutdown management service is ready.

5.15.1.8.2.3 `virtual telux::common::Status telux::therm::IThermalShutdownManager::registerListener ( std::weak_ptr< IThermalShutdownListener > listener ) [pure virtual]`

Register a listener for updates on automatic shutdown mode changes

Parameters

<i>in</i>	<i>listener</i>	Pointer of <code>IThermalShutdownListener</code> object that processes the notification

Returns

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

5.15.1.8.2.4 `virtual telux::common::Status telux::therm::IThermalShutdownManager::deregisterListener ( std::weak_ptr< IThermalShutdownListener > listener ) [pure virtual]`

Remove a previously registered listener.

Parameters

in	<i>listener</i>	Previously registered IThermalShutdownListener that needs to be removed

Returns

Status of deregisterListener, success or suitable status code

5.15.1.8.2.5 `virtual telux::common::Status telux::therm::IThermalShutdownManager::setAutoShutdownMode ( AutoShutdownMode mode, telux::common::ResponseCallback callback = nullptr, uint32_t timeout = DEFAULT_TIMEOUT ) [pure virtual]`

Set automatic thermal shutdown mode. When set to DISABLE mode successfully, it remains in DISABLE mode briefly and automatically changes to ENABLE mode after notifying the clients.

Parameters

in	<i>mode</i>	desired AutoShutdownMode to be set
in	<i>callback</i>	Optional callback to get the response of the command
in	<i>timeout</i>	Optional timeout(in seconds) for which auto-shutdown remains disabled.

Returns

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

5.15.1.8.2.6 `virtual telux::common::Status telux::therm::IThermalShutdownManager::getAutoShutdownMode ( GetAutoShutdownModeResponseCb callback ) [pure virtual]`

Get automatic thermal shutdown mode.

Parameters

in	<i>callback</i>	GetAutoShutdownModeResponseCb to get response of the request

Returns

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

5.15.2 Enumeration Type Documentation

5.15.2.1 enum telux::therm::AutoShutdownMode

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

Defines the type of trip points, it can be one of the values for ACPI (Advanced Configuration and Power Interface) thermal zone

Enumerator

UNKNOWN Trip type is unknown
CRITICAL Trip point at which system shuts down
HOT Trip point to notify emergency
PASSIVE Trip point at which kernel lowers the CPU's frequency and throttle the processor down
ACTIVE Trip point at which processor fan turns on
CONFIGURABLE_HIGH Triggering threshold at which mitigation starts. This type is added to support legacy targets
CONFIGURABLE_LOW Clearing threshold at which mitigation stops. This type is added to support legacy targets

5.15.3 Variable Documentation

5.15.3.1 const uint32_t telux::therm::DEFAULT_TIMEOUT = 30

Default time out (in seconds) for thermal auto-shutdown service to re-enable thermal auto-shutdown.

5.16 TCU Activity Manager

This section contains APIs related to TCU activity state management.

5.16.1 Data Structure Documentation

5.16.1.1 class `telux::power::PowerFactory`

`PowerFactory` allows creation of TCU-activity manager instance.

Public member functions

- `std::shared_ptr`
`< ITcuActivityManager > getTcuActivityManager (ClientType clientType=ClientType::SLAVE)`

Static Public Member Functions

- static `PowerFactory & getInstance ()`

5.16.1.1.1 Member Function Documentation

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

API to get the factory instance for TCU-activity management

5.16.1.1.1.2 `std::shared_ptr<ITcuActivityManager> telux::power::PowerFactory::getTcuActivityManager ( ClientType clientType = ClientType::SLAVE )`

API to get the TCU-activity Manager instance

Parameters

in	type	Type of the client that is going to access <code>ITcuActivityManager</code> APIs <code>ClientType</code>

Returns

Pointer of `ITcuActivityManager` object.

5.16.1.2 class `telux::power::ITcuActivityListener`

Listener class for getting notifications related to TCU-activity state and also the updates related to TCU-activity service status. The client needs to implement these methods as briefly as possible and avoid blocking calls in it. The methods in this class can be invoked from multiple different threads. Client needs to make sure that the implementation is thread-safe.

Public member functions

- virtual void [onTcuActivityStateUpdate](#) ([TcuActivityState](#) state)
- virtual void [onSlaveAckStatusUpdate](#) ([telux::common::Status](#) status)
- virtual [~ITcuActivityListener](#) ()

5.16.1.2.1 Constructors and Destructors

5.16.1.2.1.1 virtual [telux::power::ITcuActivityListener::~~ITcuActivityListener](#) () [[virtual](#)]

Destructor of [ITcuActivityListener](#)

5.16.1.2.2 Member Function Documentation

5.16.1.2.2.1 virtual void [telux::power::ITcuActivityListener::onTcuActivityStateUpdate](#) ([TcuActivity-State](#) *state*) [[virtual](#)]

This function is called when the TCU-activity state is going to change.

Parameters

in	<i>state</i>	TCU-activity state that system is about to enter

5.16.1.2.2.2 virtual void [telux::power::ITcuActivityListener::onSlaveAckStatusUpdate](#) ([telux::common::Status](#) *status*) [[virtual](#)]

This function is called with the overall acknowledgement status from all the SLAVE clients, for state change triggered previously by MASTER client.

This API will be invoked only for the MASTER client. If at least one SLAVE client does not acknowledge within the configured timeout, then [Status::EXPIRED](#) would be reported.

Parameters

in	<i>status</i>	status of the SLAVE clients acknowledgements

5.16.1.3 class [telux::power::ITcuActivityManager](#)

[ITcuActivityManager](#) provides interface to register and de-register listeners (to get TCU-activity state updates). And also API to initiate TCU-activity state transition.

Public member functions

- virtual bool [isReady](#) ()=0
- virtual std::future< bool > [onReady](#) ()=0
- virtual [telux::common::Status](#) [registerListener](#) (std::weak_ptr< [ITcuActivityListener](#) > listener)=0

- virtual [telux::common::Status deregisterListener](#) (std::weak_ptr< [ITcuActivityListener](#) > listener)=0
- virtual [telux::common::Status registerServiceStateListener](#) (std::weak_ptr< [telux::common::IServiceStatusListener](#) > listener)=0
- virtual [telux::common::Status deregisterServiceStateListener](#) (std::weak_ptr< [telux::common::IServiceStatusListener](#) > listener)=0
- virtual [telux::common::Status setActivityState](#) ([TcuActivityState](#) state, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [TcuActivityState getActivityState](#) ()=0
- virtual [telux::common::Status sendActivityStateAck](#) ([TcuActivityStateAck](#) ack)=0
- virtual [~ITcuActivityManager](#) ()

5.16.1.3.1 Constructors and Destructors

5.16.1.3.1.1 virtual [telux::power::ITcuActivityManager::~ITcuActivityManager](#) () [**virtual**]

Destructor of [ITcuActivityManager](#)

5.16.1.3.2 Member Function Documentation

5.16.1.3.2.1 virtual bool [telux::power::ITcuActivityManager::isReady](#) () [**pure virtual**]

Checks the status of TCU-activity services and if the other APIs are ready for use, and returns the result.

Returns

True if the services are ready otherwise false.

5.16.1.3.2.2 virtual std::future<bool> [telux::power::ITcuActivityManager::onReady](#) () [**pure virtual**]

Wait for TCU-activity services to be ready.

Returns

A future that caller can wait on to be notified when TCU-activity services are ready.

5.16.1.3.2.3 virtual [telux::common::Status telux::power::ITcuActivityManager::registerListener](#) (std::weak_ptr< [ITcuActivityListener](#) > *listener*) [**pure virtual**]

Register a listener for updates on TCU-activity state changes.

Parameters

in	<i>listener</i>	Pointer of ITcuActivityListener object that processes the notification

Returns

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

5.16.1.3.2.4 `virtual telux::common::Status telux::power::ITcuActivityManager::deregisterListener ( std::weak_ptr< ITcuActivityListener > listener ) [pure virtual]`

Remove a previously registered listener.

Parameters

in	<i>listener</i>	Previously registered ITcuActivityListener that needs to be removed

Returns

Status of deregisterListener, success or suitable status code

5.16.1.3.2.5 `virtual telux::common::Status telux::power::ITcuActivityManager::registerServiceStateListener ( std::weak_ptr< telux::common::IServiceStatusListener > listener ) [pure virtual]`

Register a listener for updates on TCU-activity management service status.

Parameters

in	<i>listener</i>	Pointer of IServiceStatusListener object that processes the notification

Returns

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

5.16.1.3.2.6 `virtual telux::common::Status telux::power::ITcuActivityManager::deregisterServiceStateListener ( std::weak_ptr< telux::common::IServiceStatusListener > listener ) [pure virtual]`

Remove a previously registered listener for service status updates.

Parameters

in	<i>listener</i>	Previously registered IServiceStatusListener that needs to be removed

Returns

Status of deregisterServiceStateListener, success or suitable status code

5.16.1.3.2.7 `virtual telux::common::Status telux::power::ITcuActivityManager::setActivityState ( TcuActivityState state, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Initiate a TCU-activity state transition.

This API needs to be used cautiously, as it could change the power-state of the system and may affect other processes.

This API is meant for clients that have instantiated the [ITcuActivityManager](#) instance using ClientType::MASTER

Parameters

in	<i>state</i>	TCU-activity state that the System is intended to enter
in	<i>callback</i>	Optional callback to get the response for the TCU-activity state transition command

Returns

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

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

Get the current TCU-activity state.

Returns

TcuActivityState

5.16.1.3.2.9 `virtual telux::common::Status telux::power::ITcuActivityManager::sendActivityStateAck ( TcuActivityStateAck ack ) [pure virtual]`

API to send the acknowledgement, after processing a TCU-activity state notification. This indicates that the client is prepared for state transition. Only one acknowledgement is expected from a single client process (may have multiple listeners).

Parameters

in	<i>ack</i>	Acknowledgement for a TCU-activity state notification.

Returns

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

5.16.2 Enumeration Type Documentation

5.16.2.1 enum telux::power::TcuActivityState

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

Defines the acknowledgements to TCU-activity states. The client process sends this after processing the TcuActivityState notification, indicating that it is prepared for state transition

Acknowledgement for TcuActivityState::RESUME is not required, as the state transition has already happened.

Enumerator

SUSPEND_ACK processed TcuActivityState::SUSPEND notification
SHUTDOWN_ACK processed TcuActivityState::SHUTDOWN notification

5.16.2.3 enum telux::power::ClientType

Defines the type of the client that would be using the [ITcuActivityManager](#) APIs. Client that just needs the TcuActivityState notifications needs to choose ClientType::SLAVE. And the client that determines the TcuActivityState would choose ClientType::MASTER. Only a Master client can set the TcuActivityState. In a system, there should be a single Master client.

The ClientType needs to be chosen while instantiating the [ITcuActivityManager](#), using the API [PowerFactory::getTcuActivityManager](#)

Enumerator

SLAVE Client is a slave and interested in state change notification
MASTER Client makes the decision on when the TcuActivityState should change

5.17 Remote SIM Provisioning

This section contains APIs related to Remote SIM provisioning.

5.17.1 Data Structure Documentation

5.17.1.1 class telux::tel::SimProfile

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

Public member functions

- [SimProfile](#) (int profileId, [ProfileType](#) profileType, const std::string &iccid, bool [isActive](#), const std::string &nickName, const std::string &spn, const std::string &name, [IconType](#) iconType, std::vector< uint8_t > icon, [ProfileClass](#) profileClass, [PolicyRuleMask](#) policyRuleMask)
- int [getSlotId](#) ()
- int [getProfileId](#) ()
- [ProfileType](#) [getType](#) ()
- const std::string & [getIccid](#) ()
- bool [isActive](#) ()
- const std::string & [getNickName](#) ()
- const std::string & [getSPN](#) ()
- const std::string & [getName](#) ()
- [IconType](#) [getIconType](#) ()
- std::vector< uint8_t > [getIcon](#) ()
- [ProfileClass](#) [getClass](#) ()
- [PolicyRuleMask](#) [getPolicyRule](#) ()
- std::string [toString](#) ()

5.17.1.1.1 Constructors and Destructors

5.17.1.1.1.1 `telux::tel::SimProfile::SimProfile (int profileId, ProfileType profileType, const std::string & iccid, bool isActive, const std::string & nickName, const std::string & spn, const std::string & name, IconType iconType, std::vector< uint8_t > icon, ProfileClass profileClass, PolicyRuleMask policyRuleMask)`

5.17.1.1.2 Member Function Documentation

5.17.1.1.2.1 int telux::tel::SimProfile::getSlotId ()

Get slot id associated for this profile

Returns

SlotId

5.17.1.1.2.2 int telux::tel::SimProfile::getProfileId ()

Get profile identifier.

Returns

unique identifier for the profile

5.17.1.1.2.3 ProfileType telux::tel::SimProfile::getType ()

Get profile Type.

Returns

profile type

5.17.1.1.2.4 const std::string& telux::tel::SimProfile::getIccid ()

Get profile ICCID.

Returns

profile ICCID coded as in EF-ICCID

5.17.1.1.2.5 bool telux::tel::SimProfile::isActive ()

Indicates the profile state whether active or not.

Returns

true if profile is Active

5.17.1.1.2.6 const std::string& telux::tel::SimProfile::getNickName ()

Get profile nick name.

Returns

profile nick name

5.17.1.1.2.7 const std::string& telux::tel::SimProfile::getSPN ()

Get profile service provider name.

Returns

profile service provider name.

5.17.1.1.2.8 const std::string& telux::tel::SimProfile::getName ()

Get profile name.

Returns

profile name

5.17.1.1.2.9 IconType telux::tel::SimProfile::getIconType ()

Get profile icon type.

Returns

profile icon type

5.17.1.1.2.10 std::vector<uint8_t> telux::tel::SimProfile::getIcon ()

Get profile icon content.

Returns

profile icon content

5.17.1.1.2.11 ProfileClass telux::tel::SimProfile::getClass ()

Get profile class.

Returns

profile class

5.17.1.1.2.12 PolicyRuleMask telux::tel::SimProfile::getPolicyRule ()

Get profile policy rules.

Returns

mask of profile policy rules

5.17.1.1.2.13 std::string telux::tel::SimProfile::toString ()

Get the text related informative representation of this object.

Returns

String containing informative string.

5.17.1.2 class telux::tel::ISimProfileListener

The interface listens for profile download indication and keep track of download and install progress of profile.

The methods in the listener can be invoked from multiple threads. It is client's responsibility to make sure the implementation is thread safe.

Public member functions

- virtual void [onAddProfileUpdate](#) (SlotId slotId, bool userConsentRequired, [DownloadStatus](#) status, uint8_t percentage, [DownloadErrorCause](#) cause, [PolicyRuleMask](#) mask)
- virtual [~ISimProfileListener](#) ()

5.17.1.2.1 Constructors and Destructors

5.17.1.2.1.1 virtual telux::tel::ISimProfileListener::~~ISimProfileListener () [virtual]

Destructor of [ISimProfileListener](#)

5.17.1.2.2 Member Function Documentation

5.17.1.2.2.1 virtual void telux::tel::ISimProfileListener::onAddProfileUpdate (SlotId *slotId*, bool *userConsentRequired*, [DownloadStatus](#) *status*, uint8_t *percentage*, [DownloadErrorCause](#) *cause*, [PolicyRuleMask](#) *mask*) [virtual]

This function is called when indication about status of profile download and installation comes.

Parameters

in	<i>slotId</i>	Slot on which profile get downloaded and installed.
in	<i>userConsent-Required</i>	User consent required or not.
in	<i>status</i>	ProfileDownloadStatus .
in	<i>percentage</i>	Download and installation percentage.
in	<i>cause</i>	ProfileDownloadErrorCause .
in	<i>mask</i>	PprMask (Profile policy rules Mask)

5.17.1.3 class telux::tel::ISimProfileManager

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

Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status addProfile](#) (SlotId slotId, const std::string &activationCode, const std::string &confirmationCode="", bool userConsentSupported=false, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status deleteProfile](#) (SlotId slotId, int profileId, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status setProfile](#) (SlotId slotId, int profileId, bool enable=false, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status updateNickName](#) (SlotId slotId, int profileId, const std::string &nickName, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status requestProfileList](#) (SlotId slotId, [ProfileListResponseCb](#) callback)=0
- virtual [telux::common::Status requestEid](#) (SlotId slotId, [EidResponseCb](#) callback)=0
- virtual [telux::common::Status provideUserConsent](#) (SlotId slotId, bool userConsent=false, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status registerListener](#) (std::weak_ptr< [ISimProfileListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak_ptr< [ISimProfileListener](#) > listener)=0
- virtual [~ISimProfileManager](#) ()

5.17.1.3.1 Constructors and Destructors

5.17.1.3.1.1 virtual telux::tel::ISimProfileManager::~ISimProfileManager () [virtual]

Destructor for [ISimProfileManager](#)

5.17.1.3.2 Member Function Documentation

5.17.1.3.2.1 virtual bool telux::tel::ISimProfileManager::isSubsystemReady () [pure virtual]

Checks if the eUICC subsystem is ready.

Returns

True if [ISimProfileManager](#) is ready for service, otherwise returns false.

Note

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

5.17.1.3.2.2 `virtual std::future<bool> telux::tel::ISimProfileManager::onSubsystemReady ( ) [pure virtual]`

Wait for eUICC subsystem to be ready.

Returns

A future that caller can wait on to be notified when card manager is ready.

Note

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

5.17.1.3.2.3 `virtual telux::common::Status telux::tel::ISimProfileManager::addProfile ( SlotId slotId, const std::string & activationCode, const std::string & confirmationCode = "", bool userConsentSupported = false, common::ResponseCallback callback = nullptr ) [pure virtual]`

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

Parameters

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

Returns

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

Note

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

5.17.1.3.2.4 `virtual telux::common::Status telux::tel::ISimProfileManager::deleteProfile ( SlotId slotId, int profileId, common::ResponseCallback callback = nullptr ) [pure virtual]`

Delete profile from eUICC card.

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

Returns

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

Note

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

5.17.1.3.2.5 `virtual telux::common::Status telux::tel::ISimProfileManager::setProfile ( SlotId slotId, int profileId, bool enable = false, common::ResponseCallback callback = nullptr ) [pure virtual]`

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

Parameters

in	<i>slotId</i>	Slot identifier corresponding to the card.
in	<i>profileId</i>	Profile identifier.
in	<i>enable</i>	Indicates whether a profile must be enabled or disabled. true - Enable and false - Disable.
in	<i>callback</i>	Callback function to get the result of set profile.

Returns

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

Note

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

5.17.1.3.2.6 `virtual telux::common::Status telux::tel::ISimProfileManager::updateNickName ( SlotId slotId, int profileId, const std::string & nickName, common::ResponseCallback callback = nullptr ) [pure virtual]`

Update nick name of the profile

Parameters

in	<i>slotId</i>	Slot identifier corresponding to the card.
in	<i>profileId</i>	Profile identifier
in	<i>nickName</i>	New nick name for profile.
in	<i>callback</i>	Callback function to get the result of update nickname.

Returns

Status of update nick name i.e. success or suitable error code.

Note

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

5.17.1.3.2.7 **virtual telux::common::Status telux::tel::ISimProfileManager::requestProfileList (SlotId slotId, ProfileListResponseCb callback) [pure virtual]**

Request list of profiles supported by the eUICC card.

Parameters

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

Returns

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

Note

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

5.17.1.3.2.8 **virtual telux::common::Status telux::tel::ISimProfileManager::requestEid (SlotId slotId, EidResponseCb callback) [pure virtual]**

Request eUICC identifier(EID) for the slot.

Parameters

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

Returns

Status of request EID.e. success or suitable error code.

Note

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

5.17.1.3.2.9 `virtual telux::common::Status telux::tel::ISimProfileManager::provideUserConsent ( SlotId slotId, bool userConsent = false, common::ResponseCallback callback = nullptr ) [pure virtual]`

Provide user consent required for downloading and installing profile.

Parameters

in	<i>slotId</i>	Slot identifier corresponding to the card.
in	<i>userConsent</i>	Consent for profile download and install. True means user consent to download and install.
in	<i>callback</i>	Callback function to get the result of user consent request.

Returns

Status of user consent request i.e. success or suitable error code.

Note

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

5.17.1.3.2.10 `virtual telux::common::Status telux::tel::ISimProfileManager::registerListener ( std::weak_ptr< ISimProfileListener > listener ) [pure virtual]`

Register a listener to listen for status of specific events like download and installation of profile on eUICC.

Parameters

in	<i>listener</i>	Pointer of ISimProfileListener object that processes the notification.

Returns

Status of registerListener success or suitable status code

Note

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

5.17.1.3.2.11 `virtual telux::common::Status telux::tel::ISimProfileManager::deregisterListener ( std::weak_ptr< ISimProfileListener > listener ) [pure virtual]`

De-register the listener.

Parameters

<code>in</code>	<code>listener</code>	Pointer of ISimProfileListener object that needs to be removed

Returns

Status of deregisterListener success or suitable status code

Note

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

5.17.2 Enumeration Type Documentation

5.17.2.1 `enum telux::tel::ProfileType`

Indicates profile type of card

Enumerator

UNKNOWN

REGULAR Regular profile

EMERGENCY Emergency profile

5.17.2.2 `enum telux::tel::IconType`

Indicates profile icon type.

Enumerator

NONE No icon information

JPEG JPEG icon

PNG PNG icon

5.17.2.3 `enum telux::tel::ProfileClass`

Indicates profile class.

Enumerator

UNKNOWN No info about profile class

TEST Test profile

PROVISIONING Provisioning profile

OPERATIONAL Operational profile

5.17.2.4 `enum telux::tel::DownloadStatus`

Indicates profile download status.

Enumerator

DOWNLOAD_ERROR Profile download error
DOWNLOAD_IN_PROGRESS Profile download in progress with download percentage
DOWNLOAD_COMPLETE_INSTALLATION_IN_PROGRESS Profile download is complete and installation is in progress
INSTALLATION_COMPLETE Profile installation is complete
USER_CONSENT_REQUIRED User consent is required for proceeding with download/installation of profile

5.17.2.5 enum telux::tel::DownloadErrorCause

Indicates profile download error cause.

Enumerator

GENERIC Generic error
SIM Error from the SIM card
NETWORK Error from the network
MEMORY Error due to no memory

5.17.2.6 enum telux::tel::PolicyRuleType

Defines profile policy rules(PPR). Each value represents corresponding bit for PprMask bitset.

Enumerator

PROFILE_DISABLE_NOT_ALLOWED Disabling of the profile is not allowed
PROFILE_DELETE_NOT_ALLOWED Deletion of the profile is not allowed
PROFILE_DELETE_ON_DISABLE Deletion of the profile is required on successful disabling

5.18 Remote SIM

This section contains APIs related to Remote SIM operations.

5.18.1 Data Structure Documentation

5.18.1.1 class telux::tel::IRemoteSimListener

A listener class for getting remote SIM notifications.

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

Public member functions

- virtual void [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 [~IRemoteSimListener](#) ()

5.18.1.1.1 Constructors and Destructors

5.18.1.1.1.1 virtual telux::tel::IRemoteSimListener::~~IRemoteSimListener () [virtual]

Destructor of [IRemoteSimListener](#)

5.18.1.1.2 Member Function Documentation

5.18.1.1.2.1 virtual void telux::tel::IRemoteSimListener::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)

5.18.1.1.2.2 virtual void telux::tel::IRemoteSimListener::onCardConnect () [virtual]

This function is called when the modem wants to establish a connection.

5.18.1.1.2.3 virtual void telux::tel::IRemoteSimListener::onCardDisconnect () [virtual]

This function is called when the modem wants to tear down a connection.

5.18.1.1.2.4 virtual void telux::tel::IRemoteSimListener::onCardPowerUp () [virtual]

This function is called when the modem wants to power up the card.

5.18.1.1.2.5 virtual void telux::tel::IRemoteSimListener::onCardPowerDown () [virtual]

This function is called when the modem wants to power down the card.

5.18.1.1.2.6 virtual void telux::tel::IRemoteSimListener::onCardReset () [virtual]

This function is called when the modem wants to warm reset the card.

5.18.1.2 class telux::tel::IRemoteSimManager

[IRemoteSimManager](#) provides APIs for remote SIM related operations. This allows a device to use a SIM card on another device for its WWAN modem functionality. The SIM provider service is the endpoint that interfaces with the SIM card (e.g. over bluetooth) and sends/receives data to the other endpoint, the modem. The modem sends requests to the SIM provider service to interact with the SIM card (e.g. power up, transmit APDU, etc.), and is notified of events (e.g. card errors, resets, etc.). This API is used by the SIM provider endpoint to provide a SIM card to the modem.

Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status](#) [sendReset](#) ([telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [sendConnectionAvailable](#) ([telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [sendConnectionUnavailable](#) ([telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [sendCardReset](#) (const std::vector< uint8_t > &atr, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [sendCardError](#) (const [CardErrorCause](#) cause=CardErrorCause::INVALID, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [sendCardInserted](#) (const std::vector< uint8_t > &atr, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [sendCardRemoved](#) ([telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [sendCardWakeup](#) ([telux::common::ResponseCallback](#) callback=nullptr)=0

- virtual [telux::common::Status sendApdu](#) (const unsigned int id, const std::vector< uint8_t > &apdu, const bool isSuccess=true, const unsigned int totalSize=0, const unsigned int offset=0, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status registerListener](#) (std::weak_ptr< [IRemoteSimListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak_ptr< [IRemoteSimListener](#) > listener)=0
- virtual int [getSlotId](#) ()=0
- virtual [~IRemoteSimManager](#) ()

5.18.1.2.1 Constructors and Destructors

5.18.1.2.1.1 virtual [telux::tel::IRemoteSimManager::~~IRemoteSimManager](#) () [**virtual**]

Destructor of [IRemoteSimManager](#)

5.18.1.2.2 Member Function Documentation

5.18.1.2.2.1 virtual bool [telux::tel::IRemoteSimManager::isSubsystemReady](#) () [**pure virtual**]

Checks the status of remote SIM subsystem and returns the result.

Returns

True if remote SIM subsystem is ready for service otherwise false.

5.18.1.2.2.2 virtual std::future<bool> [telux::tel::IRemoteSimManager::onSubsystemReady](#) () [**pure virtual**]

Wait for remote SIM subsystem to be ready.

Returns

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

5.18.1.2.2.3 virtual [telux::common::Status telux::tel::IRemoteSimManager::sendReset](#) ([telux::common::ResponseCallback](#) *callback* = *nullptr*) [**pure virtual**]

Send reset command to the modem to reset state variables.

Parameters

out	<i>callback</i>	Callback function pointer to get the response of sendReset.

Returns

Status of sendReset i.e. success or suitable status code.

5.18.1.2.2.4 **virtual telux::common::Status telux::tel::IRemoteSimManager::sendConnectionAvailable (telux::common::ResponseCallback *callback* = nullptr) [pure virtual]**

Send connection available event to the modem.

Parameters

out	<i>callback</i>	Callback function pointer to get the response.

Returns

Status of sendConnectionAvailable i.e. success or suitable status code.

5.18.1.2.2.5 **virtual telux::common::Status telux::tel::IRemoteSimManager::sendConnectionUnavailable (telux::common::ResponseCallback *callback* = nullptr) [pure virtual]**

Send connection unavailable event to the modem.

Parameters

out	<i>callback</i>	Callback function pointer to get the response.

Returns

Status of sendConnectionUnavailable i.e. success or suitable status code.

5.18.1.2.2.6 **virtual telux::common::Status telux::tel::IRemoteSimManager::sendCardReset (const std::vector< uint8_t > & *atr*, telux::common::ResponseCallback *callback* = nullptr) [pure virtual]**

Send card reset event to the modem.

Parameters

in	<i>atr</i>	Answer to Reset bytes (max size = 32, per ISO/IEC 7816-3:2006 section 8.1).
out	<i>callback</i>	Callback function pointer to get the response of sendCardReset.

Returns

Status of sendCardReset i.e. success or suitable status code.

5.18.1.2.2.7 **virtual telux::common::Status telux::tel::IRemoteSimManager::sendCardError (const Card-ErrorCause *cause* = CardErrorCause::INVALID, telux::common::ResponseCallback *callback* = nullptr) [pure virtual]**

Send card error event to the modem.

in	<i>cause</i>	Card Error cause.
out	<i>callback</i>	Callback function pointer to get the response of sendCardError.

Returns

Status of sendCardError i.e. success or suitable status code.

5.18.1.2.2.8 `virtual telux::common::Status telux::tel::IRemoteSimManager::sendCardInserted ( const std::vector< uint8_t > & atr, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Send card inserted event to the modem.

Parameters

in	<i>atr</i>	Answer to Reset bytes (max size = 32, per ISO/IEC 7816-3:2006 section 8.1).
out	<i>callback</i>	Callback function pointer to get the response of sendCardInserted.

Returns

Status of sendCardInserted i.e. success or suitable status code.

5.18.1.2.2.9 `virtual telux::common::Status telux::tel::IRemoteSimManager::sendCardRemoved ( telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Send card removed event to the modem.

Parameters

out	<i>callback</i>	Callback function pointer to get the response of sendCardRemoved.

Returns

Status of sendCardRemoved i.e. success or suitable status code.

5.18.1.2.2.10 `virtual telux::common::Status telux::tel::IRemoteSimManager::sendCardWakeup ( telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Send card wakeup event to the modem.

Parameters

out	<i>callback</i>	Callback function pointer to get the response of sendCardWakeup.

Returns

Status of sendCardWakeup i.e. success or suitable status code.

5.18.1.2.2.11 `virtual telux::common::Status telux::tel::IRemoteSimManager::sendApdu ( const unsigned int id, const std::vector< uint8_t > & apdu, const bool isSuccess = true, const unsigned int totalSize = 0, const unsigned int offset = 0, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Sends an APDU message to the modem, in response to a previous APDU sent by the modem.

Parameters

in	<i>id</i>	Identifier for command and response APDU pair.
in	<i>apdu</i>	Response APDU (max size = 1024).
in	<i>isSuccess</i>	Whether APDU transaction completed successfully.
in	<i>totalSize</i>	Total length of the APDU message (used when the response is larger than 1024 bytes and must be passed in multiple segments).
in	<i>offset</i>	Offset of this APDU segment in the original message.
out	<i>callback</i>	Callback function pointer to get the response of sendApdu.

Returns

Status of sendApdu i.e. success or suitable status code.

5.18.1.2.2.12 `virtual telux::common::Status telux::tel::IRemoteSimManager::registerListener ( std::weak_ptr< IRemoteSimListener > listener ) [pure virtual]`

Register a listener for specific updates from the modem.

Parameters

in	<i>listener</i>	Pointer of IRemoteSimListener object that processes the notification

Returns

Status of registerListener i.e success or suitable status code.

5.18.1.2.2.13 `virtual telux::common::Status telux::tel::IRemoteSimManager::deregisterListener ( std::weak_ptr< IRemoteSimListener > listener ) [pure virtual]`

Deregister the previously added listener.

Parameters

in	<i>listener</i>	Previously registered IRemoteSimListener that needs to be deregistered

Returns

Status of deregisterListener success or suitable status code

5.18.1.2.2.14 virtual int telux::tel::IRemoteSimManager::getSlotId () [pure virtual]

Get associated slot ID for the RemoteSimManager

Returns

The slot ID associated with this [IRemoteSimManager](#)

5.18.2 Enumeration Type Documentation

5.18.2.1 enum telux::tel::CardErrorCause

Defines the card error cause, sent to the modem by the SIM provider

Enumerator

INVALID Card error cause value will not be passed to modem

UNKNOWN_ERROR Unknown error

NO_LINK_ESTABLISHED No link was established

COMMAND_TIMEOUT Command timeout

POWER_DOWN Error due to a card power down

5.19 Modem Config

This section contains APIs related to Modem Config operations.

5.19.1 Data Structure Documentation

5.19.1.1 class telux::config::ConfigFactory

[ConfigFactory](#) allows creation of config related classes.

Public member functions

- `std::shared_ptr`
`< IModemConfigManager > getModemConfigManager ()`

Static Public Member Functions

- static [ConfigFactory](#) & `getInstance ()`

5.19.1.1.1 Member Function Documentation

5.19.1.1.1.1 static [ConfigFactory](#)& telux::config::ConfigFactory::getInstance () [static]

Get instance of Config Factory

5.19.1.1.1.2 `std::shared_ptr<IModemConfigManager> telux::config::ConfigFactory::getModemConfigManager ( )`

Get instance of ModemConfig manager

Returns

pointer of [IModemConfigManager](#) object.

5.19.1.2 struct telux::config::ConfigInfo

Data fields

Type	Field	Description
ConfigId	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
ConfigType	type	
uint32_t	size	
string	desc	
uint32_t	version	

5.19.1.3 class telux::config::IModemConfigListener

Listener class for getting notifications related to configuration change detection. The client needs to implement these methods as briefly as possible and avoid blocking calls in it. The methods in this class can be invoked from multiple different threads. Client needs to make sure that the implementation is thread-safe.

Public member functions

- virtual void [onConfigUpdateStatus](#) ([ConfigUpdateStatus](#) status, int slotId)
- virtual [~IModemConfigListener](#) ()

5.19.1.3.1 Constructors and Destructors

5.19.1.3.1.1 virtual telux::config::IModemConfigListener::~IModemConfigListener () [virtual]

Destructor of [IModemConfigListener](#)

5.19.1.3.2 Member Function Documentation

5.19.1.3.2.1 virtual void telux::config::IModemConfigListener::onConfigUpdateStatus ([ConfigUpdateStatus](#) status, int slotId) [virtual]

This function is called when a configuration update is detected. It is applicable only to SOFTWARE config.

Parameters

in	status	update status of config.
in	slotId	slotId where update is detected.

5.19.1.4 class telux::config::IModemConfigManager

[IModemConfigManager](#) provides interface to list config files present in modem's storage. load a new config file in modem, activate a config file, get active config file information, deactivate a config file, delete config file from the modem's storage, get and set mode of config auto selection, register and deregister listener for config update in modem. The config files are also referred to as MBNs.

Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status](#) requestConfigList ([ConfigListCallback](#) cb)=0
- virtual [telux::common::Status](#) loadConfigFile (std::string filePath, [ConfigType](#) configType, [telux::common::ResponseCallback](#) cb=nullptr)=0
- virtual [telux::common::Status](#) activateConfig ([ConfigType](#) configType, [ConfigId](#) configId, int slotId=DEFAULT_SLOT_ID, [telux::common::ResponseCallback](#) cb=nullptr)=0

- virtual [telux::common::Status](#) [getActiveConfig](#) ([ConfigType](#) configType, [GetActiveConfigCallback](#) cb, int slotId=DEFAULT_SLOT_ID)=0
- virtual [telux::common::Status](#) [deactivateConfig](#) ([ConfigType](#) configType, int slotId=DEFAULT_SLOT_ID, [telux::common::ResponseCallback](#) cb=nullptr)=0
- virtual [telux::common::Status](#) [deleteConfig](#) ([ConfigType](#) configType, [ConfigId](#) configId="", [telux::common::ResponseCallback](#) cb=nullptr)=0
- virtual [telux::common::Status](#) [getAutoSelectionMode](#) ([GetAutoSelectionModeCallback](#) cb, int slotId=DEFAULT_SLOT_ID)=0
- virtual [telux::common::Status](#) [setAutoSelectionMode](#) ([AutoSelectionMode](#) mode, int slotId=DEFAULT_SLOT_ID, [telux::common::ResponseCallback](#) cb=nullptr)=0
- virtual [telux::common::Status](#) [registerListener](#) (std::weak_ptr< [IModemConfigListener](#) > listener)=0
- virtual [telux::common::Status](#) [deregisterListener](#) (std::weak_ptr< [IModemConfigListener](#) > listener)=0
- virtual [~IModemConfigManager](#) ()

5.19.1.4.1 Constructors and Destructors

5.19.1.4.1.1 virtual [telux::config::IModemConfigManager::~IModemConfigManager](#) () [**virtual**]

Destructor for [IModemConfigManager](#)

5.19.1.4.2 Member Function Documentation

5.19.1.4.2.1 virtual bool [telux::config::IModemConfigManager::isSubsystemReady](#) () [**pure virtual**]

Checks the status of modem config subsystem and returns the result.

Returns

If true that means ModemConfigManager is ready for performing config operations.

5.19.1.4.2.2 virtual std::future<bool> [telux::config::IModemConfigManager::onSubsystemReady](#) () [**pure virtual**]

Wait for modem config subsystem to be ready.

Returns

A future that caller can wait on to be notified when modem config subsystem is ready.

5.19.1.4.2.3 virtual [telux::common::Status](#) [telux::config::IModemConfigManager::requestConfigList](#) ([ConfigListCallback](#) cb) [**pure virtual**]

Fetching the list of config files present in modem's storage.

in	<i>cb</i>	- callback to the Response function.

returns SUCCESS if the request to get config list is sent successfully.

5.19.1.4.2.4 virtual telux::common::Status telux::config::IModemConfigManager::loadConfigFile (std::string *filePath*, ConfigType *configType*, telux::common::ResponseCallback *cb* = nullptr) [pure virtual]

Loads a new config file into the modem's storage. This is a persistent operation. Only the config files loaded into the modem's storage can be activated.

Parameters

in	<i>filePath</i>	- it defines the path to the config file.
in	<i>configType</i>	- type of the config file.
in	<i>cb</i>	- callback to the response function.

returns SUCCESS if the request to load config file is sent successfully.

5.19.1.4.2.5 virtual telux::common::Status telux::config::IModemConfigManager::activateConfig (ConfigType *configType*, ConfigId *configId*, int *slotId* = DEFAULT_SLOT_ID, telux::common::ResponseCallback *cb* = nullptr) [pure virtual]

Activates the config file on specified slot id. A file for activation must be loaded or should already be present in modem's storage.

Parameters

in	<i>configType</i>	- type of the config file.
in	<i>configId</i>	- id of the config file.
in	<i>slotId</i>	- it defines the slot id to be selected.
in	<i>cb</i>	- callback to the response function.

Returns

SUCCESS if the request to activate config file is sent successfully.

5.19.1.4.2.6 virtual telux::common::Status telux::config::IModemConfigManager::getActiveConfig (ConfigType *configType*, GetActiveConfigCallback *cb*, int *slotId* = DEFAULT_SLOT_ID) [pure virtual]

Get the currently active config file information for the specified slot id. In case default config files are activated, would return error.

Parameters

in	<i>configType</i>	- type of the config file.
in	<i>cb</i>	- callback to the response function.

in	<i>slotId</i>	- it defines the slot id to be selected.

Returns

SUCCESS if the request to get active config information is sent successfully.

5.19.1.4.2.7 `virtual telux::common::Status telux::config::IModemConfigManager::deactivateConfig ( ConfigType configType, int slotId = DEFAULT_SLOT_ID, telux::common::ResponseCallback cb = nullptr ) [pure virtual]`

Deactivates the config file for the specified slot id.

Parameters

in	<i>configType</i>	- type of the config file.
in	<i>slotId</i>	- slot id to be selected for deactivation of config.
in	<i>cb</i>	- callback to the response function.

Returns

SUCCESS if the request to deactivate config file is sent successfully

5.19.1.4.2.8 `virtual telux::common::Status telux::config::IModemConfigManager::deleteConfig ( ConfigType configType, ConfigId configId = "", telux::common::ResponseCallback cb = nullptr ) [pure virtual]`

Deletes the config file from the modem's storage.

Parameters

in	<i>configType</i>	- type of the config file.
in	<i>configId</i>	- id of the config file. This parameter is optional if not provided all the config files of the given config type are deleted from modem's storage.
in	<i>cb</i>	- callback to the Response function.

Returns

SUCCESS if the request to delete config file is sent successfully

5.19.1.4.2.9 `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.

in	<i>cb</i>	- callback to the response function.
in	<i>slotId</i>	- slot id of config.

Returns

SUCCESS if the request to get selection mode is sent successfully

5.19.1.4.2.10 `virtual telux::common::Status telux::config::IModemConfigManager::setAutoSelectionMode ( AutoSelectionMode mode, int slotId = DEFAULT_SLOT_ID, telux::common::ResponseCallback cb = nullptr ) [pure virtual]`

Setting the mode of config auto selection for specified slot id.

Parameters

in	<i>mode</i>	- auto selection mode status.
in	<i>slotId</i>	- slot id of the config.
in	<i>cb</i>	- callback to the response function.

Returns

SUCCESS if the request to set selection mode is sent successfully.

5.19.1.4.2.11 `virtual telux::common::Status telux::config::IModemConfigManager::registerListener ( std::weak_ptr< IModemConfigListener > listener ) [pure virtual]`

Registers the listener for indications.

Parameters

in	<i>listener</i>	- pointer to implemented listener.

Returns

SUCCESS if the request to register listener is sent successfully.

5.19.1.4.2.12 `virtual telux::common::Status telux::config::IModemConfigManager::deregisterListener ( std::weak_ptr< IModemConfigListener > listener ) [pure virtual]`

Deregisters the listener from indications.

Parameters

in	<i>listener</i>	- pointer to registered listener.

Returns

SUCCESS if the request to deregister listener is sent successfully.

5.19.2 Enumeration Type Documentation

5.19.2.1 enum telux::config::ConfigType

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

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

ConfigUpdateStatus represent status of config update, a update of config happens when a software config is activated and all segments using the config are updated with new config.

Enumerator

START start of updation process

COMPLETE end of updation process

5.20 Wlan

- [Wlan Device Management](#)
- [Access Point Management](#)
- [Station Management](#)

This section contains APIs related to Wlan management operations.

5.20.1 Define Documentation

5.20.1.1 #define INVALID_AP_ID 0

5.20.2 Data Structure Documentation

5.20.2.1 struct telux::wlan::ApNetConfig

Ap Network Configuration

Data fields

Type	Field	Description
ApInfo	info	AP type
ApInterworking	interworking	AP network access (internet/local)

5.20.2.2 struct telux::wlan::ApConfig

Ap Configuration

Data fields

Type	Field	Description
Id	id	AP id
vector< Ap-NetConfig >	network	Configurations supported by AP

5.20.2.3 struct telux::wlan::DeviceIndInfo

Wlan Client Device Indication Info

Data fields

Type	Field	Description
Id	id	AP id device is connected to
string	macAddress	MAC Address of Wi-Fi device

5.20.2.4 struct telux::wlan::DeviceInfo

Wlan Client Device Info

Data fields

Type	Field	Description
Id	id	AP id device is connected to
string	name	User friendly string that identifies Wi-Fi device
string	ipv4Address	IPv4 Address of Wi-Fi device
string	ipv6Address	IPv6 Address of Wi-Fi device
string	macAddress	MAC Address of Wi-Fi device

5.20.2.5 struct telux::wlan::DeviceStats

Wlan Client Device Statistics

Data fields

Type	Field	Description
Id	id	AP id device is connected to
string	macAddress	MAC Address of Wi-Fi device
uint64_t	bytesTx	Number of bytes transmitted
uint64_t	bytesRx	Number of bytes received

5.20.2.6 class telux::wlan::IStaInterfaceManager

Manager class for configuring Wlan Station Mode.

Public member functions

- virtual [telux::common::ErrorCode](#) setIpConfig ([Id](#) staId, [StaIpConfig](#) ipConfig, [StaStaticIpConfig](#) staticIpConfig)=0
- virtual [telux::common::ErrorCode](#) setBridgeMode ([Id](#) staId, [StaBridgeMode](#) bridgeMode)=0
- virtual [telux::common::ErrorCode](#) getConfig (std::vector< [StaConfig](#) > &config)=0
- virtual [telux::common::ErrorCode](#) getStatus (std::vector< [StaStatus](#) > &status)=0
- virtual [telux::common::ErrorCode](#) manageStaService ([Id](#) staId, [ServiceOperation](#) opr)=0
- virtual [telux::common::ErrorCode](#) registerListener (std::weak_ptr< [IStaListener](#) > listener)=0
- virtual [telux::common::ErrorCode](#) deregisterListener (std::weak_ptr< [IStaListener](#) > listener)=0
- virtual [~IStaInterfaceManager](#) ()

5.20.2.7 class telux::wlan::IStaListener

Public member functions

- virtual void [onStationStatusChanged](#) (std::vector< [StaStatus](#) > staStatus)
- virtual void [onStationBandChanged](#) ([BandType](#) radio)
- virtual [~IStaListener](#) ()

5.20.2.8 struct telux::wlan::ApInfo

AP Info - captures ap type (private/guest)

Data fields

Type	Field	Description
ApType	apType	Ap type (private/guest)

5.20.2.9 struct telux::wlan::ApNetInfo

Ap Network Info

Data fields

Type	Field	Description
ApInfo	info	Ap information (AP type)

5.20.2.10 struct telux::wlan::ApStatus

AP Status for enabled Networks

Data fields

Type	Field	Description
Id	id	AP id
string	name	AP network interface name
string	ipv4Address	Local AP IP V4 address
string	macAddress	AP MAC address
vector< Ap- NetInfo >	network	Settings for AP info

5.20.2.11 struct telux::wlan::StaStatus

Station Status

Data fields

Type	Field	Description
Id	id	Station Id
string	name	Network interface name
string	ipv4Address	Public IP V4 address

Type	Field	Description
string	ipv6Address	Public IP V6 address
string	macAddress	MAC address
StaInterface-Status	status	Interface status

5.20.2.12 struct telux::wlan::InterfaceStatus

Wlan Interface status

Data fields

Type	Field	Description
HwDeviceType	device	
vector< Ap-Status >	apStatus	WiFi hardware type Vector of active APs status
vector< Sta-Status >	staStatus	Vector of active Sta status

5.20.2.13 class telux::wlan::IWlanDeviceManager

WlanDeviceManager is a primary interface for configuring Wireless LAN. it provide APIs to enable, configure, activate, and modify modes.

Public member functions

- virtual [telux::common::ServiceStatus](#) getServiceStatus ()=0
- virtual [telux::common::ErrorCode](#) enable (bool enable)=0
- virtual [telux::common::ErrorCode](#) setMode (int numOfAp, int numOfSta)=0
- virtual [telux::common::ErrorCode](#) getConfig (int &numAp, int &numSta)=0
- virtual [telux::common::ErrorCode](#) getStatus (bool &isEnabled, std::vector< [InterfaceStatus](#) > &status)=0
- virtual [telux::common::ErrorCode](#) registerListener (std::weak_ptr< [IWlanListener](#) > listener)=0
- virtual [telux::common::ErrorCode](#) deregisterListener (std::weak_ptr< [IWlanListener](#) > listener)=0
- virtual ~[IWlanDeviceManager](#) ()

5.20.2.14 class telux::wlan::IWlanListener

Public member functions

- virtual void onServiceStatusChange ([telux::common::ServiceStatus](#) status)
- virtual void onEnableChanged (bool enable)

- virtual [~IWlanListener](#) ()

5.20.2.15 class telux::wlan::WlanFactory

[WlanFactory](#) is the central factory to create all wlan classes.

Public member functions

- virtual std::shared_ptr
< [IWlanDeviceManager](#) > [getWlanDeviceManager](#) ([telux::common::InitResponseCb](#)
clientCallback=nullptr)=0
- virtual std::shared_ptr
< [IApInterfaceManager](#) > [getApInterfaceManager](#) ()=0
- virtual std::shared_ptr
< [IStaInterfaceManager](#) > [getStaInterfaceManager](#) ()=0

Static Public Member Functions

- static [WlanFactory](#) & [getInstance](#) ()

Protected Member Functions

- [WlanFactory](#) ()
- virtual [~WlanFactory](#) ()

5.20.2.15.1 Constructors and Destructors

5.20.2.15.1.1 [telux::wlan::WlanFactory::WlanFactory](#) () [[protected](#)]

5.20.2.15.1.2 virtual [telux::wlan::WlanFactory::~~WlanFactory](#) () [[protected](#)], [[virtual](#)]

5.20.2.15.2 Member Function Documentation

5.20.2.15.2.1 static [WlanFactory&](#) [telux::wlan::WlanFactory::getInstance](#) () [[static](#)]

Get Wlan Factory instance.

5.20.2.15.2.2 virtual std::shared_ptr<[IWlanDeviceManager](#)> [telux::wlan::WlanFactory::getWlanDeviceManager](#) ([telux::common::InitResponseCb](#) *clientCallback* = nullptr) [[pure virtual](#)]

Get Wlan Device Manager

Parameters

in	<i>clientCallback</i>	Optional callback to get the initialization status of WlanDeviceManager telux::common::InitResponseCb

Returns

instance of [IWlanDeviceManager](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.20.2.15.2.3 `virtual std::shared_ptr<IApInterfaceManager> telux::wlan::WlanFactory::getApInterfaceManager( ) [pure virtual]`

Get Access Point Interface Manager

Returns

instance of [IApInterfaceManager](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.20.2.15.2.4 `virtual std::shared_ptr<IStaInterfaceManager> telux::wlan::WlanFactory::getStaInterfaceManager( ) [pure virtual]`

Get Station Interface Manager

Returns

instance of [IStaInterfaceManager](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.20.3 Enumeration Type Documentation

5.20.3.1 enum telux::wlan::ApInterworking

AP Interworking Information

Enumerator

INTERNET_ACCESS AP with internet access only - No LAN access

FULL_ACCESS AP Can Access LAN and Internet

5.20.3.2 enum telux::wlan::ApDeviceConnectionEvent

AP Client Connection Status

Enumerator

CONNECTED
DISCONNECTED
IPV4_UPDATED
IPV6_UPDATED

5.20.3.3 enum telux::wlan::BandType

Radio Band Types:

Enumerator

BAND_5GHZ
BAND_2GHZ

5.20.3.4 enum telux::wlan::ConnectionStatus

Connection Status

Enumerator

UNKNOWN Device connection is unknown
CONNECTED Device is connected
DISCONNECTED Device is disconnected

5.20.3.5 enum telux::wlan::Id

Identifiers for Ap, Sta, P2p

Enumerator

PRIMARY
SECONDARY
TERTIARY
QUATERNARY

5.20.3.6 enum telux::wlan::ApType

AP Types:

Enumerator

UNKNOWN
PRIVATE
GUEST

5.20.3.7 enum telux::wlan::StaInterfaceStatus

Station Interface Status

Enumerator

UNKNOWN Station interface is unknown
CONNECTING Station interface is connecting
CONNECTED Station interface is connected
DISCONNECTED Station interface is disconnected
ASSOCIATION_FAILED Station is unable to associate with AP
IP_ASSIGNMENT_FAILED Station is unable to get IP address via DHCP

5.20.3.8 enum telux::wlan::OperationType

This applies in architectures where the modem is attached to an External Application Processor(EAP). An API that sets or configure Wlan can be invoked from the EAP or from the modems Internal Application Processor (IAP). This type specifies where the operation should be carried out.

Enumerator

WLAN_LOCAL Perform the operation on the processor where the API is invoked.
WLAN_REMOTE Perform the operation on the application processor other than where the API is invoked.

5.20.3.9 enum telux::wlan::IpFamilyType

Preferred IP family for the connection

Enumerator

UNKNOWN
IPV4 IPv4 data connection
IPV6 IPv6 data connection
IPV4V6 IPv4 and IPv6 data connection

5.20.3.10 enum telux::wlan::ServiceOperation

Service operations to be performed

Enumerator

STOP Stop service
START Start service
RESTART Restart service

5.20.3.11 enum telux::wlan::InterfaceState

Wlan Interface State

Enumerator**INACTIVE** Interface is Inactive**ACTIVE** Interface is Active**5.20.3.12 enum telux::wlan::HwDeviceType**

Wlan Interface Device

Enumerator**UNKNOWN** Wlan device is Unknown**QCA6574** Wlan device is QCA6574**QCA6696** Wlan device is QCA6696**QCA6595** Wlan device is QCA6595**5.20.4 Function Documentation****5.20.4.1 virtual telux::common::ErrorCode telux::wlan::IApInterfaceManager::set-Config (ApConfig *config*) [pure virtual]**

Set Access Point config: Used to fully configure access points including venue type, radio type (2.4/5 GHz), private/guest network and all other related settings. Configurations will take effect after hostapd service is restarted by calling [telux::wlan::IApInterfaceManager::manageApService](#).

Parameters

in	<i>config</i>	AP configuration parameters telux::wlan::ApConfig

Returns

operation error code (if any). [telux::common::ErrorCode](#) telux::common::Status::NOTALLOWED is returned if AP to be configured was not enabled in telux::wlan::WlanDeviceManager::setMode.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.20.4.2 virtual telux::common::ErrorCode telux::wlan::IApInterfaceManager::get-Config (std::vector< ApConfig > & *config*) [pure virtual]

Request Access Point Configurations

Parameters

in	<i>config</i>	Vector of AP configurations telux::wlan::ApConfig as set by telux::wlan::IApInterfaceManager::setConfig

Returns

operation error code (if any). [telux::common::ErrorCode](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.20.4.3 virtual telux::common::ErrorCode telux::wlan::IApInterfaceManager::getStatus (std::vector< ApStatus > & status) [pure virtual]

Request AP Status

Parameters

in	<i>status</i>	Vector of AP network Status telux::wlan::ApStatus

Returns

operation error code (if any). [telux::common::ErrorCode](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.20.4.4 virtual telux::common::ErrorCode telux::wlan::IApInterfaceManager::getConnectedDevices (std::vector< DeviceInfo > & clientsInfo) [pure virtual]

Request Connected Devices to all enabled access points. Each entry in returned list will contain information about a device such as access point it is connected to and IP and MAC address as defined in [telux::wlan::DeviceInfo](#)

Parameters

in	<i>clientsInfo</i>	List of connected devices Info telux::wlan::DeviceInfo

Returns

operation error code (if any). [telux::common::ErrorCode](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.20.4.5 **virtual telux::common::ErrorCode telux::wlan::IApInterfaceManager::manageApService (Id *apId*, ServiceOperation *opr*) [pure virtual]**

Execute an operation on hostapd service. Provides ability for client to either stop/start or restart hostapd service for selected access point. Restarting hostapd service is required for any changes made to hosapd.conf file and changes made by [telux::wlan::IApInterfaceManager::setConfig](#) to take effect. Stop/Start operation [telux::wlan::ServiceOperation](#) will Stop/Start WiFi service for access point. Access points selected to execute operation on, will temporarily go out of service when this API is called. This API should be called only when access point is configured through [telux::wlan::IDeviceManager::setMode](#)

Parameters

in	<i>apId</i>	AP identifier to execute operation on. telux::wlan::Id
in	<i>opr</i>	Operation to be performed on hostapd telux::wlan::ServiceOperation

Returns

operation error code (if any). [telux::common::ErrorCode](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.20.4.6 **virtual telux::common::ErrorCode telux::wlan::IApInterfaceManager::registerListener (std::weak_ptr< IApListener > *listener*) [pure virtual]**

Register a listener for specific events in Access Point Manager

Parameters

in	<i>listener</i>	pointer of IApListener object that processes the notification

Returns

operation error code (if any). [telux::common::ErrorCode](#)

5.20.4.7 **virtual telux::common::ErrorCode telux::wlan::IApInterfaceManager::deregisterListener (std::weak_ptr< IApListener > *listener*) [pure virtual]**

Removes a previously added listener.

Parameters

in	<i>listener</i>	pointer of IApListener object that needs to be removed

Returns

operation error code (if any). [telux::common::ErrorCode](#)

5.20.4.8 virtual telux::wlan::IApInterfaceManager::~~IApInterfaceManager ()
[virtual]

**5.20.4.9 virtual telux::common::ErrorCode telux::wlan::IApInterfaceManager::get-
 ConnectedDevicesStats (std::vector< DeviceStats > & *clientsStats*)**
[pure virtual]

Deprecated APIs Request statistics for all devices connected to all access points. Each entry in returned list will contains transmitted and recieved bytes for a device as defined in telux::common::DeviceStats

Parameters

in	<i>clientStats</i>	List of connected clients statistics telux::wlan::DeviceStats

Returns

operation error code (if any). [telux::common::ErrorCode](#)

Deprecated This API is no longer supported.

5.20.4.10 virtual void telux::wlan::IApListener::onApDeviceStatusChanged (
ApDeviceConnectionEvent *event*, std::vector< DeviceIndInfo > *info*)
[virtual]

This function is called when AP device status has changed

Parameters

in	<i>event</i>	Event detected on device telux::wlan::ApDeviceConnectionEvent
in	<i>info</i>	Info about devices telux::wlan::DeviceIndInfo

5.20.4.11 virtual void telux::wlan::IApListener::onApBandChanged (BandType *radio*
) [virtual]

This function is called when AP switch to different operation band

Parameters

in	<i>radio</i>	New AP operation band telux::wlan::BandType

5.20.4.12 `virtual telux::wlan::IApListener::~~IApListener ( ) [virtual]`

5.20.4.13 `virtual void telux::wlan::IApListener::onApDeviceStatusChanged ( ApDeviceConnectionEvent event, std::vector< DeviceInfo > info ) [virtual]`

Deprecated APIs This function is called when AP device status has changed

Parameters

in	<i>event</i>	Event detected on device telux::wlan::ApDeviceConnectionEvent
in	<i>info</i>	Info about devices telux::wlan::DeviceInfo

Note

The name, ipv4Address, and ipv6Address fields from [telux::wlan::DeviceInfo](#) are not populated in the indication data. please invoke [telux::wlan::IApInterfaceManager::getConnectedDevices](#) for these details.

5.20.4.14 `virtual telux::common::ServiceStatus telux::wlan::IWlanDeviceManager::getServiceStatus ( ) [pure virtual]`

Checks the readiness status of wlan manager and returns the result.

Returns

SERVICE_AVAILABLE - If wlan manager is ready for service. SERVICE_UNAVAILABLE - If wlan manager is temporarily unavailable. SERVICE_FAILED - If wlan manager encountered an irrecoverable failure.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.20.4.15 `virtual telux::common::ErrorCode telux::wlan::IWlanDeviceManager::enable ( bool enable ) [pure virtual]`

Enable or Disable Wlan Service. Configurations set by [telux::wlan::IWlanDeviceManager::setMode](#) must be completed before enabling Wlan. If any of configurations need to be changed after Wlan is enabled, this API must be called with enable set to false followed by a call with enable set to true for the new configurations to take effect. Calling this API with enable, will start hostapd and wpa_supplicant daemons. Further changes to hostapd and wpa_supplicant will require calling [telux::wlan::IApInterfaceManager::manageApService](#) and [telux::wlan::IStaInterfaceManager::manageStaService](#) respectively. Client shall wait for [IWlanListener::onEnableChanged](#) indication to confirm WLAN was enabled/disabled successfully

in	<i>enable</i>	true : Enable Wlan, false: Disable Wlan.

Returns

operation error code (if any). [telux::common::ErrorCode](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.20.4.16 **virtual telux::common::ErrorCode telux::wlan::IWlanDeviceManager::set-Mode (int *numOfAp*, int *numOfSta*) [pure virtual]**

Set Wlan mode - number of supported APs, and stations. This API shall be called when wlan is disabled. On enablement, wlan will enable APs and Stations set in this API.

Parameters

in	<i>numOfAp</i>	Num of Access Points to be enabled. If no Access Point is enabled, this argument should be set to 0. Configuration of each AP is accomplished through telux::data::wlan::IApManager instance requested from factory.
in	<i>numOfSta</i>	Num of Stations to be enabled. If no station is enabled, this argument should be set to 0. Configuration of each Station is accomplished through telux::data::wlan::IStaManager instance requested from factory.

Returns

operation error code (if any). [telux::common::ErrorCode](#).

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.20.4.17 **virtual telux::common::ErrorCode telux::wlan::IWlanDeviceManager::getConfig (int & *numAp*, int & *numSta*) [pure virtual]**

Request Wlan configuration: Returns the configuration that was set using telux::wlan::IWlanDeviceManager::setMode. This might differ from what configuration is has actually been enabled in the system, for instance, when the hardware cannot fully support the configuration that was set. To get the status of current configuration an Wlan enablement, [telux::wlan::IWlanDeviceManager::getStatus](#) should be used.

Parameters

in	<i>numAp</i>	Num of configured APs
in	<i>numSta</i>	Num of configured Stations

Returns

operation error code (if any). [telux::common::ErrorCode](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.20.4.18 virtual `telux::common::ErrorCode telux::wlan::IWlanDeviceManager::getStatus ( bool & isEnabled, std::vector< InterfaceStatus > & status ) [pure virtual]`

Request Wlan status: Return Wlan enablement status and Interface status of APs and Station such as active/inactive, network interface name and hardware device they are mapped to. Results are valid only if Wlan is enabled.

Parameters

in	<i>isEnabled</i>	true: Wlan is enabled. false: Wlan is Disabled.
in	<i>status</i>	vector of interface status InterfaceStatus .

Returns

operation error code (if any). [telux::common::ErrorCode](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.20.4.19 virtual `telux::common::ErrorCode telux::wlan::IWlanDeviceManager::registerListener ( std::weak_ptr< IWlanListener > listener ) [pure virtual]`

Register a listener for specific events in the Wlan Manager

Parameters

in	<i>listener</i>	pointer of IWlanListener object that processes the notification

Returns

Status of registerListener success or suitable status code

5.20.4.20 `virtual telux::common::ErrorCode telux::wlan::IWlanDeviceManager::deregisterListener ( std::weak_ptr< IWlanListener > listener ) [pure virtual]`

Removes a previously added listener.

Parameters

in	<i>listener</i>	pointer of IWlanListener object that needs to be removed

Returns

Status of deregisterListener success or suitable status code

5.20.4.21 `virtual telux::wlan::IWlanDeviceManager::~IWlanDeviceManager ( ) [virtual]`

Destructor for [IWlanDeviceManager](#)

5.20.4.22 `virtual void telux::wlan::IWlanListener::onServiceStatusChange ( telux::common::ServiceStatus status ) [virtual]`

This function is called when service status changes.

Parameters

in	<i>status</i>	- ServiceStatus

5.20.4.23 `virtual void telux::wlan::IWlanListener::onEnableChanged ( bool enable ) [virtual]`

This function is called when Wlan enablement has changed

Parameters

in	<i>enable</i>	True: Wlan is enabled, False: Wlan is disabled

5.20.4.24 `virtual telux::wlan::IWlanListener::~IWlanListener ( ) [virtual]`

5.20.5 Variable Documentation

5.20.5.1 ApInfo telux::wlan::ApNetConfig::info

AP type

5.20.5.2 **ApInterworking telux::wlan::ApNetConfig::interworking**

AP network access (internet/local)

5.20.5.3 **Id telux::wlan::ApConfig::id**

AP id

5.20.5.4 **std::vector<ApNetConfig> telux::wlan::ApConfig::network**

Configurations supported by AP

5.20.5.5 **Id telux::wlan::DeviceIndInfo::id**

AP id device is connected to

5.20.5.6 **std::string telux::wlan::DeviceIndInfo::macAddress**

MAC Address of Wi-Fi device

5.20.5.7 **Id telux::wlan::DeviceInfo::id**

AP id device is connected to

5.20.5.8 **std::string telux::wlan::DeviceInfo::name**

User friendly string that identifies Wi-Fi device

5.20.5.9 **std::string telux::wlan::DeviceInfo::ipv4Address**

IPv4 Address of Wi-Fi device

5.20.5.10 **std::string telux::wlan::DeviceInfo::ipv6Address**

IPv6 Address of Wi-Fi device

5.20.5.11 **std::string telux::wlan::DeviceInfo::macAddress**

MAC Address of Wi-Fi device

5.20.5.12 **Id telux::wlan::DeviceStats::id**

AP id device is connected to

5.20.5.13 **std::string telux::wlan::DeviceStats::macAddress**

MAC Address of Wi-Fi device

5.20.5.14 uint64_t telux::wlan::DeviceStats::bytesTx

Number of bytes transmitted

5.20.5.15 uint64_t telux::wlan::DeviceStats::bytesRx

Number of bytes received

5.20.5.16 HwDeviceType telux::wlan::InterfaceStatus::device

WiFi hardware type

Vector of active APs status

5.20.5.17 std::vector<ApStatus> telux::wlan::InterfaceStatus::apStatus

Vector of active Sta status

5.20.5.18 std::vector<StaStatus> telux::wlan::InterfaceStatus::staStatus

Vector of active Sta status

5.21 Wlan Device Management

This section contains APIs related to device configuration management such as number of access points and stations enabled, enable/disable Wlan, etc.

5.22 Access Point Management

This section contains APIs related to access point configuration management such as private/guest, internet/local access, etc.

5.22.1 Data Structure Documentation

5.22.1.1 class telux::wlan::IApInterfaceManager

Manager class for configuring Wlan Access Points.

Public member functions

- virtual [telux::common::ErrorCode](#) setConfig ([ApConfig](#) config)=0
- virtual [telux::common::ErrorCode](#) getConfig (std::vector< [ApConfig](#) > &config)=0
- virtual [telux::common::ErrorCode](#) getStatus (std::vector< [ApStatus](#) > &status)=0
- virtual [telux::common::ErrorCode](#) getConnectedDevices (std::vector< [DeviceInfo](#) > &clientsInfo)=0
- virtual [telux::common::ErrorCode](#) manageApService ([Id](#) apId, [ServiceOperation](#) opr)=0
- virtual [telux::common::ErrorCode](#) registerListener (std::weak_ptr< [IApListener](#) > listener)=0
- virtual [telux::common::ErrorCode](#) deregisterListener (std::weak_ptr< [IApListener](#) > listener)=0
- virtual [~IApInterfaceManager](#) ()
- virtual [telux::common::ErrorCode](#) getConnectedDevicesStats (std::vector< [DeviceStats](#) > &clientsStats)=0

5.22.1.2 class telux::wlan::IApListener

Public member functions

- virtual void onApDeviceStatusChanged ([ApDeviceConnectionEvent](#) event, std::vector< [DeviceIndInfo](#) > info)
- virtual void onApBandChanged ([BandType](#) radio)
- virtual [~IApListener](#) ()
- virtual void onApDeviceStatusChanged ([ApDeviceConnectionEvent](#) event, std::vector< [DeviceInfo](#) > info)

5.23 Station Management

- [Wlan](#)

This section contains APIs related to station configuration management such as static/dynamic IP, bridge/router mode, etc.

5.23.1 Data Structure Documentation

5.23.1.1 struct telux::wlan::StaStaticIpConfig

Static IP Configuration

Data fields

Type	Field	Description
string	ipAddr	IPv4 address to be assigned.
string	gwIpAddr	IPv4 address of the gateway.
string	netMask	Subnet mask.
string	dnsAddr	DNS IPv4 address.

5.23.1.2 struct telux::wlan::StaConfig

Station Configuration

Data fields

Type	Field	Description
Id	staId	Id of station backhaul
StaIpConfig	ipConfig	IP configuration of station backhaul
StaStaticIp-Config	staticIpConfig	Static IP configuration if selected
StaBridgeMode	bridgeMode	Station configuration as Router/bridge

5.23.2 Enumeration Type Documentation

5.23.2.1 enum telux::wlan::StaIpConfig

Station Connection IP Type

Enumerator

DYNAMIC_IP Station is configured with dynamic IP

STATIC_IP Station is configured with Static IP

5.23.2.2 enum telux::wlan::StaBridgeMode

Bridge/Router Mode

Enumerator

ROUTER Station is in Router Mode

BRIDGE Station is in Bridge Mode

5.23.3 Function Documentation

5.23.3.1 **virtual telux::common::ErrorCode telux::wlan::IStaInterfaceManager::setIpConfig (Id *stald*, StaIpConfig *ipConfig*, StaStaticIpConfig *staticIpConfig*) [pure virtual]**

Set Station IP Configurations: Set Station IP configuration dynamic/static and static IP address if selected. If API is called when WLAN is disabled, changes will take effect when WLAN is enabled using [telux::wlan::IWlanDeviceManager::enable](#) API. If API is called when WLAN is enabled, changes will take effect after restarting wpa_supplicant by calling [telux::wlan::IStaInterfaceManager::manageStaService](#)

Parameters

in	<i>stald</i>	Station Identifier telux::wlan::Id
in	<i>ipConfig</i>	Static/Dynamic IP configuration telux::wlan::StaIpConfig .
in	<i>staticIpConfig</i>	Static IP configuration, not used if station was configured to use dynamic IP.

Returns

operation error code (if any). [telux::common::ErrorCode](#).

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.23.3.2 **virtual telux::common::ErrorCode telux::wlan::IStaInterfaceManager::setBridgeMode (Id *stald*, StaBridgeMode *bridgeMode*) [pure virtual]**

Set Station backhaul to act as router or bridge: Sets Station to act as router or bridge where station internal clients get public IP addresses. If API is called when WLAN is disabled, changes will take effect when WLAN is enabled using [telux::wlan::IWlanDeviceManager::enable](#) API. If API is called when WLAN is enabled, changes will take effect after restarting wpa_supplicant by calling [telux::wlan::IStaInterfaceManager::manageStaService](#)

Parameters

in	<i>stald</i>	Station Identifier telux::wlan::Id
in	<i>bridgeMode</i>	bridgeMode telux::wlan::StaBridgeMode

Returns

operation error code (if any). [telux::common::ErrorCode](#).

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.23.3.3 **virtual telux::common::ErrorCode telux::wlan::IStaInterfaceManager::getConfig (std::vector< StaConfig > & config) [pure virtual]**

Request current station configurations: Returns configurations set by [telux::wlan::IStaInterfaceManager::setIpConfig](#) and [telux::wlan::IStaInterfaceManager::setBridgeMode](#)

Parameters

in	config	Station configurations telux::wlan::StaConfig

Returns

operation error code (if any). [telux::common::ErrorCode](#).

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.23.3.4 **virtual telux::common::ErrorCode telux::wlan::IStaInterfaceManager::getStatus (std::vector< StaStatus > & status) [pure virtual]**

Request current station status: Returns current Sta interface status such as network interface name and IP address.

Parameters

in	status	Station Status telux::wlan::StaStatus

Returns

operation error code (if any). [telux::common::ErrorCode](#).

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.23.3.5 **virtual telux::common::ErrorCode telux::wlan::IStaInterfaceManager::manageStaService (Id stald, ServiceOperation opr) [pure virtual]**

Execute an operation on wpa_supplicant service. Provides ability for client to either stop/start or restart wpa_supplicant service for selected station. Restarting wpa_supplicant service is required for any changes made to wpa_supplicant.conf file to take effect. Station selected to execute operation on, will temporarily go out of service when this API is called. This API should be called only when station mode is configured through [telux::wlan::IDeviceManager::setMode](#)

in	<i>staId</i>	Station identifier to execute operation on. telux::wlan::Id
in	<i>opr</i>	Operation to be performed on wpa_supplicant telux::wlan::ServiceOperation

Returns

operation error code (if any). [telux::common::ErrorCode](#).

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.23.3.6 **virtual telux::common::ErrorCode telux::wlan::IStaInterfaceManager::registerListener (std::weak_ptr< IStaListener > *listener*) [pure virtual]**

Register as a listener for specific events defined in [telux::wlan::IStaListener](#)

Parameters

in	<i>listener</i>	pointer of IStaListener object that processes the notification

Returns

operation error code (if any). [telux::common::ErrorCode](#).

5.23.3.7 **virtual telux::common::ErrorCode telux::wlan::IStaInterfaceManager::deregisterListener (std::weak_ptr< IStaListener > *listener*) [pure virtual]**

Removes a previously added listener.

Parameters

in	<i>listener</i>	pointer of IStaListener object that needs to be removed

Returns

operation error code (if any). [telux::common::ErrorCode](#).

5.23.3.8 **virtual telux::wlan::IStaInterfaceManager::~~IStaInterfaceManager () [virtual]**

5.23.3.9 **virtual void telux::wlan::IStaListener::onStationStatusChanged (std::vector< StaStatus > *staStatus*) [virtual]**

This function is called when Station Status Changes

in	<i>status</i>	List of station state telux::wlan::StaStatus

5.23.3.10 **virtual void telux::wlan::IStaListener::onStationBandChanged (BandType *radio*) [virtual]**

This function is called when Station switch to different operation band

Parameters

in	<i>radio</i>	New Station operation band telux::wlan::BandType

5.23.3.11 **virtual telux::wlan::IStaListener::~~IStaListener () [virtual]**

5.23.4 Variable Documentation

5.23.4.1 **std::string telux::wlan::StaStaticIpConfig::ipAddr**

IPv4 address to be assigned.

5.23.4.2 **std::string telux::wlan::StaStaticIpConfig::gwIpAddr**

IPv4 address of the gateway.

5.23.4.3 **std::string telux::wlan::StaStaticIpConfig::netMask**

Subnet mask.

5.23.4.4 **std::string telux::wlan::StaStaticIpConfig::dnsAddr**

DNS IPv4 address.

5.23.4.5 **Id telux::wlan::StaConfig::stald**

Id of station backhaul

5.23.4.6 **StalpConfig telux::wlan::StaConfig::ipConfig**

IP configuration of station backhaul

5.23.4.7 StaStaticIpConfig telux::wlan::StaConfig::staticIpConfig

Static IP configuration if selected

5.23.4.8 StaBridgeMode telux::wlan::StaConfig::bridgeMode

Station configuration as Router/bridge

5.24 Telematics_cv2x_cpp

•

5.24.1 Data Structure Documentation

5.24.1.1 class telux::cv2x::ICv2xThrottleManagerListener

Listener class for getting filter rate update notification.

Public member functions

- virtual void [onFilterRateAdjustment](#) (int rate)
- virtual void [onServiceStatusChange](#) (telux::common::ServiceStatus status)
- virtual [~ICv2xThrottleManagerListener](#) ()

5.24.1.1.1 Constructors and Destructors

5.24.1.1.1.1 virtual telux::cv2x::ICv2xThrottleManagerListener::~~ICv2xThrottleManagerListener ()
[virtual]

Destructor of [ICv2xThrottleManagerListener](#)

5.24.1.1.2 Member Function Documentation

5.24.1.1.2.1 virtual void telux::cv2x::ICv2xThrottleManagerListener::onFilterRateAdjustment (int *rate*)
[virtual]

This API is invoked to advise the client to adjust the incoming message filtering rate by *rate* messages/second. If the *rate* is positive, it indicates the client to filter *rate* more messages/second. If the *rate* is negative, it indicates the client to filter *rate* less messages/second.

Parameters

in	<i>rate</i>	the reported filter rate adjustment value.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.24.1.1.2.2 virtual void telux::cv2x::ICv2xThrottleManagerListener::onServiceStatusChange (telux::common::ServiceStatus *status*) [virtual]

This API is invoked when the service status changes for example when a subsystem restart (SSR) occurs.

in	status	- ServiceStatus

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.24.1.2 class telux::cv2x::ICv2xThrottleManager

ThrottleManager provides throttle manager client interface.

ThrottleManager provides APIs that allows applications to specify the incoming verification load on the system. This is used to make decisions on how to optimally use the resources available in the system. The API also provides feedback to clients on the suggested filtering that needs to be done when the incoming message verification rate exceeds the instantaneous system capacity.

Public member functions

- virtual `telux::common::ServiceStatus getServiceStatus ()=0`
- virtual `std::future< bool > onSubsystemReady ()=0`
- virtual `telux::common::Status registerListener (std::weak_ptr< ICv2xThrottleManagerListener > listener)=0`
- virtual `telux::common::Status deregisterListener (std::weak_ptr< ICv2xThrottleManagerListener > listener)=0`
- virtual `telux::common::Status setVerificationLoad (int load, setVerificationLoadCallback cb)=0`
- virtual `~ICv2xThrottleManager ()`

5.24.1.2.1 Constructors and Destructors

5.24.1.2.1.1 `virtual telux::cv2x::ICv2xThrottleManager::~~ICv2xThrottleManager ( ) [virtual]`

5.24.1.2.2 Member Function Documentation

5.24.1.2.2.1 `virtual telux::common::ServiceStatus telux::cv2x::ICv2xThrottleManager::getServiceStatus ( ) [pure virtual]`

This status indicates whether the object is in a usable state.

Returns

SERVICE_AVAILABLE - If location manager is ready for service. SERVICE_UNAVAILABLE - If location manager is temporarily unavailable. SERVICE_FAILED - If location manager encountered an irrecoverable failure.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.24.1.2.2.2 `virtual std::future<bool> telux::cv2x::ICv2xThrottleManager::onSubsystemReady ( )`
[pure virtual]

Wait for Cv2x throttle manager client to be ready.

Returns

A future that caller can wait on to be notified when Cv2x throttle manager client become ready.

5.24.1.2.2.3 `virtual telux::common::Status telux::cv2x::ICv2xThrottleManager::registerListener (`
`std::weak_ptr< ICv2xThrottleManagerListener > listener )` **[pure virtual]**

Registers a listener to receive the updated filer rate adjustment data.

Parameters

in	<i>listener</i>	- Listener that implement ICv2xThrottleManagerListener interface.

5.24.1.2.2.4 `virtual telux::common::Status telux::cv2x::ICv2xThrottleManager::deregisterListener (`
`std::weak_ptr< ICv2xThrottleManagerListener > listener )` **[pure virtual]**

Deregister a [ICv2xThrottleManagerListener](#).

Parameters

in	<i>listener</i>	- Previously registered Cv2xThrottleManagerListener that is deregistered.

5.24.1.2.2.5 `virtual telux::common::Status telux::cv2x::ICv2xThrottleManager::setVerificationLoad ( int`
`load, setVerificationLoadCallback cb )` **[pure virtual]**

Set current measured/average verification load.

Parameters

in	<i>load</i>	- current measured verification load(verification/second).

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.24.1.3 class telux::cv2x::ICv2xTxStatusReportListener

Listeners for CV2X Tx status report must implement this interface.

Public member functions

- virtual void onTxStatusReport (const TxStatusReport &info)
- virtual ~ICv2xTxStatusReportListener ()

5.24.1.3.1 Constructors and Destructors

5.24.1.3.1.1 virtual telux::cv2x::ICv2xTxStatusReportListener::~~ICv2xTxStatusReportListener ()
[virtual]

Destructor for ICv2xTxStatusReportListener

5.24.1.3.2 Member Function Documentation

5.24.1.3.2.1 virtual void telux::cv2x::ICv2xTxStatusReportListener::onTxStatusReport (const
TxStatusReport & info) [virtual]

Called when a CV2X transport block is transmitted in low layer if CV2X Tx status report has been enabled by calling setTxStatusReport.

Parameters

in	info	- Tx status of the transport block.

5.25 Telematics_net

5.25.1 Data Structure Documentation

5.25.1.1 class telux::data::net::IBridgeManager

[IBridgeManager](#) provides APIs to enable/disable and set/get/delete software bridges for various WLAN and Ethernet interfaces. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready.

Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status enableBridge](#) (bool enable, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status addBridge](#) ([BridgeInfo](#) config, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status requestBridgeInfo](#) ([BridgeInfoResponseCb](#) callback)=0
- virtual [telux::common::Status removeBridge](#) (std::string ifaceName, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::ErrorCode setInterfaceBridge](#) ([InterfaceType](#) ifaceType, uint32_t bridgeId)=0
- virtual [telux::common::ErrorCode getInterfaceBridge](#) ([InterfaceType](#) ifaceType, uint32_t &bridgeId)=0
- virtual [telux::common::Status registerListener](#) (std::weak_ptr< [IBridgeListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak_ptr< [IBridgeListener](#) > listener)=0
- virtual [~IBridgeManager](#) ()

5.25.1.1.1 Constructors and Destructors

5.25.1.1.1.1 virtual [telux::data::net::IBridgeManager::~~IBridgeManager](#) () [virtual]

Destructor for [IBridgeManager](#)

5.25.1.1.2 Member Function Documentation

5.25.1.1.2.1 virtual bool [telux::data::net::IBridgeManager::isSubsystemReady](#) () [pure virtual]

Checks if the data subsystem is ready.

Returns

True if the Bridge Manager is ready for service, otherwise returns false.

5.25.1.1.2.2 `virtual std::future<bool> telux::data::net::IBridgeManager::onSubsystemReady ( )`
[pure virtual]

Wait for data subsystem to be ready.

Returns

A future that caller can wait until the Bridge Manager succeed/fail to be ready.

5.25.1.1.2.3 `virtual telux::common::Status telux::data::net::IBridgeManager::enableBridge ( bool`
***enable*, telux::common::ResponseCallback *callback* = nullptr) [pure virtual]**

Enable/Disable the software bridge in the system. It will affect all the configured software bridges for various interfaces.

Parameters

in	<i>enable</i>	TRUE to enable, FALSE to disable the bridge
in	<i>callback</i>	Optional callback to get the response for enableBridge

Returns

Status of enableBridge request i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.1.2.4 `virtual telux::common::Status telux::data::net::IBridgeManager::addBridge ( BridgeInfo`
***config*, telux::common::ResponseCallback *callback* = nullptr) [pure virtual]**

Add software bridge configuration for an interface.

Parameters

in	<i>config</i>	configuration for an interface
in	<i>callback</i>	Optional callback to get the response for addBridge

Returns

Status of addBridge request i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.1.2.5 **virtual telux::common::Status telux::data::net::IBridgeManager::requestBridgeInfo (BridgeInfoResponseCb *callback*) [pure virtual]**

Request information about all the software bridge configurations in the system

Parameters

in	<i>callback</i>	Response callback with list of bridge configurations

Returns

Status of requestBridgeInfo request i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

5.25.1.1.2.6 **virtual telux::common::Status telux::data::net::IBridgeManager::removeBridge (std::string *ifaceName*, telux::common::ResponseCallback *callback* = nullptr) [pure virtual]**

Delete a software bridge configuration for an interface.

Parameters

in	<i>ifaceName</i>	Name of the interface whose configuration needs to be deleted
in	<i>callback</i>	Optional callback to get the response for removeBridge

Returns

Status of removeBridge request i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

5.25.1.1.2.7 **virtual telux::common::ErrorCode telux::data::net::IBridgeManager::setInterfaceBridge (InterfaceType *ifaceType*, uint32_t *bridgeId*) [pure virtual]**

Associates a bridge with the specified hardware interface [telux::data::InterfaceType](#).

For tethered traffic, the default route always flows through the default bridge (bridge0). This API provides the ability to use a non-default route for data traffic, using on-demand bridges instead.

The client shall specify the interface type on which the operation has to be executed and provide the bridgeId that needs to be associated with that particular interface type. If the client provides a bridgeId for a bridge that doesn't exist the bridge is created and then the interface is mapped to it.

The mapping is persistent across reboots. In case of WLAN_AP, this API should be used before the AP configuration is done. If WLAN is enabled and this API is invoked, changes only reflect when WLAN is

re-enabled.

Parameters

in	<i>ifaceType</i>	telux::data::InterfaceType
in	<i>bridgeId</i>	Bridge ID that will be mapped to the hardware interface

Returns

Operation [telux::common::ErrorCode](#) (if any).

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::ErrorCode telux::data::net::IBridgeManager::getInterfaceBridge ( InterfaceType ifaceType, uint32_t & bridgeId ) [pure virtual]`

Retrieves the bridge ID currently mapped to the specified interface type.

Parameters

in	<i>ifaceType</i>	telux::data::InterfaceType
out	<i>bridgeId</i>	Bridge ID mapped to the hardware interface

Returns

Operation [telux::common::ErrorCode](#) (if any).

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::IBridgeManager::registerListener ( std::weak_ptr< IBridgeListener > listener ) [pure virtual]`

Register Bridge Manager as listener for Data Service health events like data service available or data service not available.

Parameters

in	<i>listener</i>	pointer of IBridgeListener object that processes the notification

Returns

Status of registerListener success or suitable status code

5.25.1.1.2.10 `virtual telux::common::Status telux::data::net::IBridgeManager::deregisterListener ( std::weak_ptr< IBridgeListener > listener ) [pure virtual]`

Removes a previously added listener.

Parameters

in	<i>listener</i>	pointer of IBridgeListener object that needs to be removed

Returns

Status of deregisterListener success or suitable status code

5.25.1.2 class telux::data::net::IBridgeListener

Interface for Bridge listener object. Client needs to implement this interface to get access to Bridge services notifications like onServiceStatusChange.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

Public member functions

- virtual void [onServiceStatusChange](#) ([telux::common::ServiceStatus](#) status)
- virtual [~IBridgeListener](#) ()

5.25.1.2.1 Constructors and Destructors

5.25.1.2.1.1 `virtual telux::data::net::IBridgeListener::~~IBridgeListener ( ) [virtual]`

Destructor for [IBridgeListener](#)

5.25.1.2.2 Member Function Documentation

5.25.1.2.2.1 `virtual void telux::data::net::IBridgeListener::onServiceStatusChange ( telux::common::ServiceStatus status ) [virtual]`

This function is called when service status changes.

Parameters

in	<i>status</i>	- ServiceStatus

5.25.1.3 class telux::data::net::IFirewallManager

FirewallManager is a primary interface that filters and controls the network traffic on a pre-configured set of rules. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready.

Public member functions

- virtual bool `isSubsystemReady` ()=0
- virtual std::future< bool > `onSubsystemReady` ()=0
- virtual `telux::common::Status setFirewallConfig` (`FirewallConfig` fwConfig, `telux::common::ResponseCallback` callback=nullptr)=0
- virtual `telux::common::Status requestFirewallConfig` (`BackhaulInfo` bhInfo, `FirewallConfigCb` callback)=0
- virtual `telux::common::Status addFirewallEntry` (`FirewallEntryInfo` entry, `telux::common::ResponseCallback` callback=nullptr)=0
- virtual `telux::common::Status requestFirewallEntries` (`BackhaulInfo` bhInfo, `FirewallEntryInfoCb` callback)=0
- virtual `telux::common::Status removeFirewallEntry` (`BackhaulInfo` bhInfo, uint32_t handle, `telux::common::ResponseCallback` callback=nullptr)=0
- virtual `telux::common::Status enableDmz` (`DmzConfig` config, `telux::common::ResponseCallback` callback=nullptr)=0
- virtual `telux::common::Status disableDmz` (`BackhaulInfo` bhInfo, const `IpFamilyType` ipType, `telux::common::ResponseCallback` callback=nullptr)=0
- virtual `telux::common::Status requestDmzEntry` (`BackhaulInfo` bhInfo, `DmzEntryInfoCb` callback)=0
- virtual `telux::common::Status registerListener` (std::weak_ptr< `IFirewallListener` > listener)=0
- virtual `telux::common::Status deregisterListener` (std::weak_ptr< `IFirewallListener` > listener)=0
- virtual `telux::data::OperationType getOperationType` ()=0
- virtual `telux::common::Status setFirewall` (int profileId, bool enable, bool allowPackets, `telux::common::ResponseCallback` callback=nullptr)=0
- virtual `telux::common::Status requestFirewallStatus` (int profileId, `FirewallStatusCb` callback)=0
- virtual `telux::common::Status addFirewallEntry` (int profileId, std::shared_ptr< `IFirewallEntry` > entry, `telux::common::ResponseCallback` callback=nullptr)=0
- virtual `telux::common::Status requestFirewallEntries` (int profileId, `FirewallEntriesCb` callback)=0
- virtual `telux::common::Status removeFirewallEntry` (int profileId, uint32_t handle, `telux::common::ResponseCallback` callback=nullptr)=0
- virtual `telux::common::Status enableDmz` (int profileId, const std::string ipAddr, `telux::common::ResponseCallback` callback=nullptr)=0
- virtual `telux::common::Status disableDmz` (int profileId, const `telux::data::IpFamilyType` ipType, `telux::common::ResponseCallback` callback=nullptr)=0
- virtual `telux::common::Status requestDmzEntry` (int profileId, `DmzEntriesCb` dmzCb)=0
- virtual `~IFirewallManager` ()

5.25.1.3.1 Constructors and Destructors

5.25.1.3.1.1 `virtual telux::data::net::IFirewallManager::~IFirewallManager ( ) [virtual]`

Destructor for [IFirewallManager](#)

5.25.1.3.2 Member Function Documentation

5.25.1.3.2.1 `virtual bool telux::data::net::IFirewallManager::isSubsystemReady ( ) [pure virtual]`

Checks if the data subsystem is ready.

Returns

True if Firewall Manager is ready for service, otherwise returns false.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.3.2.2 `virtual std::future<bool> telux::data::net::IFirewallManager::onSubsystemReady ( ) [pure virtual]`

Wait for data subsystem to be ready.

Returns

A future that caller can wait on to be notified when firewall manager is ready.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.3.2.3 `virtual telux::common::Status telux::data::net::IFirewallManager::setFirewallConfig ( FirewallConfig fwConfig, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Sets firewall configuration to enable or disable firewall and update configuration to drop or accept the packets matching the rules on slot ID, profile ID and backhaul type.

On platforms with Access control enabled, Caller needs to have TELUX_DATA_NETWORK_CONFIG permission to invoke this API successfully.

Parameters

in	<i>config</i>	Firewall configuration telux::data::FirewallConfig.

Returns

Status of setFirewallConfig i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.3.2.4 **virtual telux::common::Status telux::data::net::IFirewallManager::requestFirewallConfig (BackhaulInfo *bhInfo*, FirewallConfigCb *callback*) [pure virtual]**

Request status of firewall settings on specific backhaul

Parameters

in	<i>bhInfo</i>	Backhaul Information to request firewall status for.
in	<i>callback</i>	callback to get the response of requestFirewallConfig

Returns

Status of requestFirewallConfig i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.3.2.5 **virtual telux::common::Status telux::data::net::IFirewallManager::addFirewallEntry (FirewallEntryInfo *entry*, telux::common::ResponseCallback *callback* = *nullptr*) [pure virtual]**

Adds the firewall rule to specific backhaul

On platforms with Access control enabled, Caller needs to have TELUX_DATA_NETWORK_CONFIG permission to invoke this API successfully.

Parameters

in	<i>entries</i>	Firewall rules entries settings.
in	<i>callback</i>	optional callback to get the response addFirewallEntry

Returns

Status of addFirewallEntry i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.3.2.6 **virtual telux::common::Status telux::data::net::IFirewallManager::requestFirewallEntries (BackhaulInfo *bhInfo*, FirewallEntryInfoCb *callback*) [pure virtual]**

Request Firewall rules for specific backhaul

Parameters

in	<i>bhInfo</i>	Backhaul Information to request firewall entries for.
in	<i>callback</i>	callback to get the response requestFirewallEntries.

Returns

Status of requestFirewallEntries i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.3.2.7 **virtual telux::common::Status telux::data::net::IFirewallManager::removeFirewallEntry (BackhaulInfo *bhInfo*, uint32_t *handle*, telux::common::ResponseCallback *callback* = nullptr) [pure virtual]**

Remove firewall entry set on particular backhaul

Parameters

in	<i>bhInfo</i>	Backhaul information to remove firewall entries from.
in	<i>handle</i>	handle of Firewall entry to be removed. To retrieve the handle, first use requestFirewallEntries() to get the list of entries added in the system. And then use IFirewallEntry::getHandle()
in	<i>callback</i>	callback to get the response removeFirewallEntry

Returns

Status of removeFirewallEntry i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.3.2.8 **virtual telux::common::Status telux::data::net::IFirewallManager::enableDmz (DmzConfig *config*, telux::common::ResponseCallback *callback* = nullptr) [pure virtual]**

Enable demilitarized zone (DMZ) on particular backhaul

in	<i>config</i>	DMZ configuration to be enabled
in	<i>callback</i>	optional callback to get the response addDmz

Returns

Status of enableDmz i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.3.2.9 `virtual telux::common::Status telux::data::net::IFirewallManager::disableDmz ( Backhaul-Info bhInfo, const IpFamilyType ipType, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Disable demilitarized zone (DMZ) on particular backhaul

Parameters

in	<i>bhInfo</i>	Backhaul on which DMZ will be disabled.
in	<i>ipType</i>	Specify IP type of the DMZ to be disabled
in	<i>callback</i>	optional callback to get the response removeDmz

Returns

Status of disableDmz i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.3.2.10 `virtual telux::common::Status telux::data::net::IFirewallManager::requestDmzEntry ( BackhaulInfo bhInfo, DmzEntryInfoCb callback ) [pure virtual]`

Request DMZ entry on particular backhaul that was previously set using enableDmz API

Parameters

in	<i>bhInfo</i>	Backhaul info on which DMZ entries are requested.
in	<i>callback</i>	callback to get the response requestDmzEntry

Returns

Status of requestDmzEntry i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.3.2.11 `virtual telux::common::Status telux::data::net::IFirewallManager::registerListener ( std::weak_ptr< IFirewallListener > listener ) [pure virtual]`

Register Firewall Manager as listener for Data Service health events like data service available or data service not available.

Parameters

<i>in</i>	<i>listener</i>	pointer of IFirewallListener object that processes the notification

Returns

Status of registerListener success or suitable status code

5.25.1.3.2.12 `virtual telux::common::Status telux::data::net::IFirewallManager::deregisterListener ( std::weak_ptr< IFirewallListener > listener ) [pure virtual]`

Removes a previously added listener.

Parameters

<i>in</i>	<i>listener</i>	pointer of IFirewallListener object that needs to be removed

Returns

Status of deregisterListener success or suitable status code

5.25.1.3.2.13 `virtual telux::data::OperationType telux::data::net::IFirewallManager::getOperationType ( ) [pure virtual]`

Get the associated operation type for this instance.

Returns

OperationType of getOperationType i.e. LOCAL or REMOTE.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.3.2.14 `virtual telux::common::Status telux::data::net::IFirewallManager::setFirewall ( int profileId, bool enable, bool allowPackets, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Sets firewall configuration to enable or disable and update configuration to drop or accept the packets matching the rules.

Parameters

in	<i>profileId</i>	Profile identifier corresponding to the WWAN network interface. Firewall will be configured on this WWAN network interface.
in	<i>enable</i>	Indicates whether the firewall is enabled
in	<i>allowPackets</i>	Indicates whether to accept or drop packets matching the rules
in	<i>callback</i>	optional callback to get the response setFirewall

Returns

Status of setFirewall i.e. success or suitable status code.

Deprecated Use telux::data::setFirewallConfig API to set firewall on any backhaul

5.25.1.3.2.15 `virtual telux::common::Status telux::data::net::IFirewallManager::requestFirewallStatus ( int profileId, FirewallStatusCb callback ) [pure virtual]`

Request status of firewall

Parameters

in	<i>profileId</i>	Profile identifier for which firewall status is requested.
in	<i>callback</i>	callback to get the response of requestFirewallStatus

Returns

Status of requestFirewallStatus i.e. success or suitable status code.

Deprecated Use telux::data::requestFirewallConfig API to request firewall status on any backhaul

5.25.1.3.2.16 `virtual telux::common::Status telux::data::net::IFirewallManager::addFirewallEntry ( int profileId, std::shared_ptr< IFirewallEntry > entry, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Adds the firewall rule

Parameters

in	<i>profileId</i>	Profile identifier on which firewall rule will be added.
in	<i>entry</i>	Firewall entry based on protocol type

in	<i>callback</i>	optional callback to get the response addFirewallEntry

Returns

Status of addFirewallEntry i.e. success or suitable status code.

Deprecated Use telux::data::addFirewallEntry API to add firewall rule on any backhaul

5.25.1.3.2.17 **virtual telux::common::Status telux::data::net::IFirewallManager::requestFirewallEntries (int *profileId*, FirewallEntriesCb *callback*) [pure virtual]**

Request Firewall rules

Parameters

in	<i>profileId</i>	Profile identifier on which firewall entries are retrieved.
in	<i>callback</i>	callback to get the response requestFirewallEntries.

Returns

Status of requestFirewallEntries i.e. success or suitable status code.

Deprecated Use telux::data::requestFirewallEntries API to request firewall rules on any backhaul

5.25.1.3.2.18 **virtual telux::common::Status telux::data::net::IFirewallManager::removeFirewallEntry (int *profileId*, uint32_t *handle*, telux::common::ResponseCallback *callback* = nullptr) [pure virtual]**

Remove firewall entry

Parameters

in	<i>profileId</i>	Profile identifier on which firewall entry will be removed.
in	<i>handle</i>	handle of Firewall entry to be removed. To retrieve the handle, first use requestFirewallEntries() to get the list of entries added in the system. And then use IFirewallEntry::getHandle()
in	<i>callback</i>	callback to get the response removeFirewallEntry

Returns

Status of removeFirewallEntry i.e. success or suitable status code.

Deprecated Use telux::data::removeFirewallEntry API to remove firewall rule from any backhaul

5.25.1.3.2.19 `virtual telux::common::Status telux::data::net::IFirewallManager::enableDmz ( int profileId, const std::string ipAddr, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Enable demilitarized zone (DMZ)

Parameters

in	<i>profileId</i>	Profile identifier on which DMZ will be enabled.
in	<i>ipAddr</i>	IP address for which DMZ will be enabled
in	<i>callback</i>	optional callback to get the response addDmz

Returns

Status of enableDmz i.e. success or suitable status code.

Deprecated Use telux::data::enableDmz API to enable DMZ on any backhaul

5.25.1.3.2.20 `virtual telux::common::Status telux::data::net::IFirewallManager::disableDmz ( int profileId, const telux::data::IpFamilyType ipType, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Disable demilitarized zone (DMZ)

Parameters

in	<i>profileId</i>	Profile identifier on which DMZ will be disabled.
in	<i>ipType</i>	Specify IP type of the DMZ to be disabled
in	<i>callback</i>	optional callback to get the response removeDmz

Returns

Status of disableDmz i.e. success or suitable status code.

Deprecated Use telux::data::disableDmz API to Disable DMZ on any backhaul

5.25.1.3.2.21 `virtual telux::common::Status telux::data::net::IFirewallManager::requestDmzEntry ( int profileId, DmzEntriesCb dmzCb ) [pure virtual]`

Request DMZ entry that was previously set using enableDmz API

Parameters

in	<i>profileId</i>	Profile identifier on which DMZ entries are requested.
in	<i>dmzCb</i>	callback to get the response requestDmzEntry

Returns

Status of requestDmzEntry i.e. success or suitable status code.

Deprecated Use `telux::data::requestDmzEntry` API to request DMZ on any backhaul

5.25.1.4 class telux::data::net::IFirewallEntry

Firewall entry class is used for configuring firewall rules.

Public member functions

- virtual `std::shared_ptr<IIPFilter>` `getIPProtocolFilter` ()=0
- virtual `telux::data::Direction` `getDirection` ()=0
- virtual `telux::data::IpFamilyType` `getIpFamilyType` ()=0
- virtual `uint32_t` `getHandle` ()=0
- virtual `~IFirewallEntry` ()

Static Public Attributes

- static const `uint32_t` `INVALID_HANDLE` = 0

5.25.1.4.1 Constructors and Destructors

5.25.1.4.1.1 virtual `telux::data::net::IFirewallEntry::~IFirewallEntry` () [virtual]

Destructor for `IFirewallEntry`

5.25.1.4.2 Member Function Documentation

5.25.1.4.2.1 virtual `std::shared_ptr<IIPFilter>` `telux::data::net::IFirewallEntry::getIPProtocolFilter` () [pure virtual]

Get IPProtocol filter type

Returns

`telux::data::IIPFilter`.

5.25.1.4.2.2 virtual `telux::data::Direction` `telux::data::net::IFirewallEntry::getDirection` () [pure virtual]

Get firewall direction

Returns

`telux::data::Direction`.

5.25.1.4.2.3 `virtual telux::data::IpFamilyType telux::data::net::IFirewallEntry::getIpFamilyType ( )`
`[pure virtual]`

Get Ip FamilyType

Returns

[telux::data::IpFamilyType](#).

5.25.1.4.2.4 `virtual uint32_t telux::data::net::IFirewallEntry::getHandle ( )` `[pure virtual]`

Get the unique handle identifying this Firewall entry in the system

Returns

uint32_t handle if initialized or INVALID_HANDLE otherwise

5.25.1.4.3 Field Documentation

5.25.1.4.3.1 `const uint32_t telux::data::net::IFirewallEntry::INVALID_HANDLE = 0` `[static]`

5.25.1.5 class telux::data::net::IFirewallListener

Interface for Firewall listener object. Client needs to implement this interface to get access to Firewall services notifications like onServiceStatusChange.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

Public member functions

- virtual void [onServiceStatusChange](#) ([telux::common::ServiceStatus](#) status)
- virtual [~IFirewallListener](#) ()

5.25.1.5.1 Constructors and Destructors

5.25.1.5.1.1 `virtual telux::data::net::IFirewallListener::~~IFirewallListener ( )` `[virtual]`

Destructor for [IFirewallListener](#)

5.25.1.5.2 Member Function Documentation

5.25.1.5.2.1 `virtual void telux::data::net::IFirewallListener::onServiceStatusChange ( telux::common::ServiceStatus status )` `[virtual]`

This function is called when service status changes.

in	status	- ServiceStatus

5.25.1.6 struct telux::data::net::L2tpSessionConfig

L2TP tunnel sessions configuration

Data fields

Type	Field	Description
uint32_t	locId	Local session id
uint32_t	peerId	Peer session id

5.25.1.7 struct telux::data::net::L2tpTunnelConfig

L2TP tunnel configuration

Data fields

Type	Field	Description
L2tpProtocol	prot	Encapsulation protocols
uint32_t	locId	Local tunnel id
uint32_t	peerId	Peer tunnel id
uint32_t	localUdpPort	Local udp port - if UDP encapsulation is used
uint32_t	peerUdpPort	Peer udp port - if IP encapsulation is used
string	peerIpv6Addr	Peer IPv6 Address - for Ipv6 tunnels
string	peerIpv4Addr	Peer IPv4 Address - for Ipv4 tunnels
string	locIface	interface name to create L2TP tunnel on
IpFamilyType	ipType	Ip family type telux::data::IpFamilyType
vector< L2tpSessionConfig >	sessionConfig	List of L2tp tunnel sessions

5.25.1.8 struct telux::data::net::L2tpSysConfig

L2TP Configuration

Data fields

Type	Field	Description
vector< L2tpTunnelConfig >	configList	List of L2tp tunnel configurations
bool	enableMtu	Enable MTU size setting on underlying interfaces to avoid segmentation
bool	enableTcpMss	Enable TCP MSS clamping on L2TP interfaces to avoid segmentation

5.25.1.9 class telux::data::net::IL2tpManager

L2tpManager is a primary interface for configuring L2TP Service. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready.

Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status setConfig](#) (bool enable, bool enableMss, bool enableMtu, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status addTunnel](#) (const [L2tpTunnelConfig](#) &l2tpTunnelConfig, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status requestConfig](#) ([L2tpConfigCb](#) l2tpConfigCb)=0
- virtual [telux::common::Status removeTunnel](#) (uint32_t tunnelId, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status registerListener](#) (std::weak_ptr< [IL2tpListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak_ptr< [IL2tpListener](#) > listener)=0
- virtual [~IL2tpManager](#) ()

5.25.1.9.1 Constructors and Destructors

5.25.1.9.1.1 virtual [telux::data::net::IL2tpManager::~~IL2tpManager](#) () [**virtual**]

Destructor for [IL2tpManager](#)

5.25.1.9.2 Member Function Documentation

5.25.1.9.2.1 virtual bool [telux::data::net::IL2tpManager::isSubsystemReady](#) () [**pure virtual**]

Checks if the L2tp manager subsystem is ready.

Returns

True if L2tp Manager is ready for service, otherwise returns false.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.9.2.2 virtual std::future<bool> [telux::data::net::IL2tpManager::onSubsystemReady](#) () [**pure virtual**]

Wait for L2tp manager subsystem to be ready.

Returns

A future that caller can wait on to be notified when L2tp manager is ready.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.9.2.3 `virtual telux::common::Status telux::data::net::IL2tpManager::setConfig ( bool enable, bool enableMss, bool enableMtu, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Enable L2TP for unmanaged Tunnel State

Parameters

in	<i>enable</i>	Enable/Disable L2TP for unmanaged tunnels.
in	<i>enableMss</i>	Enable/Disable TCP MSS to be clamped on L2TP interfaces to avoid Segmentation
in	<i>enableMtu</i>	Enable/Disable MTU size to be set on underlying interfaces to avoid fragmentation
in	<i>callback</i>	optional callback to get the response setConfig

Returns

Status of setConfig i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.9.2.4 `virtual telux::common::Status telux::data::net::IL2tpManager::addTunnel ( const L2tp-TunnelConfig & l2tpTunnelConfig, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Set L2TP Configuration for one tunnel

Parameters

in	<i>l2tpTunnelConfig</i>	Configuration to be set telux::net::L2tpTunnelConfig
in	<i>callback</i>	Optional callback to get the response addTunnel

Returns

Status of addTunnel i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.9.2.5 **virtual telux::common::Status telux::data::net::IL2tpManager::requestConfig (L2tpConfigCb l2tpConfigCb) [pure virtual]**

Get Current L2TP Configuration

Parameters

in	<i>l2tpConfigCb</i>	Asynchronous callback to get current L2TP configurations

Returns

Status of requestConfig i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.9.2.6 **virtual telux::common::Status telux::data::net::IL2tpManager::removeTunnel (uint32_t tunnelId, telux::common::ResponseCallback callback = nullptr) [pure virtual]**

Remove L2TP Tunnel

Parameters

in	<i>tunnelId</i>	Tunnel ID to be removed
in	<i>callback</i>	optional callback to get the response removeConfig

Returns

Status of removeTunnel i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.9.2.7 **virtual telux::common::Status telux::data::net::IL2tpManager::registerListener (std::weak_ptr< IL2tpListener > listener) [pure virtual]**

Register L2TP Manager as listener for Data Service health events like data service available or data service not available.

in	<i>listener</i>	pointer of IL2tpListener object that processes the notification

Returns

Status of registerListener success or suitable status code

5.25.1.9.2.8 `virtual telux::common::Status telux::data::net::IL2tpManager::deregisterListener ( std::weak_ptr< IL2tpListener > listener ) [pure virtual]`

Removes a previously added listener.

Parameters

in	<i>listener</i>	pointer of IL2tpListener object that needs to be removed

Returns

Status of deregisterListener success or suitable status code

5.25.1.10 class telux::data::net::IL2tpListener

Interface for L2TP listener object. Client needs to implement this interface to get access to L2TP services notifications like onServiceStatusChange.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

Public member functions

- virtual void [onServiceStatusChange](#) ([telux::common::ServiceStatus](#) status)
- virtual [~IL2tpListener](#) ()

5.25.1.10.1 Constructors and Destructors

5.25.1.10.1.1 `virtual telux::data::net::IL2tpListener::~~IL2tpListener ( ) [virtual]`

Destructor for [IL2tpListener](#)

5.25.1.10.2 Member Function Documentation

5.25.1.10.2.1 `virtual void telux::data::net::IL2tpListener::onServiceStatusChange ( telux::common::ServiceStatus status ) [virtual]`

This function is called when service status changes.

Parameters

<code>in</code>	<code>status</code>	- ServiceStatus

5.25.1.11 struct telux::data::net::NatConfig

Structure represents Network Address Translation (NAT) configuration

Data fields

Type	Field	Description
string	addr	Private IP address
uint16_t	port	Private port
uint16_t	globalPort	Global port
IpProtocol	proto	IP protocol telux::net::IpProtocol

5.25.1.12 class telux::data::net::INatManager

NatManager is a primary interface for configuring static network address translation(SNAT) and DMZ (demilitarized zone). It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready.

Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status addStaticNatEntry](#) (int profileId, const [NatConfig](#) &snatConfig, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status removeStaticNatEntry](#) (int profileId, const [NatConfig](#) &snatConfig, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status requestStaticNatEntries](#) (int profileId, [StaticNatEntriesCb](#) snatEntriesCb)=0
- virtual [telux::common::Status registerListener](#) (std::weak_ptr< [INatListener](#) > listener)=0
- virtual [telux::common::Status deregisterListener](#) (std::weak_ptr< [INatListener](#) > listener)=0
- virtual [telux::data::OperationType getOperationType](#) ()=0
- virtual [~INatManager](#) ()

5.25.1.12.1 Constructors and Destructors**5.25.1.12.1.1 virtual telux::data::net::INatManager::~INatManager () [virtual]**

Destructor for [INatManager](#)

5.25.1.12.2 Member Function Documentation

5.25.1.12.2.1 `virtual bool telux::data::net::INatManager::isSubsystemReady ( ) [pure virtual]`

Checks if the NAT manager subsystem is ready.

Returns

True if NAT Manager is ready for service, otherwise returns false.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.12.2.2 `virtual std::future<bool> telux::data::net::INatManager::onSubsystemReady ( ) [pure virtual]`

Wait for NAT manager subsystem to be ready.

Returns

A future that caller can wait on to be notified when NAT manager is ready.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.12.2.3 `virtual telux::common::Status telux::data::net::INatManager::addStaticNatEntry ( int profileId, const NatConfig & snatConfig, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Adds a static Network Address Translation (NAT) entry in the NAT table, these entries are persistent across object, connection and reboot lifetimes. To remove an entry it needs a explicit call to [removeStaticNatEntry\(\)](#) API, it supports both IPv4 and IPv6

Parameters

in	<i>profileId</i>	Profile identifier to which static entry will be mapped to.
in	<i>snatConfig</i>	snatConfiguration telux::net::NatConfig
in	<i>callback</i>	optional callback to get the response addStaticNatEntry

Returns

Status of addStaticNatEntry i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.12.2.4 `virtual telux::common::Status telux::data::net::INatManager::removeStaticNatEntry ( int profileId, const NatConfig & snatConfig, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Removes a static Network Address Translation (NAT) entry in the NAT table, it supports both IPv4 and IPv6

Parameters

in	<i>profileId</i>	Profile identifier to which static entry will be removed from.
in	<i>snatConfig</i>	snatConfiguration telux::net::NatConfig
in	<i>callback</i>	optional callback to get the response removeStaticNatEntry

Returns

Status of removeStaticNatEntry i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.12.2.5 `virtual telux::common::Status telux::data::net::INatManager::requestStaticNatEntries ( int profileId, StaticNatEntriesCb snatEntriesCb ) [pure virtual]`

Request list of static nat entries available in the NAT table

Parameters

in	<i>profileId</i>	Profile identifier to which static entries will be retrieved.
in	<i>snatEntriesCb</i>	Asynchronous callback to get the list of static Network Address Translation (NAT) entries

Returns

Status of requestStaticNatEntries i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.12.2.6 `virtual telux::common::Status telux::data::net::INatManager::registerListener ( std::weak_ptr< INatListener > listener ) [pure virtual]`

Register Nat Manager as listener for Data Service health events like data service available or data service not available.

Parameters

in	<i>listener</i>	pointer of INatListener object that processes the notification

Returns

Status of registerListener success or suitable status code

5.25.1.12.2.7 `virtual telux::common::Status telux::data::net::INatManager::deregisterListener ( std::weak_ptr< INatListener > listener ) [pure virtual]`

Removes a previously added listener.

Parameters

in	<i>listener</i>	pointer of INatListener object that needs to be removed

Returns

Status of deregisterListener success or suitable status code

5.25.1.12.2.8 `virtual telux::data::OperationType telux::data::net::INatManager::getOperationType ( ) [pure virtual]`

Get the associated operation type for this instance.

Returns

OperationType of getOperationType i.e. LOCAL or REMOTE.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.13 class telux::data::net::INatListener

Interface for Nat listener object. Client needs to implement this interface to get access to Nat services notifications like onServiceStatusChange.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

Public member functions

- virtual void [onServiceStatusChange](#) ([telux::common::ServiceStatus](#) status)
- virtual [~INatListener](#) ()

5.25.1.13.1 Constructors and Destructors

5.25.1.13.1.1 virtual [telux::data::net::INatListener::~INatListener](#) () [[virtual](#)]

Destructor for [INatListener](#)

5.25.1.13.2 Member Function Documentation

5.25.1.13.2.1 virtual void [telux::data::net::INatListener::onServiceStatusChange](#) ([telux::common::ServiceStatus](#) *status*) [[virtual](#)]

This function is called when service status changes.

Parameters

<i>in</i>	<i>status</i>	- ServiceStatus

5.25.1.14 class [telux::data::net::ISocksManager](#)

SocksManager is a primary interface for configuring legacy Socks proxy server. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready.

Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status](#) [enableSocks](#) (bool enable, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [registerListener](#) (std::weak_ptr< [ISocksListener](#) > listener)=0
- virtual [telux::common::Status](#) [deregisterListener](#) (std::weak_ptr< [ISocksListener](#) > listener)=0
- virtual [telux::data::OperationType](#) [getOperationType](#) ()=0
- virtual [~ISocksManager](#) ()

5.25.1.14.1 Constructors and Destructors

5.25.1.14.1.1 virtual [telux::data::net::ISocksManager::~ISocksManager](#) () [[virtual](#)]

Destructor for Socks Manager

5.25.1.14.2 Member Function Documentation

5.25.1.14.2.1 `virtual bool telux::data::net::ISocksManager::isSubsystemReady ( ) [pure virtual]`

Checks if the SocksManager subsystem is ready.

Returns

True if SocksManager is ready for service, otherwise returns false.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.14.2.2 `virtual std::future<bool> telux::data::net::ISocksManager::onSubsystemReady ( ) [pure virtual]`

Wait for SocksManager subsystem to be ready.

Returns

A future that caller can wait on to be notified when SocksManager is ready.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.14.2.3 `virtual telux::common::Status telux::data::net::ISocksManager::enableSocks ( bool enable, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Enable or Disable Socks proxy service.

Parameters

in	<i>enable</i>	true: enable proxy, false: disable proxy
in	<i>callback</i>	optional callback to get the operation error code if any

Returns

Status of proxy enablement i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.14.2.4 **virtual telux::common::Status telux::data::net::ISocksManager::registerListener (std::weak_ptr< ISocksListener > *listener*) [pure virtual]**

Register Socks Manager as listener for Data Service health events like data service available or data service not available.

Parameters

in	<i>listener</i>	pointer of ISocksListener object that processes the notification

Returns

Status of registerListener success or suitable status code

5.25.1.14.2.5 **virtual telux::common::Status telux::data::net::ISocksManager::deregisterListener (std::weak_ptr< ISocksListener > *listener*) [pure virtual]**

Removes a previously added listener.

Parameters

in	<i>listener</i>	pointer of ISocksListener object that needs to be removed

Returns

Status of deregisterListener success or suitable status code

5.25.1.14.2.6 **virtual telux::data::OperationType telux::data::net::ISocksManager::getOperationType () [pure virtual]**

Get the associated operation type for this instance.

Returns

OperationType of getOperationType i.e. LOCAL or REMOTE.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.15 **class telux::data::net::ISocksListener**

Interface for Socks listener object. Client needs to implement this interface to get access to Socks services notifications like onServiceStatusChange.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

Public member functions

- virtual void [onServiceStatusChange](#) ([telux::common::ServiceStatus](#) status)
- virtual [~ISocksListener](#) ()

5.25.1.15.1 Constructors and Destructors

5.25.1.15.1.1 virtual [telux::data::net::ISocksListener::~~ISocksListener](#) () [[virtual](#)]

Destructor for [ISocksListener](#)

5.25.1.15.2 Member Function Documentation

5.25.1.15.2.1 virtual void [telux::data::net::ISocksListener::onServiceStatusChange](#) ([telux::common::ServiceStatus](#) *status*) [[virtual](#)]

This function is called when service status changes.

Parameters

<i>in</i>	<i>status</i>	- ServiceStatus

5.25.1.16 class [telux::data::net::IVlanManager](#)

VlanManager is a primary interface for configuring VLAN (Virtual Local Area Network). it provide APIs for create, query, remove VLAN interfaces and associate or disassociate with profile IDs. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready.

Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status](#) [createVlan](#) (const [VlanConfig](#) &vlanConfig, [CreateVlanCb](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [removeVlan](#) (int16_t vlanId, [InterfaceType](#) ifaceType, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [queryVlanInfo](#) ([QueryVlanResponseCb](#) callback)=0
- virtual [telux::common::Status](#) [bindToBackhaul](#) ([VlanBindConfig](#) vlanBindConfig, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [unbindFromBackhaul](#) ([VlanBindConfig](#) vlanBindConfig, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [queryVlanToBackhaulBindings](#) ([BackhaulType](#) backhaul, [VlanBindingsResponseCb](#) callback)=0

- virtual `telux::common::Status registerListener (std::weak_ptr< IVlanListener > listener)=0`
- virtual `telux::common::Status deregisterListener (std::weak_ptr< IVlanListener > listener)=0`
- virtual `telux::data::OperationType getOperationType ()=0`
- virtual `telux::common::Status bindWithProfile (int profileId, int vlanId, telux::common::ResponseCallback callback)=0`
- virtual `telux::common::Status unbindFromProfile (int profileId, int vlanId, telux::common::ResponseCallback callback)=0`
- virtual `telux::common::Status queryVlanMappingList (VlanMappingResponseCb callback)=0`
- virtual `~IVlanManager ()`

5.25.1.16.1 Constructors and Destructors

5.25.1.16.1.1 `virtual telux::data::net::IVlanManager::~~IVlanManager ( ) [virtual]`

Destructor for `IVlanManager`

5.25.1.16.2 Member Function Documentation

5.25.1.16.2.1 `virtual bool telux::data::net::IVlanManager::isSubsystemReady ( ) [pure virtual]`

Checks if the data subsystem is ready.

Returns

True if VLAN Manager is ready for service, otherwise returns false.

5.25.1.16.2.2 `virtual std::future<bool> telux::data::net::IVlanManager::onSubsystemReady ( ) [pure virtual]`

Wait for data subsystem to be ready.

Returns

A future that caller can wait on to be notified when VLAN manager is ready.

5.25.1.16.2.3 `virtual telux::common::Status telux::data::net::IVlanManager::createVlan ( const VlanConfig & vlanConfig, CreateVlanCb callback = nullptr ) [pure virtual]`

Create a VLAN associated with multiple interfaces

Note

if interface configured as VLAN for the first time, it may trigger auto reboot.

in	<i>vlanConfig</i>	vlan configuration
out	<i>callback</i>	optional callback to get the response createVlan

Returns

Immediate status of [createVlan\(\)](#) request sent i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.16.2.4 `virtual telux::common::Status telux::data::net::IVlanManager::removeVlan ( int16_t vlanId, InterfaceType ifaceType, telux::common::ResponseCallback callback = nullptr ) [pure virtual]`

Remove VLAN configuration

Note

This will delete all clients associated with interface

Parameters

in	<i>vlanId</i>	VLAN ID
in	<i>ifaceType</i>	telux::net::InterfaceType
out	<i>callback</i>	optional callback to get the response removeVlan

Returns

Immediate status of [removeVlan\(\)](#) request sent i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.16.2.5 `virtual telux::common::Status telux::data::net::IVlanManager::queryVlanInfo ( QueryVlanResponseCb callback ) [pure virtual]`

Query information about all the VLANs in the system

Parameters

out	<i>callback</i>	Response callback with list of configured VLANs

Returns

Immediate status of [queryVlanInfo\(\)](#) request sent i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.16.2.6 `virtual telux::common::Status telux::data::net::IVlanManager::bindToBackhaul (VlanBindConfig vlanBindConfig, telux::common::ResponseCallback callback = nullptr) [pure virtual]`

Bind a VLAN with a particular profile id. When a WWAN network interface is brought up using [IDataConnectionManager::startDataCall](#) on that profile id, that interface will be accessible from this VLAN. The behavior of this API is dependent on platform/system configuration. If the platform is configured to allow multiple VLANs to be bound to the same profile id then:

- Binding multiple VLANs to any profile id can be achieved by calling this API with each VLAN id. Each VLAN will be associated with its own bridge.
- Reboot is not triggered with any bind operation. If the platform is not configured to allow multiple VLANs to be bound to the same profile id then:
- Binding VLAN to default profile id will associate it with bridge0 and trigger automatic reboot.
- Binding VLAN to any other profile id will associate it with own bridge.
- Multiple VLAN binding attempt to any profile id will result in error `telux::common::ErrorCode::INVALID_OPERATION`. This setting will be persistent across multiple boots.

Parameters

in	<i>vlanBindConfig</i>	Backhaul information and vlan id to bind it to. telux::data::net::VlanBindConfig
out	<i>callback</i>	Callback to get the response of bindToBackhaul API

Returns

Immediate status of [bindToBackhaul\(\)](#) request sent i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.16.2.7 `virtual telux::common::Status telux::data::net::IVlanManager::unbindFromBackhaul (VlanBindConfig vlanBindConfig, telux::common::ResponseCallback callback = nullptr) [pure virtual]`

Unbind VLAN from particular backhaul. This API will stop vlan traffic flow to/from specified backhaul type. Slot ID and profile ID are relevant only for WWAN backhaul. For all other backhaul types, values are don't care.

in	<i>vlanBindConfig</i>	Backhaul information and vlan id to unbind it from. telux::data::net::VlanBindConfig
in	<i>callback</i>	Callback to get the response of unbindFromBackhaul API

Returns

Immediate status of [unbindFromBackhaul\(\)](#) request sent i.e. success or suitable status code

Note

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

5.25.1.16.2.8 **virtual** **telux::common::Status** **telux::data::net::IVlanManager::queryVlanToBackhaul-
Bindings (BackhaulType *backhaul*, VlanBindingsResponseCb *callback*) [pure
virtual]**

Query VLAN to backhaul binding configurations

Parameters

in	<i>backhaul</i>	Backhaul to query vlan binding for.
in	<i>callback</i>	callback to get the response of queryVlanToBackhaulBindings API

Returns

Immediate status of [queryVlanToBackhaulBindings\(\)](#) request sent i.e. success or suitable status code

Note

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

5.25.1.16.2.9 **virtual** **telux::common::Status** **telux::data::net::IVlanManager::registerListener (**
std::weak_ptr< IVlanListener > *listener*) [pure virtual]

Register Vlan Manager as a listener for Data Service health events like data service available or data service not available.

Parameters

in	<i>listener</i>	pointer of IVlanListener object that processes the notification

Returns

Status of registerListener success or suitable status code

5.25.1.16.2.10 `virtual telux::common::Status telux::data::net::IVlanManager::deregisterListener ( std::weak_ptr< IVlanListener > listener ) [pure virtual]`

Removes a previously added listener.

Parameters

in	<i>listener</i>	pointer of IVlanListener object that needs to be removed

Returns

Status of deregisterListener success or suitable status code

5.25.1.16.2.11 `virtual telux::data::OperationType telux::data::net::IVlanManager::getOperationType ( ) [pure virtual]`

Get the associated operation type for this instance.

Returns

OperationType of getOperationType i.e. LOCAL or REMOTE.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.25.1.16.2.12 `virtual telux::common::Status telux::data::net::IVlanManager::bindWithProfile ( int profileId, int vlanId, telux::common::ResponseCallback callback ) [pure virtual]`

Bind a Vlan with a particular profile ID. When a WWAN network interface is brought up using [IDataConnectionManager::startDataCall](#) on that profile ID, that interface will be accessible from this Vlan

Parameters

in	<i>profileId</i>	profile id for vlan association
in	<i>vlanId</i>	sets vlan id
out	<i>callback</i>	callback to get the response of associateWithProfileId API

Returns

Immediate status of associateWithProfileId() request sent i.e. success or suitable status code.

Deprecated Use [bindToBackhaul\(\)](#) API below to bind VLAN to backhaul

5.25.1.16.2.13 `virtual telux::common::Status telux::data::net::IVlanManager::unbindFromProfile ( int profileId, int vlanId, telux::common::ResponseCallback callback ) [pure virtual]`

Unbind VLAN id with given profile id

Parameters

in	<i>profileId</i>	profile id for vlan association
in	<i>vlanId</i>	vlan id
in	<i>callback</i>	callback to get the response of <code>associateWithProfileId</code> API

Returns

Immediate status of `disassociateFromProfileId()` request sent i.e. success or suitable status code

Deprecated Use `unbindFromBackhaul()` API below to unbind VLAN to backhaul

5.25.1.16.2.14 `virtual telux::common::Status telux::data::net::IVlanManager::queryVlanMappingList ( VlanMappingResponseCb callback ) [pure virtual]`

Query VLAN mapping list with associated profile id and vlan id

Parameters

in	<i>callback</i>	callback to get the response of <code>queryVlanMappingList</code> API

Returns

Immediate status of `queryVlanMappingList()` request sent i.e. success or suitable status code

Deprecated Use `queryVlanToBackhaulBindings()` API below to request VLAN to backhaul mapping

5.25.1.17 class telux::data::net::IVlanListener

Interface for Vlan listener object. Client needs to implement this interface to get access to Socks services notifications like `onServiceStatusChange`.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

Public member functions

- virtual void `onServiceStatusChange` (telux::common::ServiceStatus status)
- virtual void `onHwAccelerationChanged` (const ServiceState state)
- virtual `~IVlanListener` ()

5.25.1.17.1 Constructors and Destructors

5.25.1.17.1.1 `virtual telux::data::net::IVlanListener::~IVlanListener ( ) [virtual]`

Destructor for [IVlanListener](#)

5.25.1.17.2 Member Function Documentation

5.25.1.17.2.1 `virtual void telux::data::net::IVlanListener::onServiceStatusChange ( telux::common::ServiceStatus status ) [virtual]`

This function is called when service status changes.

Parameters

<i>in</i>	<i>status</i>	- ServiceStatus

5.25.1.17.2.2 `virtual void telux::data::net::IVlanListener::onHwAccelerationChanged ( const ServiceState state ) [virtual]`

This function is called when there is a change in IPA Connection Manager daemon state.

Parameters

<i>in</i>	<i>state</i>	New state of IPA connection Manager daemon Active/Inactive

5.25.2 Enumeration Type Documentation

5.25.2.1 `enum telux::data::net::L2tpProtocol`

L2TP encapsulation protocols

Enumerator

NONE

IP IP Protocol used for encapsulation

UDP UDP Protocol used for encapsulation

5.26 Telematics_platform_deviceinfo

5.26.1 Data Structure Documentation

5.26.1.1 class telux::platform::IDeviceInfoListener

Listener class for getting device info related notifications . The client needs to implement these methods as briefly as possible and avoid blocking calls in it. The methods in this class can be invoked from multiple different threads. Client needs to make sure that the implementation is thread-safe.

Public member functions

- virtual [~IDeviceInfoListener](#) ()

5.26.1.1.1 Constructors and Destructors

5.26.1.1.1.1 virtual telux::platform::IDeviceInfoListener::~~IDeviceInfoListener () [virtual]

Destructor of [IDeviceInfoListener](#)

5.26.1.2 struct telux::platform::PlatformVersion

Structure contains the version of the platform software

Data fields

Type	Field	Description
string	meta	Meta Version, for example: SA2150P_SA515M.LE_LE.1-3_2-1-00297-STD.INT-1
string	modem	Modem Version, for example: MPSS.HI.3.1.c3-00114-SDX55_GENAUTO_TEST-1
string	externalApp	External App Version, for example: LE.UM.3.2.3-72102-SA2150p.Int-1
string	integratedApp	Integrated App MDM Version, for example: LE.UM.4.1.1-71802-sa515m.Int-1

5.26.1.3 class telux::platform::IDeviceInfoManager

[IDeviceInfoManager](#) provides interface to to retrieve IMEI and platform version operations.

Public member functions

- virtual [telux::common::ServiceStatus](#) getServiceStatus ()=0
- virtual [telux::common::Status](#) registerListener (std::weak_ptr< [IDeviceInfoListener](#) > listener)=0

- virtual [telux::common::Status deregisterListener](#) (std::weak_ptr< [IDeviceInfoListener](#) > listener)=0
- virtual [telux::common::Status getPlatformVersion](#) ([PlatformVersion](#) &pv)=0
- virtual [telux::common::Status getIMEI](#) (std::string &imei)=0
- virtual [~IDeviceInfoManager](#) ()

5.26.1.3.1 Constructors and Destructors

5.26.1.3.1.1 virtual [telux::platform::IDeviceInfoManager::~IDeviceInfoManager](#) () [virtual]

Destructor of [IDeviceInfoManager](#)

5.26.1.3.2 Member Function Documentation

5.26.1.3.2.1 virtual [telux::common::ServiceStatus](#) [telux::platform::IDeviceInfoManager::getServiceStatus](#) () [pure virtual]

This status indicates whether the object is in a usable state.

Returns

[telux::common::ServiceStatus](#) indicating the current status of the device info service.

5.26.1.3.2.2 virtual [telux::common::Status](#) [telux::platform::IDeviceInfoManager::registerListener](#) (std::weak_ptr< [IDeviceInfoListener](#) > *listener*) [pure virtual]

Registers the listener for FileSystem Manager indications.

Parameters

in	<i>listener</i>	- pointer to implemented listener.

Returns

status of the registration request.

5.26.1.3.2.3 virtual [telux::common::Status](#) [telux::platform::IDeviceInfoManager::deregisterListener](#) (std::weak_ptr< [IDeviceInfoListener](#) > *listener*) [pure virtual]

Deregisters the previously registered listener.

Parameters

in	<i>listener</i>	- pointer to registered listener that needs to be removed.

Returns

status of the deregistration request.

5.26.1.3.2.4 **virtual telux::common::Status telux::platform::DeviceInfoManager::getPlatformVersion (PlatformVersion & pv) [pure virtual]**

Get the platform version. Need obtain required permissions from telux_allow_version.

Parameters

out	<i>pv</i>	- telux::platform::PlatformVersion

Returns

- [telux::common::Status](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.26.1.3.2.5 **virtual telux::common::Status telux::platform::DeviceInfoManager::getIMEI (std::string & imei) [pure virtual]**

Get the international mobile equipment identity.

Parameters

out	<i>imei</i>	- std::string

Returns

- [telux::common::Status](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.27 Telematics_platform

5.27.1 Data Structure Documentation

5.27.1.1 class telux::platform::PlatformFactory

[PlatformFactory](#) allows creation of Platform services related classes.

Public member functions

- virtual std::shared_ptr
< [IDeviceInfoManager](#) > [getDeviceInfoManager](#) ([telux::common::InitResponseCb](#)
callback=nullptr)=0

Static Public Member Functions

- static [PlatformFactory](#) & [getInstance](#) ()

Protected Member Functions

- [PlatformFactory](#) ()
- virtual [~PlatformFactory](#) ()

5.27.1.1.1 Constructors and Destructors

5.27.1.1.1.1 [telux::platform::PlatformFactory::PlatformFactory](#) () [[protected](#)]

5.27.1.1.1.2 [virtual telux::platform::PlatformFactory::~~PlatformFactory](#) () [[protected](#)],
[[virtual](#)]

5.27.1.1.2 Member Function Documentation

5.27.1.1.2.1 [static PlatformFactory& telux::platform::PlatformFactory::getInstance](#) () [[static](#)]

Get instance of platform Factory

5.27.1.1.2.2 [virtual std::shared_ptr<IDeviceInfoManager> telux::platform::PlatformFactory::get-](#)
[DeviceInfoManager](#) ([telux::common::InitResponseCb](#) *callback = nullptr*) [[pure](#)
[virtual](#)]

Get instance of device info manager ([IDeviceInfoManager](#)). The device info manager supports device info request like retrieving IMEI and platform version.

Parameters

in	<i>callback</i>	Optional callback to get the initialization status of FsManager. telux::common::InitResponseCb

Returns

pointer of [IDeviceInfoManager](#) object.

5.28 Telematics_cellbroadcast

5.28.1 Data Structure Documentation

5.28.1.1 struct telux::tel::CellBroadcastFilter

Defines cellbroadcast message filter. Refer spec 3GPP TS 23.041 9.4.1.2.2 for message identifier. Eg: If user want to receive from 0x1112 to 0x1116 then, startMessageId is 0x1112 and endMessageId is 0x1116. If user want to receive only 0x1112, then both startMessageId and endMessageId is 0x1112.

Data fields

Type	Field	Description
int	startMessageId	Intended to receive start from which MessageType
int	endMessageId	Intended to receive upto which MessageType

5.28.2 Enumeration Type Documentation

5.28.2.1 enum telux::tel::GeographicalScope

Defines geographical scope of cell broadcast.

Enumerator

CELL_WIDE_IMMEDIATE Cell wide geographical scope with immediate display
PLMN_WIDE PLMN wide geographical scope
LA_WIDE
CELL_WIDE Location / Service/ Tracking area wide geographical scope
 (GSM/UMTS/E-UTRAN/NG-RAN). Cell wide geographical scope

5.28.2.2 enum telux::tel::MessagePriority

Defines priority for cell broadcast message.

Enumerator

UNKNOWN Unknown message priority
NORMAL Normal message priority
EMERGENCY Emergency message priority

5.28.2.3 enum telux::tel::MessageType

Defines message type for cell broadcast message.

Enumerator

UNKNOWN Unknown message type
ETWS Earthquake and Tsunami Warning System
CMAS Commercial Mobile Alert System

5.28.2.4 enum telux::tel::EtwsWarningType

Defines warning type for ETWS cell broadcast message.

Enumerator

UNKNOWN Unknown ETWS warning type
EARTHQUAKE ETWS warning type for earthquake
TSUNAMI ETWS warning type for tsunami
EARTHQUAKE_AND_TSUNAMI ETWS warning type for earthquake and tsunami
TEST_MESSAGE ETWS warning type for test messages
OTHER_EMERGENCY ETWS warning type for other emergency types

5.28.2.5 enum telux::tel::CmasMessageClass

Defines message class for CMAS cell broadcast message.

Enumerator

UNKNOWN CMAS category for warning types that
are reserved for future extension
PRESIDENTIAL_LEVEL_ALERT Presidential-level alert (Korean Public Alert System Class 0
message)
EXTREME_THREAT Extreme threat to life and property (Korean Public Alert System Class 1
message)
SEVERE_THREAT Severe threat to life and property
(Korean Public Alert System Class 1 message).
CHILD_ABDUCTION_EMERGENCY Child abduction emergency (AMBER Alert)
REQUIRED_MONTHLY_TEST CMAS test message
CMAS_EXERCISE CMAS exercise
OPERATOR_DEFINED_USE CMAS category for operator defined use

5.28.2.6 enum telux::tel::CmasSeverity

Defines severity type for CMAS cell broadcast message.

Enumerator

UNKNOWN CMAS alert severity is unknown. The severity is available for all GSM/UMTS alerts
except for the Presidential-level alert class (Korean Public Alert System Class 0).
EXTREME Extraordinary threat to life or property
SEVERE Significant threat to life or property

5.28.2.7 enum telux::tel::CmasUrgency

Defines urgency type for CMAS cell broadcast message.

Enumerator

UNKNOWN CMAS alert urgency is unknown. The urgency is available for all GSM/UMTS alerts except for the Presidential-level alert class (Korean Public Alert System Class 0).

IMMEDIATE Responsive action should be taken immediately

EXPECTED Responsive action should be taken within the next hour

5.28.2.8 enum telux::tel::CmasCertainty

Defines certainty type for CMAS cell broadcast message.

Enumerator

UNKNOWN CMAS alert certainty is unknown. The certainty is available for all GSM/UMTS alerts except for the Presidential-level alert class (Korean Public Alert System Class 0).

OBSERVED Determined to have occurred or to be ongoing.

LIKELY Likely (probability > ~50%)

5.28.2.9 enum telux::tel::GeometryType

Defines geometry type specified in wireless emergency alert.

Enumerator

UNKNOWN Unknown geometry type

POLYGON Polygon geometry type

CIRCLE Circle geometry type

5.29 Telematics_ecall

5.29.1 Data Structure Documentation

5.29.1.1 class telux::tel::IEcallManager

[IEcallManager](#) allows operations related to automotive emergency call management and its related configurations.

Public member functions

- virtual [telux::common::ServiceStatus](#) `getServiceStatus ()`=0
- virtual [telux::common::Status](#) `setConfig (EcallConfig config)`=0
- virtual [telux::common::Status](#) `getConfig (EcallConfig &config)`=0
- virtual [telux::common::Status](#) `registerListener (std::weak_ptr< IEcallListener > listener)`=0
- virtual [telux::common::Status](#) `deregisterListener (std::weak_ptr< IEcallListener > listener)`=0
- virtual [~IEcallManager \(\)](#)

5.29.1.1.1 Constructors and Destructors

5.29.1.1.1.1 virtual [telux::tel::IEcallManager::~~IEcallManager \(\)](#) [[virtual](#)]

5.29.1.1.2 Member Function Documentation

5.29.1.1.2.1 virtual [telux::common::ServiceStatus](#) [telux::tel::IEcallManager::getServiceStatus \(\)](#) [[pure virtual](#)]

Checks the status of [IEcallManager](#) sub-system and returns the result.

Returns

the status of [IEcallManager](#) sub-system status [telux::common::ServiceStatus](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.29.1.1.2.2 virtual [telux::common::Status](#) [telux::tel::IEcallManager::setConfig \(EcallConfig config \)](#) [[pure virtual](#)]

Set the configuration related to emergency call. The configuration is persistent and takes effect when the next emergency call is dialed.

<i>in</i>	<i>config</i>	eCall configuration to be set EcallConfig

Returns

Status of setConfig i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.29.1.1.2.3 virtual telux::common::Status telux::tel::IEcallManager::getConfig (EcallConfig & *config*) [pure virtual]

Get the configuration related to emergency call.

Parameters

<i>out</i>	<i>config</i>	Parameter to hold the fetched eCall configuration EcallConfig

Returns

Status of getConfig i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.29.1.1.2.4 virtual telux::common::Status telux::tel::IEcallManager::registerListener (std::weak_ptr< IEcallListener > *listener*) [pure virtual]

Register a listener for notifications from the EcallManager.

Parameters

<i>in</i>	<i>listener</i>	Pointer to IEcallListener object that processes the notification

Returns

Status of registerListener i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.29.1.1.2.5 `virtual telux::common::Status telux::tel::IEcallManager::deregisterListener ( std::weak_ptr< IEcallListener > listener ) [pure virtual]`

Deregister a previously registered listener.

Parameters

in	<i>listener</i>	Pointer to IEcallListener object that needs to be deregistered.

Returns

Status of deregisterListener i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.29.1.2 class telux::tel::IEcallListener

Listener class to notify service status change notifications. The listener method can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe.

Public member functions

- virtual [~IEcallListener](#) ()

5.29.1.2.1 Constructors and Destructors

5.29.1.2.1.1 `virtual telux::tel::IEcallListener::~~IEcallListener ( ) [virtual]`

Destructor of [IEcallListener](#)

5.30 Telematics_ims_settings

5.30.1 Data Structure Documentation

5.30.1.1 struct telux::tel::ImsServiceConfig

Defines the selected IMS service configuration parameters and their corresponding value

Data fields

Type	Field	Description
ImsService-ConfigValidity	configValidity-Mask	Indicates the configurations type. Bit set to 1 denotes the config is valid.
bool	imsService-Enabled	Enable/Disable IMS service
bool	voImsEnabled	Enable/Disable VOIMS service

5.30.1.2 class telux::tel::ImsSettingsManager

ImsSettingsManager allows IMS settings. For example enabling or disabling IMS service, VOIMS service.

Public member functions

- virtual [telux::common::ServiceStatus](#) getServiceStatus ()=0
- virtual [telux::common::Status](#) requestServiceConfig (SlotId slotId, [ImsServiceConfigCb](#) callback)=0
- virtual [telux::common::Status](#) setServiceConfig (SlotId slotId, [ImsServiceConfig](#) config, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) registerListener (std::weak_ptr< [ImsSettingsListener](#) > listener)=0
- virtual [telux::common::Status](#) deregisterListener (std::weak_ptr< [ImsSettingsListener](#) > listener)=0
- virtual [~ImsSettingsManager](#) ()

5.30.1.2.1 Constructors and Destructors

5.30.1.2.1.1 virtual [telux::tel::ImsSettingsManager::~ImsSettingsManager](#) () [[virtual](#)]

5.30.1.2.2 Member Function Documentation

5.30.1.2.2.1 `virtual telux::common::ServiceStatus telux::tel::IImSettingsManager::getServiceStatus ( ) [pure virtual]`

This status indicates whether the [IImSettingsManager](#) object is in a usable state.

Returns

SERVICE_AVAILABLE - If IMS settings manager is ready for service. SERVICE_UNAVAILABLE - If IMS settings manager is temporarily unavailable. SERVICE_FAILED - If IMS settings manager encountered an irrecoverable failure.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.30.1.2.2.2 `virtual telux::common::Status telux::tel::IImSettingsManager::requestServiceConfig ( SlotId slotId, ImsServiceConfigCb callback ) [pure virtual]`

Request the IMS service configurations

Parameters

in	<i>slotId</i>	Slot for which the IMS service configurations is requested.
in	<i>callback</i>	Callback function to get the response of request IMS service configurations.

Returns

Status of requestServiceConfig i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.30.1.2.2.3 `virtual telux::common::Status telux::tel::IImSettingsManager::setServiceConfig ( SlotId slotId, ImsServiceConfig config, common::ResponseCallback callback = nullptr ) [pure virtual]`

To configure the IMS service configurations. Also specify whether configuration needs to be enabled or disabled.

Parameters

in	<i>slotId</i>	Slot for which the IMS service configuration is intended.
in	<i>config</i>	Indicates which configuration are configured currently and whether the config is enabled or disabled. ImsServiceConfig .

<i>in</i>	<i>callback</i>	Callback function to get the response of set IMS service configuration request.

Returns

Status of setServiceConfig i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.30.1.2.2.4 `virtual telux::common::Status telux::tel::IImSettingsManager::registerListener (std::weak_ptr< IImSettingsListener > listener ) [pure virtual]`

Register a listener for specific events in the IMS settings subsystem.

Parameters

<i>in</i>	<i>listener</i>	Pointer to IImSettingsListener object that processes the notification

Returns

Status of registerListener i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.30.1.2.2.5 `virtual telux::common::Status telux::tel::IImSettingsManager::deregisterListener (std::weak_ptr< IImSettingsListener > listener ) [pure virtual]`

Deregister the previously added listener.

Parameters

<i>in</i>	<i>listener</i>	Pointer to IImSettingsListener object that needs to be deregistered.

Returns

Status of deregisterListener i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.30.1.3 class telux::tel::ImsSettingsListener

Listener class for getting IMS service configuration change notifications. The listener method can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe.

Public member functions

- virtual void [onImsServiceConfigsChange](#) (SlotId slotId, [ImsServiceConfig](#) config)
- virtual void [onServiceStatusChange](#) (telux::common::ServiceStatus status)
- virtual [~ImsSettingsListener](#) ()

5.30.1.3.1 Constructors and Destructors

5.30.1.3.1.1 virtual telux::tel::ImsSettingsListener::~ImsSettingsListener () [virtual]

Destructor of [ImsSettingsListener](#)

5.30.1.3.2 Member Function Documentation

5.30.1.3.2.1 virtual void telux::tel::ImsSettingsListener::onImsServiceConfigsChange (SlotId *slotId*, [ImsServiceConfig](#) *config*) [virtual]

This function is called whenever any IMS service configuration is changed.

Parameters

in	<i>slotId</i>	SIM corresponding to slot identifier for which the IMS service configuration has changed.
in	<i>config</i>	Indicates which configuration is valid and whether the config is enabled or disabled. ImsServiceConfig .

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.30.1.3.2.2 virtual void telux::tel::ImsSettingsListener::onServiceStatusChange (telux::common::ServiceStatus *status*) [virtual]

This function is called when [ImsSettingsManager](#) service status changes.

in	status	- ServiceStatus

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.30.2 Enumeration Type Documentation

5.30.2.1 enum telux::tel::ImsServiceConfigType

Defines the IMS service configuration parameters

Enumerator

IMSSETTINGS_VOIMS Voice calling support on LTE

IMSSETTINGS_IMS_SERVICE IMS Normal Registration configuration

5.31 Telematics_multi_sim

5.31.1 Data Structure Documentation

5.31.1.1 struct telux::tel::SlotStatus

Represents status of a physical SIM slot

Data fields

Type	Field	Description
SlotState	slotState	State of the physical SIM slot
CardState	cardState	Status of the card in the physical slot
CardError	cardError	Indicates the reason for the card error, and is valid only when the card state is CARDSTATE_ERROR.

5.31.1.2 class telux::tel::IMultiSimManager

MultiSimManager allows to perform operation pertaining to devices which have more than one SIM/UICC card. Clients should check if the subsystem is ready before invoking any of the APIs as follows.

```
bool isReady = MultiSimManager->isSubsystemReady();
```

Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status](#) [getSlotCount](#) (int &count)=0
- virtual [telux::common::Status](#) [requestHighCapability](#) ([HighCapabilityCallback](#) callback)=0
- virtual [telux::common::Status](#) [setHighCapability](#) (int slotId, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [switchActiveSlot](#) (SlotId slotId, [common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [requestSlotStatus](#) ([SlotStatusCallback](#) callback)=0
- virtual [telux::common::Status](#) [registerListener](#) (std::weak_ptr< [IMultiSimListener](#) > listener)=0
- virtual [telux::common::Status](#) [deregisterListener](#) (std::weak_ptr< [IMultiSimListener](#) > listener)=0
- virtual [~IMultiSimManager](#) ()

5.31.1.2.1 Constructors and Destructors

5.31.1.2.1.1 `virtual telux::tel::IMultiSimManager::~IMultiSimManager ( ) [virtual]`

5.31.1.2.2 Member Function Documentation

5.31.1.2.2.1 `virtual bool telux::tel::IMultiSimManager::isSubsystemReady ( ) [pure virtual]`

Checks the status of Multi SIM subsystem and returns the result.

Returns

If true MultiSimManager is ready.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.31.1.2.2.2 `virtual std::future<bool> telux::tel::IMultiSimManager::onSubsystemReady ( ) [pure virtual]`

Wait for Multi SIM subsystem to be ready.

Returns

A future that caller can wait on to be notified when Multi SIM subsystem is ready.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.31.1.2.2.3 `virtual telux::common::Status telux::tel::IMultiSimManager::getSlotCount ( int & count ) [pure virtual]`

Get SIM slot count. The count can be used to determine whether the device supports multi SIM.

Parameters

out	<i>count</i>	Slot count.

Returns

Status of getSlotCount i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.31.1.2.2.4 **virtual telux::common::Status telux::tel::IMultiSimManager::requestHighCapability (HighCapabilityCallback *callback*) [pure virtual]**

Request to find out which SIM/slot is allowed to use advance Radio Technology like 5G at a time. For example SIM/slot with high capability may allowed to use RAT capabilities like 5G/4G/3G/2G while the SIM/slot with low capability may be allowed to use RAT capabilities like 4G/2G.

Parameters

in	<i>callback</i>	Callback function to get the response of request high capability.

Returns

Status of requestHighCapability i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.31.1.2.2.5 **virtual telux::common::Status telux::tel::IMultiSimManager::setHighCapability (int *slotId*, common::ResponseCallback *callback* = nullptr) [pure virtual]**

Set SIM/slot with high capability asynchronously. On dual SIM devices, only one SIM may be allowed to use advanced Radio technology like 5G at a time. This API sets the SIM/slot that should be allowed the highest RAT capability. The other SIM/slot will be given lower RAT capabilities. For example, SIM in slot1 will be allowed 2G/3G/4G/5G and the SIM in slot2 will be allowed only 2G/4G.

Parameters

in	<i>slotId</i>	Slot set with high capability.
in	<i>callback</i>	Callback function to get the response of set high capability request.

Returns

Status of setHighCapability i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.31.1.2.2.6 **virtual telux::common::Status telux::tel::IMultiSimManager::switchActiveSlot (SlotId *slotId*, common::ResponseCallback *callback* = nullptr) [pure virtual]**

Choose the physical SIM slot to be used by modem on Single-SIM TCU platforms. After switching the slot, only the SIM on chosen physical slot can be used for WWAN functionality.

in	<i>slotId</i>	physical slot to be made active
in	<i>callback</i>	Callback function to get the response of slot switch request

Returns

Status of switchActiveSlot i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.31.1.2.2.7 **virtual telux::common::Status telux::tel::IMultiSimManager::requestSlotStatus (SlotStatusCallback *callback*) [pure virtual]**

Request the status of physical slots.

Parameters

in	<i>callback</i>	Callback function to get the response of slot status request

Returns

Status of requestSlotStatus i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.31.1.2.2.8 **virtual telux::common::Status telux::tel::IMultiSimManager::registerListener (std::weak_ptr< IMultiSimListener > *listener*) [pure virtual]**

Register a listener for specific events in the Multi SIM subsystem.

Parameters

in	<i>listener</i>	Pointer to IMultiSimListener object that processes the notification

Returns

Status of registerListener i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.31.1.2.2.9 `virtual telux::common::Status telux::tel::IMultiSimManager::deregisterListener ( std::weak_ptr< IMultiSimListener > listener ) [pure virtual]`

Deregister the previously added listener.

Parameters

in	<i>listener</i>	Pointer to IMultiSimListener object that needs to be deregistered.

Returns

Status of deregisterListener i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

5.31.1.3 class telux::tel::IMultiSimListener

Listener class for getting high capability change notification. The listener method can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe.

Public member functions

- virtual void [onHighCapabilityChanged](#) (int slotId)
- virtual void [onSlotStatusChanged](#) (std::map< SlotId, [SlotStatus](#) > slotStatus)
- virtual [~IMultiSimListener](#) ()

5.31.1.3.1 Constructors and Destructors

5.31.1.3.1.1 `virtual telux::tel::IMultiSimListener::~~IMultiSimListener ( ) [virtual]`

Destructor of [IMultiSimListener](#)

5.31.1.3.2 Member Function Documentation

5.31.1.3.2.1 `virtual void telux::tel::IMultiSimListener::onHighCapabilityChanged ( int slotId ) [virtual]`

This function is called whenever there is change in high capability for SIM/slot.

Parameters

in	<i>slotId</i>	SIM corresponding to slot identifier has high capability now.

Note

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

5.31.1.3.2 **virtual void telux::tel::IMultiSimListener::onSlotStatusChanged (std::map< SlotId, SlotStatus > *slotStatus*) [virtual]**

This function is called whenever there is change in physical SIM slots status.

Parameters

in	<i>slotStatus</i>	list of slots status SlotStatus

Note

Eval: This is a new API and is being evaluated.It is subject to change and could break backwards compatibility.

5.31.2 Enumeration Type Documentation

5.31.2.1 enum telux::tel::SlotState

Represents state of the physical SIM slot

Enumerator

- UNKNOWN**
- INACTIVE** Slot is inactive
- ACTIVE** Slot is active

6 Namespace Documentation

6.1 telux Namespace Reference

DeviceConfig provides utility functions to get device configuration details such as multi SIM support.

Namespaces

- namespace [audio](#)
- namespace [common](#)
- namespace [config](#)
- namespace [cv2x](#)
- namespace [data](#)
- namespace [loc](#)
- namespace [platform](#)
- namespace [power](#)
- namespace [tel](#)
- namespace [therm](#)
- namespace [wlan](#)

DeviceConfig provides utility functions to get device configuration details such as multi SIM support.

6.2 telux::audio Namespace Reference

Data Structures

- struct [FormatParams](#)
- struct [Amrwbparams](#)
- struct [StreamConfig](#)
- struct [FormatInfo](#)
- struct [ChannelVolume](#)
- struct [StreamVolume](#)
- struct [StreamMute](#)

- struct [StreamBuffer](#)
- struct [DtmfTone](#)
- class [AudioFactory](#)

[AudioFactory](#) allows creation of audio manager. [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...](#)

- class [IPlayListener](#)
- class [ITranscodeListener](#)
- class [IAudioListener](#)
- 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 [IStreamBuffer](#)
- 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 [IAudioDevice](#)

Audio device and it's characteristics like Direction (Sink or Source), type. [More...](#)

- class [IAudioStream](#)

[IAudioStream](#) represents single audio stream with base properties. [More...](#)

- class [IAudioVoiceStream](#)

[IAudioVoiceStream](#) represents single voice stream. [More...](#)

- class [IAudioPlayStream](#)

[IAudioPlayStream](#) represents single audio playback stream. [More...](#)

- class [IAudioCaptureStream](#)

[IAudioCaptureStream](#) represents single audio capture stream. [More...](#)

- class [IAudioLoopbackStream](#)

[IAudioLoopbackStream](#) represents audio loopback stream. [More...](#)

- class [IAudioToneGeneratorStream](#)

[IAudioToneGeneratorStream](#) represents tone generator stream. [More...](#)

- class [ITranscoder](#)

[ITranscoder](#) is used to convert one audio format to another audio format using the transcoding operation.
[More...](#)

6.2.1 Typedef Documentation

6.2.1.1 using telux::audio::ChannelTypeMask = typedef int

Represent Stream's consolidated lists of Channel presence

6.2.1.2 using telux::audio::GetDevicesResponseCb = typedef std::function<void(std::vector<std::shared_ptr<IAudioDevice>> devices, telux::common::ErrorCode error)>

This function is called with the response to getDevices API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

in	<i>devices</i>	Devices list.
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. ErrorCode

6.2.1.3 using telux::audio::GetStreamTypesResponseCb = typedef std::function<void(std::vector<StreamType> streamTypes, telux::common::ErrorCode error)>

This function is called with the response to getStreamTypes API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

in	<i>streamTypes</i>	Stream type list.
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. ErrorCode

6.2.1.4 using telux::audio::CreateStreamResponseCb = typedef std::function<void(std::shared_ptr<IAudioStream> &stream, telux::common::ErrorCode error)>

This function is called with the response to createStream API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

in	<i>stream</i>	Interface pointer of Stream created. VOICE_CALL - Provides IAudioVoiceStream pointer PLAY - Provides IAudioPlayStream pointer CAPTURE - Provides IAudioCaptureStream pointer

<i>in</i>	<i>error</i>	Return code which indicates whether the operation succeeded or not. ErrorCode

6.2.1.5 **using telux::audio::CreateTranscoderResponseCb = typedef std::function<void(std::shared_ptr<ITranscoder> &transcoder, telux::common::ErrorCode error)>**

This function is called with the response to createTranscoder API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

<i>in</i>	<i>transcoder</i>	Interface pointer of transcoder.
<i>in</i>	<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.

6.2.1.6 **using telux::audio::DeleteStreamResponseCb = typedef std::function<void(telux::common::ErrorCode error)>**

This function is called with the response to deleteStream API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

<i>in</i>	<i>error</i>	Return code which indicates whether the operation succeeded or not. ErrorCode

6.2.1.7 **using telux::audio::GetStreamDeviceResponseCb = typedef std::function<void(std::vector<DeviceType> devices, telux::common::ErrorCode error)>**

This function is called with the response to stream getDevice API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

<i>in</i>	<i>devices</i>	Devices list.
<i>in</i>	<i>error</i>	Return code which indicates whether the operation succeeded or not. ErrorCode

6.2.1.8 using telux::audio::GetStreamVolumeResponseCb = typedef std::function<void(StreamVolume volume, telux::common::ErrorCode error)>

This function is called with the response to stream getVolume API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

in	<i>volume</i>	stream volume details.
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. ErrorCode

6.2.1.9 using telux::audio::GetStreamMuteResponseCb = typedef std::function<void(StreamMute mute, telux::common::ErrorCode error)>

This function is called with the response to stream getMute API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

in	<i>mute</i>	stream mute details.
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. ErrorCode

6.2.1.10 using telux::audio::WriteResponseCb = typedef std::function<void(std::shared_ptr<IStreamBuffer> buffer, uint32_t bytesWritten, telux::common::ErrorCode error)>

This function is called with the response to [IAudioPlayStream::write\(\)](#).

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

in	<i>buffer</i>	Buffer that was used for the write operation. Application could call IStreamBuffer::reset() and reuse this buffer for subsequent write operations on the same stream.
in	<i>bytesWritten</i>	Return how many bytes are written to the stream.
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. ErrorCode

6.2.1.11 using telux::audio::ReadResponseCb = typedef std::function<void(std::shared_ptr<IStreamBuffer> buffer, telux::common::ErrorCode error)>

This function is called with the response to [IAudioCaptureStream::read\(\)](#).

The callback can be invoked from multiple different threads. The implementation should be thread safe.

in	<i>buffer</i>	Buffer that was used to capture the data from the read operation. Applications could call IStreamBuffer::reset() and reuse this buffer for subsequent read operations on the same stream. Also <code>buffer.getDataSize()</code> will represent the number of bytes read.
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. <code>ErrorCode</code>

6.2.1.12 **using telux::audio::TranscoderReadResponseCb = typedef std- ::function<void(std::shared_ptr<IAudioBuffer> buffer, uint32_t isLastBuffer, telux::common::ErrorCode error)>**

This function is called with the response to [ITranscoder::read\(\)](#).

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

in	<i>buffer</i>	Buffer that was used to capture the data from the transcode read operation. Applications could call IAudioBuffer::reset() and reuse this buffer for subsequent read operations on the same transcoder instance. Also <code>buffer.getDataSize()</code> will represent the number of bytes contained in a buffer.
in	<i>isLastBuffer</i>	represents whether the transcoded buffer is last buffer or not. Once the last buffer is received no more further read operations are required.
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. <code>ErrorCode</code>

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

6.2.1.13 **using telux::audio::TranscoderWriteResponseCb = typedef std- ::function<void(std::shared_ptr<IAudioBuffer> buffer, uint32_t bytesWritten, telux::common::ErrorCode error)>**

This function is called with the response to [ITranscoder::write\(\)](#).

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

in	<i>buffer</i>	Buffer that was used for the write operation for transcoding. Application could call IAudioBuffer::reset() and reuse this buffer for subsequent write operations on the same transcoder instance.
in	<i>bytesWritten</i>	Return how many bytes are sent for transcoding.

<code>in</code>	<code>error</code>	Return code which indicates whether the operation succeeded or not. <code>ErrorCode</code>

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

6.2.2 Variable Documentation

6.2.2.1 `const uint16_t telux::audio::INFINITE_DTMF_DURATION = 0xFFFF`

Duration to play DTMF tone for infinite time

6.2.2.2 `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](#)
- class [DeviceConfig](#)
- class [Log](#)
- struct [SdkVersion](#)
- class [Version](#)

Provides version of SDK. [More...](#)

6.3.1 Typedef Documentation

6.3.1.1 `using telux::common::ResponseCallback = typedef std::function<void(telux::common::ErrorCode errorCode)>`

General response callback for most of the requests, client needs to implement this function to get the asynchronous response.

The methods in callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

<code>in</code>	<code>errorCode</code>	ErrorCode

6.3.1.2 using telux::common::InitResponseCb = typedef std::function<void(telux::common::ServiceStatus status)>

This API is invoked when the initialization of an object completes.

Parameters

in	status	- Service status

6.4 telux::config Namespace Reference

Data Structures

- class [ConfigFactory](#)

[ConfigFactory](#) allows creation of config related classes. [More...](#)

- struct [ConfigInfo](#)

- class [IModemConfigListener](#)

Listener class for getting notifications related to configuration change detection. The client needs to implement these methods as briefly as possible and avoid blocking calls in it. The methods in this class can be invoked from multiple different threads. Client needs to make sure that the implementation is thread-safe. [More...](#)

- class [IModemConfigManager](#)

[IModemConfigManager](#) provides interface to list config files present in modem's storage. load a new config file in modem, activate a config file, get active config file information, deactivate a config file, delete config file from the modem's storage, get and set mode of config auto selection, register and deregister listener for config update in modem. The config files are also referred to as MBNs. [More...](#)

6.4.1 Typedef Documentation

6.4.1.1 using telux::config::ConfigId = typedef std::string

6.4.1.2 using telux::config::ConfigListCallback = typedef std::function<void (std::vector<ConfigInfo> configList, telux::common::ErrorCode error)>

This function is called as a response to requestConfigList().

Parameters

in	configList	- Contains the list of config files in modem's storage.
in	error	- Return code which indicates whether the operation succeeded or not. ErrorCode

6.4.1.3 using telux::config::GetAutoSelectionModeCallback = typedef std::function<void (AutoSelectionMode mode, telux::common::ErrorCode error)>

This function is called as a response to getAutoSelectionMode().

Parameters

in	<i>mode</i>	- contains status of auto selection for configs.
in	<i>error</i>	- Return code which indicates whether the operation succeeded or not. ErrorCode

6.4.1.4 using telux::config::GetActiveConfigCallback = typedef std::function<void (ConfigInfo configInfo, telux::common::ErrorCode error)>

This function is called as a response to getActiveConfig().

Parameters

in	<i>configInfo</i>	- Information of active config file for specified slot id.
in	<i>error</i>	- Return code which indicates whether the operation succeeded or not. ErrorCode

6.5 telux::cv2x Namespace Reference

Data Structures

- class [ICv2xConfigListener](#)

Listeners for ICv2xConfig must implement this interface. [More...](#)

- class [ICv2xConfig](#)

Cv2xConfig provide operations to update or request cv2x configuration. [More...](#)

- class [Cv2xFactory](#)

Cv2xFactory is the factory that creates the Cv2x Radio. [More...](#)

- class [ICv2xRadio](#)

- class [ICv2xRadioListener](#)

Listeners for Cv2xRadio must implement this interface. [More...](#)

- class [ICv2xListener](#)

Cv2x Radio Manager listeners implement this interface.

- class [ICv2xRadioManager](#)

Cv2xRadioManager manages instances of Cv2xRadio. [More...](#)

- struct [SocketInfo](#)

- struct [Cv2xStatus](#)
- struct [Cv2xPoolStatus](#)
- struct [Cv2xStatusEx](#)
- struct [TxPoolIdInfo](#)
- struct [EventFlowInfo](#)
- struct [SpsFlowInfo](#)
- struct [Cv2xRadioCapabilities](#)
- struct [MacDetails](#)
- struct [SpsSchedulingInfo](#)
- struct [TrustedUEInfo](#)
- struct [TrustedUEInfoList](#)
- struct [IPv6Address](#)
- struct [DataSessionSettings](#)
- struct [ConfigEventInfo](#)
- struct [RFTxInfo](#)
- struct [TxStatusReport](#)
- class [ICv2xRxSubscription](#)
- class [ICv2xThrottleManagerListener](#)

Listener class for getting filter rate update notification. [More...](#)

- class [ICv2xThrottleManager](#)

ThrottleManager provides throttle manager client interface. [More...](#)

- class [ICv2xTxFlow](#)
- class [ICv2xTxRxSocket](#)
- class [ICv2xTxStatusReportListener](#)

Listeners for CV2X Tx status report must implement this interface. [More...](#)

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

<code>in</code>	<code>rxSub</code>	- Rx Subscription

<i>in</i>	<i>error</i>	- Indicates whether socket creation succeeded • SUCCESS • GENERIC_FAILURE

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

<i>in</i>	<i>spsFlow</i>	- Sps flow
<i>in</i>	<i>eventFlow</i>	- Optional event flow. Will be nullptr if event flow was not specified in the request
<i>in</i>	<i>error</i>	- Indicates whether Tx SPS flow creation succeeded • SUCCESS • GENERIC_FAILURE
<i>in</i>	<i>error</i>	- Indicates whether optional Tx Event flow creation succeeded • SUCCESS

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

<i>in</i>	<i>txEventFlow</i>	- Event flow
<i>in</i>	<i>error</i>	- Indicates whether Tx event flow creation succeeded • SUCCESS • GENERIC_FAILURE

6.5.1.4 `using telux::cv2x::CloseTxFlowCallback = typedef std::function<void (std::shared_ptr<ICv2xTxFlow> txFlow, telux::common::ErrorCode error)>`

This function is called with the response to closeTxFlow.

in	<i>txFlow</i>	- Closed tx flow
in	<i>error</i>	- Indicates whether close operation succeeded • SUCCESS • GENERIC_FAILURE

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 • SUCCESS • GENERIC_FAILURE

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 • SUCCESS • GENERIC_FAILURE

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 • SUCCESS • GENERIC_FAILURE

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 • SUCCESS • GENERIC_FAILURE

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

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

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.

<i>in</i>	<i>error</i>	- SUCCESS if update succeeded • INVALID_ARGUMENTS if trustedUEs or maliciousIds length greater than maximum value • SUCCESS • GENERIC_FAILURE • INVALID_ARGUMENTS

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

<i>in</i>	<i>error</i>	- SUCCESS if Tx reservation change succeeded • SUCCESS • GENERIC_FAILURE

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

<i>in</i>	<i>sock</i>	- TCP socket
<i>in</i>	<i>error</i>	- Indicates whether TCP socket creation succeeded • SUCCESS • GENERIC_FAILURE

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

<i>in</i>	<i>sock</i>	- Closed TCP socket
<i>in</i>	<i>error</i>	- Indicates whether close operation succeeded • SUCCESS • GENERIC_FAILURE

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

<i>in</i>	<i>error</i>	- SUCCESS if Cv2x mode successfully started • SUCCESS • GENERIC_FAILURE

6.5.1.16 using telux::cv2x::StopCv2xCallback = typedef std::function<void (telux::common::ErrorCode error)>

This function is called as a response to stopCv2x

Parameters

<i>in</i>	<i>error</i>	- SUCCESS if Cv2x mode successfully stopped • SUCCESS • GENERIC_FAILURE

6.5.1.17 using telux::cv2x::RequestCv2xStatusCallback = typedef std::function<void (Cv2xStatus status, telux::common::ErrorCode error)>

This function is called as a response to requestCv2xStatus

Parameters

<i>in</i>	<i>status</i>	- Cv2x status
<i>in</i>	<i>error</i>	- SUCCESS if Cv2x status was successfully retrieved • SUCCESS • GENERIC_FAILURE

Deprecated use RequestCv2xStatusCallbackEx

6.5.1.18 using telux::cv2x::RequestCv2xStatusCallbackEx = typedef std::function<void (Cv2xStatusEx status, telux::common::ErrorCode error)>

This function is called as a response to requestCv2xStatus

Parameters

<i>in</i>	<i>status</i>	- Cv2x status
<i>in</i>	<i>error</i>	- SUCCESS if Cv2x status was successfully retrieved • SUCCESS • GENERIC_FAILURE

6.5.1.19 using `telux::cv2x::UpdateConfigurationCallback = typedef std::function<void (telux::common::ErrorCode error)>`

This function is called with the response to updateConfiguration

Parameters

<i>in</i>	<i>error</i>	- SUCCESS if configuration was updated successfully • SUCCESS • GENERIC_FAILURE

6.5.1.20 using `telux::cv2x::setVerificationLoadCallback = typedef std::function<void (telux::common::ErrorCode error)>`

This function is called as a response to setVerificationLoad

Parameters

<i>in</i>	<i>error</i>	- SUCCESS if verification load was set successfully • SUCCESS • GENERIC_FAILURE

6.6 telux::data Namespace Reference

Namespaces

- namespace [net](#)

Data Structures

- struct [IpFamilyInfo](#)
- struct [QosFilterRule](#)
- struct [TrafficFlowTemplate](#)
- struct [TftChangeInfo](#)
- class [IDataConnectionManager](#)

[IDataConnectionManager](#) is a primary interface for cellular connectivity This interface provides APIs for

start and stop data call connections, get data call information and listener for monitoring data calls. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready. [More...](#)

- class [IDataCall](#)

Represents single established data call on the device. [More...](#)

- class [IDataConnectionListener](#)
- struct [DataRestrictMode](#)
- struct [PortInfo](#)
- struct [ProfileParams](#)
- struct [DataCallStats](#)
- struct [IpAddrInfo](#)
- struct [DataCallEndReason](#)
- struct [LatencyConfig](#)
- struct [BackhaulInfo](#)
- struct [VlanConfig](#)
- struct [FlowDataRate](#)
- struct [QosIPFlowInfo](#)
- class [DataFactory](#)

[DataFactory](#) is the central factory to create all data classes. [More...](#)

- 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 [DataProfile](#)

[DataProfile](#) class represents single data profile on the modem. [More...](#)

- class [IDataProfileListener](#)

Listener class for getting profile change notification. [More...](#)

- class [IDataProfileManager](#)
- class [IDataCreateProfileCallback](#)
- 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 [IDataProfileCallback](#)
- class [IDataSettingsManager](#)

The Data Settings Manager class provides an interface to the data subsystem settings. [More...](#)

- class [IDataSettingsListener](#)
- struct [IPv4Info](#)
- struct [IPv6Info](#)
- struct [UdpInfo](#)
- struct [TcpInfo](#)
- struct [IcmpInfo](#)
- struct [EspInfo](#)
- 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.

- class [IUdpFilter](#)

This class represents a IP Filter for the UDP, get the new instance from [telux::data::DataFactory](#).

- class [ITcpFilter](#)

This class represents a IP Filter for the TCP, get the new instance from [telux::data::DataFactory](#).

- class [IcmpFilter](#)

This class represents a IP Filter for the ICMP, get the new instance from [telux::data::DataFactory](#).

- class [IEspFilter](#)

This class represents a IP Filter for the ESP, get the new instance from [telux::data::DataFactory](#).

- struct [RoamingStatus](#)

Roaming Status. [More...](#)

- struct [ServiceStatus](#)

Data Service Status Info. [More...](#)

- class [IServingSystemManager](#)

Serving System Manager class provides APIs related to the serving system for data functionality. For example, ability to query or be notified about the state of the platform's WWAN PS data serving information. [More...](#)

- class [IServingSystemListener](#)

Listener class for data serving system change notification. [More...](#)

- union [DataCallEndReason](#).[__unnamed__](#)

6.6.1 Data Structure Documentation

6.6.1.1 struct telux::data::IpFamilyInfo

IP Family related Info

Data fields

Type	Field	Description
DataCallStatus	status	
IpAddrInfo	addr	

6.6.1.2 struct telux::data::QosFilterRule

Encapsulate the Qos Filter rule

Data fields

Type	Field	Description
vector< shared- _ptr< IpFilter > >	filter	IpFilter
uint16_t	filterId	Unique identifier for each filter.
uint16_t	filterPrecedence	Specifies the order in which filters are applied. A lower numerical value has a higher precedence.

6.6.1.3 struct telux::data::TrafficFlowTemplate

QOS TFT Flow info

Data fields

Type	Field	Description
QosFlowId	qosId	Mandatory defines current flow id
QosFlowState- ChangeEvent	stateChange	Flow state change event
QosFlowMask	mask	bitmask to denote which of the optional fields in TrafficFlowTemplate are valid
QosIPFlowInfo	txGrantedFlow	Optional
QosIPFlowInfo	rxGrantedFlow	
uint32_t	txFiltersLength	
QosFilterRule	txFilters[MAX- _QOS_FILTE- RS]	
uint32_t	rxFiltersLength	

Type	Field	Description
QosFilterRule	rxFilters[MAX-_QOS_FILTE- RS]	

6.6.1.4 struct telux::data::TftChangeInfo

QOS TFT flow change info

Data fields

Type	Field	Description
shared_ptr< TrafficFlow- Template >	tft	TFT flow info TrafficFlowTemplate
QosFlowState- ChangeEvent	stateChange	Flow state change event

6.6.1.5 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	
TypeOfService	value	level of throughput, reliability, and delay
TypeOfService	mask	
IpProtocol	nextProtoId	Protocol ID (i.e TCP, UDP or ICMP)

6.6.1.6 struct telux::data::IPv6Info

IPv6 header info

Data fields

Type	Field	Description
string	srcAddr	address of the device that sends the packet.
uint8_t	srcPrefixLen	source prefix length used to create subnet
string	destAddr	address of receiving end
uint8_t	dstPrefixLen	destination prefix length used to create subnet
IpProtocol	nextProtoId	Protocol ID (i.e TCP, UDP or ICMP)

Type	Field	Description
TrafficClass	val	indicates the class or priority of the IPv6 packet, enables the ability to track specific traffic flows at the network layer.
TrafficClass	mask	
FlowLabel	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	

6.6.1.7 struct telux::data::UdpInfo

UDP header info

Data fields

Type	Field	Description
PortInfo	src	Source port and range
PortInfo	dest	Destination port and range

6.6.1.8 struct telux::data::TcpInfo

TCP header info

Data fields

Type	Field	Description
PortInfo	src	Source port and range
PortInfo	dest	Destination port and range

6.6.1.9 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

6.6.1.10 struct telux::data::EspInfo

Encapsulating Security Payload

Data fields

Type	Field	Description
uint32_t	spi	Security Parameters Index

6.6.1.11 struct telux::data::RoamingStatus

Roaming Status.

Data fields

Type	Field	Description
bool	isRoaming	True: Roaming on, False: Roaming off
RoamingType	type	International/Domestic. Valid only if roaming is on

6.6.1.12 struct telux::data::ServiceStatus

Data Service Status Info.

Data fields

Type	Field	Description
DataServiceState	serviceState	
NetworkRat	networkRat	

6.6.2 Typedef Documentation**6.6.2.1 using telux::data::DataCallResponseCb = typedef std::function<void(const std::shared_ptr<IDataCall> &dataCall, telux::common::ErrorCode error)>**

This function is called with the response to startDataCall / stopDataCall API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

When callback is used with startDataCall, expected behavior is as following:

- If this is first client to start datacall in the system and no error is detected, state of data call will be NET_CONNECTING and onDataCallInfoChanged will be called once data call is brought up successfully or failed.
- If client tries to start data call that is already up and no error is detected, state of data call will be NET_CONNECTED and onDataCallInfoChanged will not get called.
- If any client that start data call and error is detected, error argument will contain error code and onDataCallInfoChanged will not get called.
- If client tries to start data call on specific profile id and specifies a network interface name to be used the behavior would be as follows:
 - if data call already exists with the same profile ID but a different network interface then error will be set to telux::ErrorCode::DEVICE_IN_USE and onDataCallInfoChanged will not get called.

- if data call already exists with the same network interface but different profile ID then error will be set to `telux::ErrorCode::INVALID_OPERATION` and `onDataCallInfoChanged` will not get called.
- If a data call exists with the same profile id and network interface name, and that data call is currently being torn down, error will be set to `telux::ErrorCode::OP_IN_PROGRESS` and `onDataCallInfoChanged` will not get called. Client can reattempt after the tear down is complete.
- On platforms where a client is not allowed to specify the name of the network interface when starting a data call, the callback will be invoked with error `telux::ErrorCode::REQUEST_NOT_SUPPORTED` and `onDataCallInfoChanged` will not get called.

When callback is used with `stopDataCall`, expected behavior is as following:

- First/Last client that attempts to stop data call and no error is detected, state of data call will be `NET_DISCONNECTING` and `onDataCallInfoChanged` will be called once data call is down.
- If a client starts a data call and then tries to stop it while there are other clients in the system who also started the same data call, and no error is detected, data call status will be `NET_CONNECTED` and `onDataCallInfoChanged` will not get called.
- If any client attempt to stop data call and error detected, error argument will contain error code and `onDataCallInfoChanged` will not get called.

Parameters

in	<i>dataCall</i>	Pointer to IDataCall
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 IDataCall list
in	<i>error</i>	Return code for whether the operation succeeded or failed

6.6.2.4 **using telux::data::TrafficFlowTemplateCb = typedef std::function<void(const std::vector<std::shared_ptr<TrafficFlowTemplate>> &tft, telux::common::ErrorCode error)>**

This function is called in the response to requestTrafficFlowTemplate().

Parameters

in	<i>tft</i>	Vector of TFT flow info. TrafficFlowTemplate
in	<i>error</i>	Code which indicates whether the operation succeeded or not. ErrorCode.

6.6.2.5 **using telux::data::ApnTypes = typedef std::bitset<16>**

16 bit mask to set apn types paramater. ApnMaskType enum are used to set apn types.

6.6.2.6 **using telux::data::IpProtocol = typedef uint8_t**

Internet (IP) protocol numbers found in IPv4 or IPv6 headers the protocol numbers are defined by Internet Assigned Numbers Authority (IANA)

6.6.2.7 **using telux::data::QosFlowId = typedef uint32_t**

QOS Flow identifier

6.6.2.8 **using telux::data::QosIPFlowMask = typedef std::bitset<16>**

16 bit mask that denotes which of the flow paramaters defined in QosIPFlowMaskType enum are used for TFT .

6.6.2.9 **using telux::data::QosFlowMask = typedef std::bitset<16>**

16 bit mask that denotes which of the flow paramaters defined in QosFlowMaskType enum are used for TFT .

6.6.2.10 **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.

6.6.2.11 `using telux::data::TypeOfService = typedef uint8_t`

6.6.2.12 `using telux::data::TrafficClass = typedef uint8_t`

6.6.2.13 `using telux::data::FlowLabel = typedef uint32_t`

6.6.2.14 `using telux::data::RequestServiceStatusResponseCb = typedef std-
::function<void(ServiceStatus serviceStatus, telux::common::ErrorCode
error)>`

This function is called in response to requestServiceStatus API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

in	<i>serviceStatus</i>	Current service status telux::data::ServiceStatus
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.

6.6.2.15 `using telux::data::RequestRoamingStatusResponseCb = typedef std-
::function<void(RoamingStatus roamingStatus, telux::common::ErrorCode
error)>`

This function is called in response to requestRoamingStatus API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

in	<i>roamingStatus</i>	Current roaming status telux::data::RoamingStatus
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.

6.6.3 Enumeration Type Documentation

6.6.3.1 `enum telux::data::RoamingType`

Roaming Type.

Enumerator

UNKNOWN Device roaming mode is unknown

DOMESTIC Device is in Domestic roaming network

INTERNATIONAL Device is in International roaming network

6.6.3.2 enum telux::data::DataServiceState

Data Service State. Indicates whether data service is ready to setup a data call or not.

Enumerator

UNKNOWN Service State not available

IN_SERVICE Service Available

OUT_OF_SERVICE Service Not Available

6.6.3.3 enum telux::data::NetworkRat

Data Network RATs.

Enumerator

UNKNOWN UNKNOWN

CDMA_1X CDMA_1X

CDMA_EVDO CDMA_EVDO

GSM GSM

WCDMA WCDMA

LTE LTE

TDSCDMA TDSCDMA

NR5G NR5G

6.7 telux::data::net Namespace Reference

Data Structures

- struct [BridgeInfo](#)
- class [IBridgeManager](#)

[IBridgeManager](#) provides APIs to enable/disable and set/get/delete software bridges for various WLAN and Ethernet interfaces. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready. [More...](#)
- class [IBridgeListener](#)
- struct [FirewallConfig](#)
- struct [DmzConfig](#)
- struct [FirewallEntryInfo](#)
- class [IFirewallManager](#)

[FirewallManager](#) is a primary interface that filters and controls the network traffic on a pre-configured set of rules. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready. [More...](#)
- class [IFirewallEntry](#)

Firewall entry class is used for configuring firewall rules. [More...](#)

- class [IFirewallListener](#)
- struct [L2tpSessionConfig](#)
- struct [L2tpTunnelConfig](#)
- struct [L2tpSysConfig](#)
- class [IL2tpManager](#)

L2tpManager is a primary interface for configuring L2TP Service. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready. [More...](#)

- class [IL2tpListener](#)
- struct [NatConfig](#)
- class [INatManager](#)

NatManager is a primary interface for configuring static network address translation(SNAT) and DMZ (demilitarized zone). It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready. [More...](#)

- class [INatListener](#)
- class [ISocksManager](#)

SocksManager is a primary interface for configuring legacy Socks proxy server. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready. [More...](#)

- class [ISocksListener](#)
- struct [VlanBindConfig](#)
- class [IVlanManager](#)

VlanManager is a primary interface for configuring VLAN (Virtual Local Area Network). it provide APIs for create, query, remove VLAN interfaces and associate or disassociate with profile IDs. It also provides interface to Subsystem Restart events by registering as listener. Notifications will be received when modem is ready/not ready. [More...](#)

- class [IVlanListener](#)

6.7.1 Data Structure Documentation

6.7.1.1 struct telux::data::net::BridgeInfo

Structure to configure a software bridge for an interface

Data fields

Type	Field	Description
string	ifaceName	Interface name
BridgeIFace-Type	ifaceType	Interface type
uint32_t	bandwidth	Bandwidth(in Mbps) required for software bridge

6.7.1.2 struct telux::data::net::FirewallConfig

Firewall configuration parameters

Data fields

Type	Field	Description
BackhaulInfo	bhInfo	Backhaul Information to apply firewal settings on
bool	enable	True: Firewall enabled. False: Firewall disabled
bool	allowPackets	True: Packets that match rules will be allowed. False: Packets that match rules will be dropped

6.7.1.3 struct telux::data::net::DmzConfig

DMZ configuration parameters

Data fields

Type	Field	Description
BackhaulInfo	bhInfo	Backhaul Information to apply firewal settings on
string	ipAddr	IP address for which DMZ will be enabled

6.7.1.4 struct telux::data::net::FirewallEntryInfo

Firewall rules parameters

Data fields

Type	Field	Description
shared_ptr< I-FirewallEntry >	fwEntry	Shared pointer to firewall rules for the backhaul
BackhaulInfo	bhInfo	Backhaul Information to add firewal rules on

6.7.1.5 struct telux::data::net::VlanBindConfig

Data fields

Type	Field	Description
int	vlanId	
BackhaulInfo	bhInfo	VLAN ID to be bound to the specified backhaul Configuration of Backhaul to bind VLAN to

6.7.2 Typedef Documentation

6.7.2.1 using telux::data::net::BridgeInfoResponseCb = typedef std::function<void(const std::vector<BridgeInfo> &infos, telux::common::ErrorCode error)>

This function is called as a response to requestBridgeInfo

Parameters

in	<i>infos</i>	List of the software bridge configurations in the system
in	<i>error</i>	Return code which indicates whether the operation is succeeded or not telux::common::ErrorCode

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

6.7.2.2 using telux::data::net::FirewallConfigCb = typedef std::function<void(FirewallConfig status, telux::common::ErrorCode error)>

This function is called as a response to requestFirewallConfig()

Parameters

in	<i>config</i>	Firewall configuration status for specific backhaul telux::data::FirewallConfig .
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. telux::common::ErrorCode

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

6.7.2.3 using telux::data::net::FirewallEntryInfoCb = typedef std::function<void(std::vector<FirewallEntryInfo> entry, telux::common::ErrorCode error)>

This function is called as a response to requestFirewallEntries()

Parameters

in	<i>entries</i>	Vector of firewall entries for specific backhaul
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. telux::common::ErrorCode

6.7.2.4 using telux::data::net::DmzEntryInfoCb = typedef std::function<void(std::vector<DmzConfig> dmzEntries, telux::common::ErrorCode error)>

This function is called as a response to requestDmzEntries()

in	<i>dmzEntries</i>	list of dmz entries configurations
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. telux::common::ErrorCode

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

6.7.2.5 using `telux::data::net::FirewallStatusCb = typedef std::function<void(bool enable, bool allowPackets, telux::common::ErrorCode error)>`

This function is called as a response to `requestFirewallStatus()`

Parameters

in	<i>enable</i>	Indicates whether the firewall is enabled
in	<i>allowPackets</i>	Indicates whether to accept or drop packets matching the rules
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. telux::common::ErrorCode

Deprecated `telux::data::FirewallConfigCb` callback is used to get Firewall configuration

6.7.2.6 using `telux::data::net::FirewallEntriesCb = typedef std::function<void(std::vector<std::shared_ptr<IFirewallEntry>> entries, telux::common::ErrorCode error)>`

This function is called as a response to `requestFirewallEntries()`

Parameters

in	<i>entries</i>	list of firewall entries
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. telux::common::ErrorCode

Deprecated `telux::data::FirewallEntryInfoCb` callback is used to get Firewall entries

6.7.2.7 using `telux::data::net::DmzEntriesCb = typedef std::function<void(std::vector<std::string> dmzEntries, telux::common::ErrorCode error)>`

This function is called as a response to `requestDmzEntries()`

Parameters

in	<i>dmzEntries</i>	list of dmz entries
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. telux::common::ErrorCode

Deprecated `telux::data::DmzEntryInfoCb` callback is used to get DMZ entries

6.7.2.8 using `telux::data::net::L2tpConfigCb = typedef std::function<void(const L2tpSysConfig &l2tpSysConfig, telux::common::ErrorCode error)>`

This function is called as a response to `requestConfig()`

Parameters

in	<i>l2tpSysConfig</i>	Current L2TP configuration
in	<i>error</i>	Return code which indicates whether the operation succeeded or not telux::common::ErrorCode

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

6.7.2.9 using `telux::data::net::StaticNatEntriesCb = typedef std::function<void(const std::vector<NatConfig> &snatEntries, telux::common::ErrorCode error)>`

This function is called as a response to `requestStaticNatEntries()`

Parameters

in	<i>snatEntries</i>	list of static Network Address Translation (NAT)
in	<i>error</i>	Return code which indicates whether the operation succeeded or not telux::common::ErrorCode

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

6.7.2.10 using `telux::data::net::CreateVlanCb = typedef std::function<void(bool isAccelerated, telux::common::ErrorCode error)>`

This function is called as a response to `createVlan()`

Parameters

in	<i>isAccelerated</i>	Offload status returned by server
in	<i>error</i>	Return code which indicates whether the operation succeeded or not telux::common::ErrorCode

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

6.7.2.11 **using telux::data::net::QueryVlanResponseCb = typedef std::function<void(const std::vector<VlanConfig> &configs, telux::common::ErrorCode error)>**

This function is called as a response to queryVlanInfo()

Parameters

in	<i>configs</i>	List of VLAN configs
in	<i>error</i>	Return code which indicates whether the operation succeeded or not telux::common::ErrorCode

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

6.7.2.12 **using telux::data::net::VlanMappingResponseCb = typedef std::function<void(const std::list<std::pair<int, int>> &mapping, telux::common::ErrorCode error)>**

This function is called as a response to queryVlanMappingList()

Parameters

in	<i>mapping</i>	List of profile Id and Vlan id map Key is Profile Id and value is VLAN id
in	<i>error</i>	Return code which indicates whether the operation succeeded or not telux::common::ErrorCode

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

6.7.2.13 **using telux::data::net::VlanBindingsResponseCb = typedef std::function<void(const std::vector<VlanBindConfig> bindings, telux::common::ErrorCode error)>**

This function is called as a response to queryVlanToBackhaulBindings()

Parameters

in	<i>bindings</i>	list of Vlan binding configurations telux::data::net::VlanBindConfig
in	<i>error</i>	Return code which indicates whether the operation succeeded or not telux::common::ErrorCode

Note

Eval: This is a new API and is being evaluated. It is subject to change.

6.7.3 Enumeration Type Documentation

6.7.3.1 enum telux::data::net::BridgeFaceType

Interface types supported for bridge configuration

Enumerator

UNKNOWN

WLAN_AP Wireless Local Area Network (WLAN) in AP mode

WLAN_STA Wireless Local Area Network (WLAN) in STA mode

ETH Ethernet (ETH)

6.8 telux::loc Namespace Reference

Data Structures

- class [IDgnssStatusListener](#)

Listener class for getting RTCM injection event notification information. [More...](#)

- class [IDgnssManager](#)

IRtcManager provides interface to inject RTCM data into modem, register event listener reported by cdfw(correction data framework). [More...](#)

- class [ILocationConfigurator](#)

[ILocationConfigurator](#) allows general engine configurations (example: TUNC, PACE etc), configuration of specific engines like SPE (example: minSVElevation, minGPSWeek etc) or DRE, deletion of warm and cold aiding data, NMEA configuration and support for XTRA feature. [ILocationConfigurator](#) APIs strictly adheres to the principle of single client per process.

- struct [GnssKinematicsData](#)
- struct [LLAInfo](#)
- struct [TimeInfo](#)
- struct [GlonassTimeInfo](#)
- union [SystemTimeInfo](#)
- struct [SystemTime](#)
- struct [GnssMeasurementInfo](#)
- struct [SvUsedInPosition](#)
- struct [GnssData](#)
- struct [SvBlackListInfo](#)
- struct [LeverArmParams](#)
- struct [ExternalLocationInfo](#)
- struct [GnssMeasurementsData](#)
- struct [GnssMeasurementsClock](#)

- struct [GnssMeasurements](#)
- struct [LeapSecondChangeInfo](#)
- struct [LeapSecondInfo](#)
- struct [LocationSystemInfo](#)
- struct [GnssEnergyConsumedInfo](#)
- struct [RobustLocationVersion](#)
- struct [RobustLocationConfiguration](#)
- struct [NmeaConfig](#)
- struct [BodyToSensorMountParams](#)
- struct [DREngineConfiguration](#)
- struct [XtraConfig](#)
- struct [XtraStatus](#)
- 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 [ISVInfo](#)

[ISVInfo](#) provides interface to retrieve information about Satellite Vehicles, their position and health status. [More...](#)

- class [IGnssSVInfo](#)

[IGnssSVInfo](#) provides interface to retrieve the list of SV info available and whether altitude is assumed or calculated. [More...](#)

- class [IGnssSignalInfo](#)

[IGnssSignalInfo](#) provides interface to retrieve GNSS data information like jammer metrics and automatic gain control for satellite signal type. [More...](#)

- class [LocationFactory](#)

[LocationFactory](#) allows creation of location manager. [More...](#)

- class [ILocationListener](#)

[ILocationListener](#) Listener class for getting location updates and satellite vehicle information. [More...](#)

- class [ILocationSystemInfoListener](#)

- class [ILocationInjectionListener](#)

- class [ILocationConfigListener](#)

[ILocationConfigListener](#) interface is used to receive notifications related to configuration events. [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.

6.8.1 Typedef Documentation

6.8.1.1 `using telux::loc::SbasCorrection = typedef std::bitset<SBAS_COUNT>`

8 bit mask that denotes which of the SBAS corrections in *SbasCorrection* used to improve the performance of GNSS output.

6.8.1.2 `using telux::loc::GnssPositionTech = typedef uint32_t`

6.8.1.3 `using telux::loc::KinematicDataValidity = typedef uint32_t`

6.8.1.4 `using telux::loc::LocConfigIndications = typedef std::bitset<32>`

This bitset represents the list of the Location Config Indications selected by the Client.

6.8.1.5 `using telux::loc::GnssReportTypeMask = typedef uint32_t`

Specifies the applicable reports using the bits represented in *GnssReportType*

6.8.1.6 `using telux::loc::DRConfigValidity = typedef uint16_t`

Specifies *DRConfigValidityType*

6.8.1.7 `using telux::loc::NmeaSentenceConfig = typedef uint32_t`

Specifies *NmeaSentenceType* mask

6.8.1.8 `using telux::loc::RobustLocationConfig = typedef uint16_t`

Specifies *RobustLocationConfigType* mask

6.8.1.9 `using telux::loc::TerrestrialTechnology = typedef uint32_t`

Specifies *TerrestrialTechnologyType* mask

6.8.1.10 `using telux::loc::AidingData = typedef uint32_t`

Specifies *AidingDataType* mask

6.8.1.11 `using telux::loc::GnssEnergyConsumedInfoValidity = typedef uint16_t`

Specifies *GnssEnergyConsumedInfoValidityType*

6.8.1.12 using telux::loc::LocationSystemInfoValidity = typedef uint32_t

Specifies LocationSystemInfoValidityType mask

6.8.1.13 using telux::loc::LeapSecondInfoValidity = typedef uint32_t

Specifies LeapSecondInfoValidityType mask

6.8.1.14 using telux::loc::GnssMeasurementsClockValidity = typedef uint32_t

Specifies GnssMeasurementsClockValidityType.

6.8.1.15 using telux::loc::GnssMeasurementsAdrStateValidity = typedef uint32_t

Specifies GnssMeasurementsAdrStateValidityType.

6.8.1.16 using telux::loc::GnssMeasurementsStateValidity = typedef uint32_t

Specifies GnssMeasurementsStateValidityType.

6.8.1.17 using telux::loc::GnssMeasurementsDataValidity = typedef uint32_t

Specifies GnssMeasurementsDataValidityType.

6.8.1.18 using telux::loc::PositioningEngine = typedef uint32_t

Specifies PositioningEngineType mask

6.8.1.19 using telux::loc::LocReqEngine = typedef uint16_t

Specifies LocReqEngineType mask

6.8.1.20 using telux::loc::DrCalibrationStatus = typedef uint32_t

Specifies DrCalibrationStatusType mask

6.8.1.21 using telux::loc::GnssDataValidity = typedef uint32_t

Specifies GnssDataValidityType mask

6.8.1.22 using telux::loc::LocationInfoExValidity = typedef uint64_t**6.8.1.23 using telux::loc::LocationInfoValidity = typedef uint32_t****6.8.1.24 using telux::loc::LocationTechnology = typedef uint32_t****6.8.1.25 using telux::loc::GnssSignal = typedef uint32_t**

6.8.1.26 using telux::loc::GnssTimeValidity = typedef uint32_t

6.8.1.27 using telux::loc::TimeValidity = typedef uint32_t

6.8.2 Enumeration Type Documentation

6.8.2.1 enum telux::loc::SbasCorrectionType

Defines Satellite Based Augmentation System(SBAS) corrections. SBAS contributes to improve the performance of GNSS system.

Enumerator

SBAS_CORRECTION_IONO Bit mask to specify whether SBAS ionospheric correction is used
SBAS_CORRECTION_FAST Bit mask to specify whether SBAS fast correction is used
SBAS_CORRECTION_LONG Bit mask to specify whether SBAS long correction is used
SBAS_INTEGRITY Bit mask to specify whether SBAS integrity information is used
SBAS_CORRECTION_DGNSS Bit mask to specify whether SBAS DGNSS correction is used
SBAS_CORRECTION_RTK Bit mask to specify whether SBAS RTK correction is used
SBAS_CORRECTION_PPP Bit mask to specify whether SBAS PPP correction is used
SBAS_CORRECTION_RTK_FIXED Bit mask to specify whether SBAS RTK fixed correction is used
SBAS_CORRECTION_ONLY_SBAS_CORRECTED_SV_USED Bit mask to specify only SBAS corrected SV is used
SBAS_COUNT Bitset

6.9 telux::platform Namespace Reference

Data Structures

- class [IDeviceInfoListener](#)

Listener class for getting device info related notifications . The client needs to implement these methods as briefly as possible and avoid blocking calls in it. The methods in this class can be invoked from multiple different threads. Client needs to make sure that the implementation is thread-safe. [More...](#)

- struct [PlatformVersion](#)

- class [IDeviceInfoManager](#)

[IDeviceInfoManager](#) provides interface to to retrieve IMEI and platform version operations. [More...](#)

- class [PlatformFactory](#)

[PlatformFactory](#) allows creation of Platform services related classes. [More...](#)

6.10 telux::power Namespace Reference

Data Structures

- class [PowerFactory](#)
PowerFactory allows creation of TCU-activity manager instance. [More...](#)
- 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...](#)

6.11 telux::tel Namespace Reference

Data Structures

- 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 [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 [ICardApp](#)
Represents a single card application. [More...](#)
- struct [IccResult](#)
- class [ICardManager](#)
- class [ICard](#)
ICard represents currently inserted UICC or eUICC. [More...](#)
- class [ICardChannelCallback](#)
- class [ICardCommandCallback](#)
- class [ICardListener](#)
- struct [CellBroadcastFilter](#)

- struct [Point](#)
- class [Polygon](#)
- class [Circle](#)
- class [Geometry](#)
- class [WarningAreaInfo](#)
- class [EtwsInfo](#)
- class [CmasInfo](#)
- class [CellBroadcastMessage](#)
Cell Broadcast message.
- class [ICellBroadcastManager](#)
CellBroadcastManager class is primary interface to configure and activate emergency broadcast messages and receive broadcast messages.
- class [ICellBroadcastListener](#)
A listener class which monitors cell broadcast messages.
- class [GsmCellIdentity](#)
- class [CdmaCellIdentity](#)
- class [LteCellIdentity](#)
- class [WcdmaCellIdentity](#)
- class [TdsdmaCellIdentity](#)
- class [CellInfo](#)
- class [GsmCellInfo](#)
- class [CdmaCellInfo](#)
- class [LteCellInfo](#)
- class [WcdmaCellInfo](#)
- class [TdsdmaCellInfo](#)
- struct [ECallMsdOptionals](#)
- struct [ECallMsdControlBits](#)
- struct [ECallVehicleIdentificationNumber](#)
- struct [ECallVehiclePropulsionStorageType](#)
- struct [ECallVehicleLocation](#)
- struct [ECallVehicleLocationDelta](#)
- struct [ECallDeltaV](#)
- struct [ECallOptionalEuroNcapData](#)
- struct [ECallOptionalPdu](#)

- struct [ECallMsdData](#)
- struct [ECallModeInfo](#)
- struct [ECallHlapTimerStatus](#)
- struct [ECallHlapTimerEvents](#)
- struct [EcallConfig](#)
- class [IEcallManager](#)

[IEcallManager](#) allows operations related to automotive emergency call management and its related configurations. [More...](#)

- class [IEcallListener](#)

Listener class to notify service status change notifications. The listener method can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe. [More...](#)

- struct [ImsRegistrationInfo](#)
- class [ImsServingSystemManager](#)

IMS Serving System Manager is the primary interface for IMS related operations Allows to query IMS registration status.

- class [ImsServingSystemListener](#)

A listener class for monitoring changes in IMS Serving System manager, including IMS registration status change. Override the methods for the state that you wish to receive updates for.

- struct [ImsServiceConfig](#)
- class [ImsSettingsManager](#)

ImsSettingsManager allows IMS settings. For example enabling or disabling IMS service, VOIMS service. [More...](#)

- class [ImsSettingsListener](#)

Listener class for getting IMS service configuration change notifications. The listener method can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe. [More...](#)

- struct [SlotStatus](#)
- class [IMultiSimManager](#)

MultiSimManager allows to perform operation pertaining to devices which have more than one SIM/UICC card. Clients should check if the subsystem is ready before invoking any of the APIs as follows. [More...](#)

- class [IMultiSimListener](#)

Listener class for getting high capability change notification. The listener method can be invoked from multiple different threads. Client needs to make sure that implementation is thread-safe. [More...](#)

- struct [PreferredNetworkInfo](#)
- struct [OperatorStatus](#)
- 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 [OperatorInfo](#)
- class [INetworkSelectionListener](#)

Listener class for getting network selection mode change notification. [More...](#)
- 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 [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 [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...](#)
- struct [SimRatCapability](#)
- struct [CellularCapabilityInfo](#)
- class [PhoneFactory](#)

[PhoneFactory](#) is the central factory to create all Telephony SDK Classes and services. [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 [ICellularCapabilityCallback](#)
- class [IOperatingModeCallback](#)
- 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...](#)
- struct [CardReaderStatus](#)
- class [ISapCardManager](#)

[ISapCardManager](#) provide APIs for SAP related operations. [More...](#)
- class [IAtrResponseCallback](#)

- class [ISapCardCommandCallback](#)
- class [ICardReaderCallback](#)
- class [ISapCardListener](#)
- struct [DcStatus](#)
- struct [NetworkTimeInfo](#)
- class [IServingSystemManager](#)

Serving System Manager class provides the API to request and set service domain preference and RAT preference. [More...](#)

- class [IServingSystemListener](#)

Listener class for getting radio access technology mode preference change notification. [More...](#)

- class [SignalStrength](#)
- class [LteSignalStrengthInfo](#)
- class [GsmSignalStrengthInfo](#)
- class [CdmaSignalStrengthInfo](#)
- class [WcdmaSignalStrengthInfo](#)
- class [TdsdmaSignalStrengthInfo](#)
- class [Nr5gSignalStrengthInfo](#)
- class [SimProfile](#)

SimProfile class represents single eUICC profile on the card. [More...](#)

- class [ISimProfileListener](#)

The interface listens for profile download indication and keep track of download and install progress of profile. [More...](#)

- class [ISimProfileManager](#)

ISimProfileManager is a primary interface for remote eUICCs (eSIMs or embedded SIMs) provisioning. This interface provides APIs to add, delete, set profile, update nickname, provide user consent, get Eid on the eUICC. [More...](#)

- struct [MessageAttributes](#)

Contains structure of message attributes like encoding type, number of segments, characters left in last segment. [More...](#)

- struct [MessagePartInfo](#)

Structure containing information about the part of multi-part SMS such as concatenated message reference number, number of segments and segment number. During concatenation this information along with originating address helps in associating each part of the multi-part message to the corresponding multi-part message. [More...](#)

- class [SmsMessage](#)

Data structure represents an incoming SMS. This is applicable for single part message or part of the multipart message. [More...](#)

- class [ISmsManager](#)

SmsManager class is the primary interface to manage SMS operations such as send and receive an SMS text and raw encoded PDU(s). This class handles single part and multi-part messages. [More...](#)

- class [ISmsListener](#)

A listener class receives notification for the incoming message(s) and delivery report for sent message(s). [More...](#)

- class [ISmscAddressCallback](#)

- 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 [VoiceServiceInfo](#)

6.11.1 Data Structure Documentation

6.11.1.1 struct telux::tel::Point

[Point](#) represented by latitude and longitude.

Data fields

Type	Field	Description
double	latitude	
double	longitude	

Type	Field	Description

6.11.1.2 struct telux::tel::ImsRegistrationInfo

Defines the IMS registration status parameters and the error code value

Data fields

Type	Field	Description
Registration-Status	imsRegStatus	The status of the IMS registration with the network
int	errorCode	An error code is returned when the IMS registration status is <code>RegistrationStatus::NOT_REGISTERED</code> . Values(Defined in SIP-RFC3261 section 13.2.2.2 and section 13.2.2.3): • 3xx – Redirection responses • 4xx – Client failure responses • 5xx – Server failure responses • 6xx – Global failure responses
string	errorString	Registration failure error string when the IMS is not registered.

6.11.1.3 struct telux::tel::NetworkTimeInfo

Defines Network time information

Data fields

Type	Field	Description
uint16_t	year	Year.
uint8_t	month	Month. 1 is January and 12 is December.
uint8_t	day	Day. Range: 1 to 31.
uint8_t	hour	Hour. Range: 0 to 23.
uint8_t	minute	Minute. Range: 0 to 59.
uint8_t	second	Second. Range: 0 to 59.
uint8_t	dayOfWeek	Day of the week. 0 is Monday and 6 is Sunday.
int8_t	timeZone	Offset between UTC and local time in units of 15 minutes (signed value). Actual value = field value * 15 minutes.

Type	Field	Description
uint8_t	dstAdj	Daylight saving adjustment in hours to obtain local time. Possible values: 0, 1, and 2.
string	nitzTime	Network Identity and Time Zone (NITZ) information in the form "yyyy/mm/dd,hh:mm:ss(+/-)tz:tzm,dt

6.11.2 Typedef Documentation

6.11.2.1 using telux::tel::MakeCallCallback = typedef std::function<void(telux::common::ErrorCode error, std::shared_ptr<ICall> call)>

This function is called with the response to make normal call and emergency call.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

out	<i>error</i>	ErrorCode
out	<i>call</i>	Pointer to Call object or nullptr in case of failure

6.11.2.2 using telux::tel::ECallHlapTimerStatusCallback = typedef std::function<void(telux::common::ErrorCode error, int phoneId, ECallHlapTimerStatus timersStatus)>

This function is called with response to request for eCall High Level Application Protocol (HLAP) timers status.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

out	<i>error</i>	ErrorCode
out	<i>phoneId</i>	Represents the phone corresponding to which the response is being reported.
out	<i>timersStatus</i>	ECallHlapTimerStatus

6.11.2.3 using telux::tel::EcbmStatusCallback = typedef std::function<void(telux::tel::EcbMode ecbMode, telux::common::ErrorCode error)>

This function is called with response to request for ECBM(requestEcbm API).

The callback can be invoked from multiple different threads. The implementation should be thread safe.

out	<i>ecbMode</i>	Indicates the status of the ECBM. EcbMode
out	<i>error</i>	ErrorCode

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

6.11.2.4 **using telux::tel::PinOperationResponseCb = typedef std::function<void(int retryCount, telux::common::ErrorCode error)>**

This function is called with the response to pin operations like change pin password, unlock card and set card lock.

Parameters

in	<i>retryCount</i>	No of retry attempts left
in	<i>error</i>	Return code for whether the operation succeeded or failed

6.11.2.5 **using telux::tel::QueryFdnLockResponseCb = typedef std::function<void(bool isAvailable, bool isEnabled, telux::common::ErrorCode error)>**

This function is called with the response to queryFdnLockState API.

Parameters

in	<i>isAvailable</i>	Determine FDN lock state availability
in	<i>isEnabled</i>	Determine FDN lock state i.e enable or disable
in	<i>error</i>	Return code for whether the operation succeeded or failed

6.11.2.6 **using telux::tel::QueryPin1LockResponseCb = typedef std::function<void(bool state, telux::common::ErrorCode error)>**

This function is called with the response to queryPin1LockState API.

Parameters

in	<i>state</i>	Determine state whether enabled or disabled
in	<i>error</i>	Return code for whether the operation succeeded or failed

6.11.2.7 **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.

in	<i>eid</i>	eUICC identifier.
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. ErrorCode

6.11.2.8 using telux::tel::RequestFiltersResponseCallback = typedef std::function<void(std::vector<CellBroadcastFilter> filters, telux::common::ErrorCode error)>

This function is called with the response to requestMessageIdFilters API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

in	<i>filters</i>	List of broadcast filters CellBroadcastFilter
in	<i>error</i>	Return code which indicates whether the operation succeeded or not ErrorCode

6.11.2.9 using telux::tel::RequestActivationStatusResponseCallback = typedef std::function<void(bool isActivated, telux::common::ErrorCode error)>

This function is called with the response to requestActivationStatus API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

in	<i>isActivated</i>	If true then filters are activated else filters are deactivated
in	<i>error</i>	Return code which indicates whether the operation succeeded or not ErrorCode

6.11.2.10 using telux::tel::EcallConfigValidity = typedef std::bitset<ECALL_CONFIG_COUNT>

Bit mask that denotes which of the ECall configuration parameters defined in EcallConfigType enum are valid(and to be considered) in the provided [EcallConfig](#) structure. For example, if the configuration related to Canned MSD is provided, then EcallConfigValidity valid = (1 << ECALL_CONFIG_USE_CANNED_MSD).

6.11.2.11 using telux::tel::ImsRegistrationInfoCb = typedef std::function<void(ImsRegistrationInfo status, telux::common::ErrorCode error)>

This function is called in the response to requestRegistrationInfo API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

in	<i>status</i>	Indicates the IMS registration status and the error code telux::tel::ImsRegistrationInfo .
in	<i>error</i>	Return code which indicates whether the operation succeeded or not telux::common::ErrorCode .

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

6.11.2.12 using telux::tel::ImsServiceConfigValidity = typedef std::bitset<32>

32 bit mask that denotes which of the IMS settings configuration parameters defined in `ImsServiceConfigType` enum are configured currently. For example, if the client selects the VOIMS configuration, 1st bit position is set.

6.11.2.13 using telux::tel::ImsServiceConfigCb = typedef std::function<void(SlotId slotId, ImsServiceConfig config, telux::common::ErrorCode error)>

This function is called in the response to `requestServiceConfig` API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

in	<i>slotId</i>	Slot for which the IMS service configuration is intended.
in	<i>config</i>	Indicates which configuration is valid and whether the configuration is enabled or disabled. ImsServiceConfig .
in	<i>error</i>	Return code which indicates whether the operation succeeded or not .

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

6.11.2.14 using telux::tel::HighCapabilityCallback = typedef std::function<void(int slotId, telux::common::ErrorCode error)>

This function is called in the response to `requestHighCapability` API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

in	<i>slotId</i>	SIM corresponding to slot identifier has high capability.
in	<i>error</i>	Return code which indicates whether the operation succeeded or not <code>ErrorCode</code>

6.11.2.15 `using telux::tel::SlotStatusCallback = typedef std::function<void(std::map<SlotId, SlotStatus> slotStatus, telux::common::ErrorCode error)>`

This function is called in response to requestSlotStatus API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

in	<i>slotStatus</i>	list of slots status SlotStatus
in	<i>error</i>	Return code which indicates whether the operation succeeded or not ErrorCode

6.11.2.16 `using telux::tel::RatMask = typedef std::bitset<16>`

16 bit mask that denotes which of the radio access technologies defined in RatType enum are used for preferred networks.

6.11.2.17 `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>	NetworkSelectionMode
in	<i>error</i>	Return code which indicates whether the operation succeeded or not ErrorCode

6.11.2.18 `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

6.11.2.19 `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

6.11.2.20 `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 - telux::common::ErrorCode::SUCCESS - telux::common::ErrorCode::RADIO_NOT_AVAILABLE - telux::common::ErrorCode::GENERIC_FAILURE

6.11.2.21 `using telux::tel::CellInfoCallback = typedef std::function<void(std::vector<std::shared_ptr<CellInfo>> cellInfoList, telux::common::ErrorCode error)>`

This function is called with the response to requestCellInfo API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

out	<i>cellInfoList</i>	vector of shared pointers to cell info object
out	<i>error</i>	Return code for whether the operation succeeded or failed

6.11.2.22 `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>	ECallMode
out	<i>error</i>	Return code for whether the operation succeeded or failed

6.11.2.23 using telux::tel::RATCapabilitiesMask = typedef std::bitset<16>

6.11.2.24 using telux::tel::VoiceServiceTechnologiesMask = typedef std::bitset<16>

6.11.2.25 using telux::tel::SapStateResponseCallback = typedef std::function<void(Sap-State 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>	SapState of SIM access profile (SAP) connection
in	<i>error</i>	Return code for whether the operation succeeded or failed

6.11.2.26 using telux::tel::RatPreference = typedef std::bitset<16>

16 bit mask that denotes which of the radio access technology mode preference defined in RatPrefType enum are used to set or get RAT preference.

6.11.2.27 using telux::tel::RatPreferenceCallback = typedef std::function<void(Rat-Preference 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>	RatPreference
in	<i>error</i>	Return code which indicates whether the operation succeeded or not ErrorCode

6.11.2.28 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.

<i>in</i>	<i>preference</i>	ServiceDomainPreference
<i>in</i>	<i>error</i>	Return code which indicates whether the operation succeeded or not ErrorCode

6.11.2.29 **using telux::tel::NetworkTimeResponseCallback = typedef std::function<void(NetworkTimeInfo info, telux::common::ErrorCode error)>**

This function is called with the response to requestNetworkTime API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

<i>in</i>	<i>info</i>	NetworkTimeInfo
<i>in</i>	<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.

6.11.2.30 **using telux::tel::PolicyRuleMask = typedef std::bitset<16>**

16 bit mask that denotes which of the profile policy rules(PPR) defined in Policy Rule Type enum are used.

6.11.2.31 **using telux::tel::ProfileListResponseCb = typedef std::function<void(const std::vector<std::shared_ptr<SimProfile>> &profiles, telux::common::ErrorCode error)>**

This function is called with the response to requestProfileList API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

<i>in</i>	<i>info</i>	Profiles information SimProfile .
<i>in</i>	<i>error</i>	Return code which indicates whether the operation succeeded or not. ErrorCode.

6.11.2.32 **using telux::tel::EidResponseCb = typedef std::function<void(std::string eid, telux::common::ErrorCode error)>**

This function is called with the response to requestEid API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

in	<i>eid</i>	eUICC identifier.
in	<i>error</i>	Return code which indicates whether the operation succeeded or not. ErrorCode.

6.11.2.33 `using telux::tel::PduBuffer = typedef std::vector<uint8_t>`

6.11.2.34 `using telux::tel::SmsResponseCb = typedef std::function<void(std::vector<int> msgRefs, telux::common::ErrorCode errorCode)>`

This function is called in response to sending a single part or multi-part SMS. This response callback is invoked when a single part message is sent or when all the parts of a multi-part message is sent. This function is called in response to [telux::tel::ISmsManager::sendSms](#) and [telux::tel::ISmsManager::sendRawSms](#) APIs.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

in	<i>msgRefs</i>	This parameter represent the unique message reference number(s) corresponding to single/multi-part message that we successfully sent. When part of a message is delivered, the notification API i.e telux::tel::ISmsListener::onDeliveryReport will be invoked with the message reference number corresponding to that part.
in	<i>errorCode</i>	If sending any part of a multi-part message fails or a single part message fails this API will return an <code>telux::common::errorCode</code> corresponding to the failure.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

6.11.3 Enumeration Type Documentation

6.11.3.1 `enum telux::tel::EcallConfigType`

Defines the supported ECall configuration parameters

Enumerator

ECALL_CONFIG_MUTE_RX_AUDIO Mute the local audio device during MSD transmission
ECALL_CONFIG_NUM_TYPE Decides which number needs to be dialed when an eCall is initiated
ECALL_CONFIG_OVERRIDDEN_NUM User configured/overridden number that will be dialed for eCall
ECALL_CONFIG_USE_CANNED_MSD Use the pre-defined MSD in modem for eCall
ECALL_CONFIG_GNSS_UPDATE_INTERVAL Time interval in milliseconds, at which modem updates the GNSS information in its internally generated MSD
ECALL_CONFIG_T2_TIMER T2 timer value
ECALL_CONFIG_T7_TIMER T7 timer value

ECALL_CONFIG_T9_TIMER T9 timer value

ECALL_CONFIG_MSD_VERSION MSD version to be used by modem when it internally generates MSD i.e when MSD is not sent by application and also canned MSD is not used

ECALL_CONFIG_COUNT

6.11.3.2 enum telux::tel::EcallHlapTimerId

Represents timers that need to be restarted by the application after a modem reset or when the operating mode of the device changes from low power mode to online.

Enumerator

UNKNOWN Unknown timer ID.

T9 Timer ID for T9 timer for a regulatory eCall or test eCall.
Applicable for both the eCall operating modes (Normal and eCall only).

T10 Timer ID for T10 timer for a regulatory eCall or test eCall.
Applicable for eCall only operating mode.

6.11.3.3 enum telux::tel::RegistrationStatus

Defines the IMS registration status parameters

Enumerator

UNKOWN_STATE Unknown status for IMS

NOT_REGISTERED Not registered status for IMS

REGISTERING Registering status for IMS

REGISTERED Registered status for IMS

LIMITED_REGISTERED Limited registration status for IMS

6.12 telux::therm Namespace Reference

Data Structures

- class [ThermalFactory](#)
ThermalFactory allows creation of thermal manager. [More...](#)
- struct [BoundCoolingDevice](#)
- class [IThermalManager](#)
IThermalManager provides interface to get thermal zone and cooling device information. [More...](#)
- class [ITripPoint](#)
ITripPoint provides interface to get trip point type, trip point temperature and hysteresis value for that trip point. [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 [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 [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...](#)

6.12.1 Typedef Documentation

6.12.1.1 using telux::therm::GetAutoShutdownModeResponseCb = typedef std::function<void(AutoShutdownMode mode)>

This function is called with the response to getAutoShutdownMode API.

The callback can be invoked from multiple different threads. The implementation should be thread safe.

Parameters

in	mode	AutoShutdownMode

6.13 telux::wlan Namespace Reference

Data Structures

- struct [ApNetConfig](#)
- struct [ApConfig](#)
- struct [DeviceIndInfo](#)
- struct [DeviceInfo](#)
- struct [DeviceStats](#)
- class [IApInterfaceManager](#)
- class [IApListener](#)
- struct [StaStaticIpConfig](#)
- struct [StaConfig](#)
- class [IStaInterfaceManager](#)

Manager class for configuring Wlan Access Points. [More...](#)

Manager class for configuring Wlan Station Mode. [More...](#)

- class [IStaListener](#)
- struct [ApInfo](#)
- struct [ApNetInfo](#)
- struct [ApStatus](#)
- struct [StaStatus](#)
- struct [InterfaceStatus](#)
- class [IWlanDeviceManager](#)

WlanDeviceManager is a primary interface for configuring Wireless LAN. it provide APIs to enable, configure, activate, and modify modes. [More...](#)

- class [IWlanListener](#)
- class [WlanFactory](#)

WlanFactory is the central factory to create all wlan classes. [More...](#)

7 Data Structure Documentation

7.1 telux::tel::CellBroadcastMessage Class Reference

Cell Broadcast message.

Public member functions

- [CellBroadcastMessage](#) (std::shared_ptr< [EtwsInfo](#) > etwsInfo)
- [CellBroadcastMessage](#) (std::shared_ptr< [CmasInfo](#) > cmasInfo)
- [MessageType](#) getMessageType () const
- std::shared_ptr< [EtwsInfo](#) > getEtwsInfo () const
- std::shared_ptr< [CmasInfo](#) > getCmasInfo () const

Cell Broadcast message.

7.1.1 Constructors and Destructors

7.1.1.1 telux::tel::CellBroadcastMessage::CellBroadcastMessage (std::shared_ptr< [EtwsInfo](#) > *etwsInfo*)

[CellBroadcastMessage](#) constructor.

Parameters

in	<i>etwsInfo</i>	EtwsInfo

7.1.1.2 telux::tel::CellBroadcastMessage::CellBroadcastMessage (std::shared_ptr< [CmasInfo](#) > *cmasInfo*)

[CellBroadcastMessage](#) constructor.

Parameters

in	<i>cmasInfo</i>	CmasInfo

7.1.2 Member Function Documentation

7.1.2.1 `MessageType telux::tel::CellBroadcastMessage::getMessageType ( ) const`

Get the cellbroadcast message type.

Returns

[MessageType](#).

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.1.2.2 `std::shared_ptr<EtwsInfo> telux::tel::CellBroadcastMessage::getEtwsInfo ( ) const`

Get ETWS warning notification containing information about the ETWS warning type, the emergency user alert flag and the popup flag. This method should be called only if message type returned by [getMessageType\(\)](#) API is `MessageType::ETWS`

Returns

pointer to [EtwsInfo](#) or null if this is not an ETWS warning notification

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.1.2.3 `std::shared_ptr<CmasInfo> telux::tel::CellBroadcastMessage::getCmasInfo ( ) const`

Get CMAS warning notification containing information about the CMAS message class, severity, urgency and certainty. This method should be called only if message type returned by [getMessageType\(\)](#) API is `MessageType::CMAS`

Returns

pointer to [CmasInfo](#) or null if this is not a CMAS warning notification

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.2 `telux::tel::Circle` Class Reference

Public member functions

- [Circle](#) ([Point](#) center, double radius)
- [Point](#) `getCenter` ()
- double `getRadius` ()

This class represents a geometry represented as simple circle.

7.2.1 Constructors and Destructors**7.2.1.1 `telux::tel::Circle::Circle ( Point center, double radius )`**

[Circle](#) constructor.

Parameters

in	<i>center</i>	Center of circle represented by telux::tel::Point
in	<i>radius</i>	Radius of circle in meters

7.2.2 Member Function Documentation**7.2.2.1 `Point telux::tel::Circle::getCenter ( )`**

Get center point of circle.

Returns

Center of circle.

7.2.2.2 `double telux::tel::Circle::getRadius ( )`

Get radius of circle.

Returns

Radius of circle.

7.3 `telux::tel::CmasInfo` Class Reference**Public member functions**

- [CmasInfo](#) ([GeographicalScope](#) geographicalScope, int msgId, int serialNumber, std::string languageCode, std::string messageText, [MessagePriority](#) priority, [CmasMessageClass](#) messageClass, [CmasSeverity](#) severity, [CmasUrgency](#) urgency, [CmasCertainty](#) certainty, std::shared_ptr<[WarningAreaInfo](#)> warningAreaInfo)
- [GeographicalScope](#) `getGeographicalScope` () const
- int `getMessageId` () const

- `int getSerialNumber () const`
- `std::string getLanguageCode () const`
- `std::string getMessageBody () const`
- `MessagePriority getPriority () const`
- `int getMessageCode () const`
- `int getUpdateNumber () const`
- `CmasMessageClass getMessageClass ()`
- `CmasSeverity getSeverity ()`
- `CmasUrgency getUrgency ()`
- `CmasCertainty getCertainty ()`
- `std::shared_ptr< WarningAreaInfo > getWarningAreaInfo ()`

Contains information elements for a GSM/UMTS/E-UTRAN/NG-RAN CMAS warning notification. Supported values for each element are defined in 3GPP TS 23.041.

7.3.1 Constructors and Destructors

7.3.1.1 `telux::tel::CmasInfo::CmasInfo (GeographicalScope geographicalScope, int msgId, int serialNumber, std::string languageCode, std::string messageText, MessagePriority priority, CmasMessageClass messageClass, CmasSeverity severity, CmasUrgency urgency, CmasCertainty certainty, std::shared_ptr< WarningAreaInfo > warningAreaInfo)`

`CmasInfo` constructor.

Parameters

<code>in</code>	<code>geographicalScope</code>	<code>GeographicalScope</code>
<code>in</code>	<code>msgId</code>	Unique message identifier
<code>in</code>	<code>serialNumber</code>	Serial number for message
<code>in</code>	<code>languageCode</code>	ISO-639-1 language code for message
<code>in</code>	<code>messageText</code>	Message text
<code>in</code>	<code>priority</code>	<code>MessagePriority</code>
<code>in</code>	<code>messageClass</code>	<code>CmasMessageClass</code>
<code>in</code>	<code>severity</code>	<code>CmasSeverity</code>
<code>in</code>	<code>urgency</code>	<code>CmasUrgency</code>
<code>in</code>	<code>certainty</code>	<code>CmasCertainty</code>

7.3.2 Member Function Documentation

7.3.2.1 GeographicalScope telux::tel::CmasInfo::getGeographicalScope () const

Get the geographicalScope of cellbroadcast message.

Returns

GeographicalScope.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.3.2.2 int telux::tel::CmasInfo::getMessageId () const

Get cellbroadcast message identifier. The message identifier identifies the type of the cell broadcast message defined in spec 3GPP TS 23.041 9.4.1.2.2

Returns

Message identifier.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.3.2.3 int telux::tel::CmasInfo::getSerialNumber () const

Get the serial number of broadcast (geographical scope + message code + update number for GSM/UMTS).

Returns

int containing cellbroadcast serial number.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.3.2.4 std::string telux::tel::CmasInfo::getLanguageCode () const

Get the ISO-639-1 language code for cell broadcast message, or empty string if unspecified. This is not applicable for ETWS primary notification.

Returns

Language code

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.3.2.5 std::string telux::tel::CmasInfo::getMessageBody () const

Get the body of cell broadcast message, or empty string if no body available. For ETWS primary notification based on message identifier pre canned message will be sent.

Returns

body or empty string

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.3.2.6 MessagePriority telux::tel::CmasInfo::getPriority () const

Get the priority for the cell broadcast message.

Returns

[MessagePriority](#).

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.3.2.7 int telux::tel::CmasInfo::getMessageCode () const

Get the cellbroadcast message code.

Returns

int containing cellbroadcast message code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.3.2.8 `int telux::tel::CmasInfo::getUpdateNumber ( ) const`

Get the cellbroadcast message update number.

Returns

int containing cellbroadcast message's update number.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.3.2.9 `CmasMessageClass telux::tel::CmasInfo::getMessageClass ( )`

Get CMAS message class.

Returns

[CmasMessageClass](#).

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.3.2.10 `CmasSeverity telux::tel::CmasInfo::getSeverity ( )`

Get CMAS message severity.

Returns

[CmasSeverity](#).

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.3.2.11 `CmasUrgency telux::tel::CmasInfo::getUrgency ( )`

Get CMAS message urgency.

Returns

[CmasUrgency](#).

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.3.2.12 CmasCertainty telux::tel::CmasInfo::getCertainty ()

Get CMAS message certainty.

Returns

[CmasCertainty](#).

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.3.2.13 std::shared_ptr<WarningAreaInfo> telux::tel::CmasInfo::getWarningAreaInfo ()

Returns warning area information for alert. This is applicable for LTE and NR5G

Returns

pointer to [WarningAreaInfo](#) or null if there is no warning area information available.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.4 telux::tel::EtwsInfo Class Reference

Public member functions

- [EtwsInfo](#) ([GeographicalScope](#) geographicalScope, int msgId, int serialNumber, std::string languageCode, std::string messageText, [MessagePriority](#) priority, [EtwsWarningType](#) warningType, bool emergencyUserAlert, bool activatePopup, bool primary, std::vector< uint8_t > warningSecurityInformation)
- [GeographicalScope](#) getGeographicalScope () const
- int getMessageId () const
- int getSerialNumber () const
- std::string getLanguageCode () const
- std::string getMessageBody () const
- [MessagePriority](#) getPriority () const
- int getMessageCode () const
- int getUpdateNumber () const
- [EtwsWarningType](#) getEtwsWarningType ()

- bool [isEmergencyUserAlert](#) ()
- bool [isPopupAlert](#) ()
- bool [isPrimary](#) ()
- std::vector< uint8_t > [getWarningSecurityInformation](#) ()

Contains information elements for a GSM/UMTS/E-UTRAN/NG-RAN ETWS warning notification.

Supported values for each element are defined in 3GPP TS 23.041.

7.4.1 Constructors and Destructors

7.4.1.1 `telux::tel::EtwsInfo::EtwsInfo (GeographicalScope geographicalScope, int msgId, int serialNumber, std::string languageCode, std::string messageText, MessagePriority priority, EtwsWarningType warningType, bool emergencyUserAlert, bool activatePopup, bool primary, std::vector< uint8_t > warningSecurityInformation)`

[EtwsInfo](#) constructor.

Parameters

in	<i>geographicalScope</i>	GeographicalScope
in	<i>msgId</i>	Unique message identifier
in	<i>serialNumber</i>	Serial number for message
in	<i>languageCode</i>	ISO-639-1 language code for message
in	<i>messageText</i>	Message text
in	<i>priority</i>	MessagePriority
in	<i>warningType</i>	EtwsWarningType
in	<i>emergencyUserAlert</i>	If true message is emergency user alert otherwise not
in	<i>activatePopup</i>	If true message message activate popup flag is set, otherwise popup flag is false.
in	<i>primary</i>	If true ETWS message is primary notification otherwise not
in	<i>warningSecurity-Information</i>	Buffer containing security information about ETWS primary notification such as timestamp and digital signature

7.4.2 Member Function Documentation

7.4.2.1 `GeographicalScope telux::tel::EtwsInfo::getGeographicalScope ( ) const`

Get the `geographicalScope` of cellbroadcast message.

Returns

[GeographicalScope](#).

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.4.2.2 int telux::tel::EtwsInfo::getMessageId () const

Get cellbroadcast message identifier. The message identifier identifies the type of the cell broadcast message defined in spec 3GPP TS 23.041 9.4.1.2.2

Returns

Message identifier.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.4.2.3 int telux::tel::EtwsInfo::getSerialNumber () const

Get the serial number of broadcast (geographical scope + message code + update number for GSM/UMTS).

Returns

int containing cellbroadcast serial number.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.4.2.4 std::string telux::tel::EtwsInfo::getLanguageCode () const

Get the ISO-639-1 language code for cell broadcast message, or empty string if unspecified. This is not applicable for ETWS primary notification.

Returns

Language code

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.4.2.5 `std::string telux::tel::EtwsInfo::getMessageBody ( ) const`

Get the body of cell broadcast message, or empty string if no body available. For ETWS primary notification based on message identifier pre canned message will be sent.

Returns

body or empty string

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.4.2.6 `MessagePriority telux::tel::EtwsInfo::getPriority ( ) const`

Get the priority for the cell broadcast message.

Returns

[MessagePriority](#).

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.4.2.7 `int telux::tel::EtwsInfo::getMessageCode ( ) const`

Get the cellbroadcast message code.

Returns

int containing cellbroadcast message code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.4.2.8 `int telux::tel::EtwsInfo::getUpdateNumber ( ) const`

Get the cellbroadcast message update number.

Returns

int containing cellbroadcast message's update number.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.4.2.9 EtwsWarningType telux::tel::EtwsInfo::getEtwsWarningType ()

Get ETWS warning type.

Returns

[EtwsWarningType](#).

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.4.2.10 bool telux::tel::EtwsInfo::isEmergencyUserAlert ()

Returns the ETWS emergency user alert flag.

Returns

true to notify terminal to activate emergency user alert or false otherwise

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.4.2.11 bool telux::tel::EtwsInfo::isPopupAlert ()

Returns the ETWS activate popup flag.

Returns

true to notify terminal to activate display popup or false otherwise

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.4.2.12 bool telux::tel::EtwsInfo::isPrimary ()

Returns the ETWS format flag. This flag determine whether ETWS message is primary notification or not.

Returns

true if the message is primary message, otherwise secondary message

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.4.2.13 `std::vector<uint8_t> telux::tel::EtwsInfo::getWarningSecurityInformation ( )`

Returns security information about ETWS primary notification such as timestamp and digital signature(applicable only for GSM).

Returns

byte buffer

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.5 `telux::tel::Geometry` Class Reference**Public member functions**

- [Geometry](#) (std::shared_ptr< [Polygon](#) > polygon)
- [Geometry](#) (std::shared_ptr< [Circle](#) > circle)
- [GeometryType](#) `getType ()` const
- std::shared_ptr< [Polygon](#) > `getPolygon ()` const
- std::shared_ptr< [Circle](#) > `getCircle ()` const

This class represents warning area geometry to perform geofencing on alert.

7.5.1 Constructors and Destructors**7.5.1.1 `telux::tel::Geometry::Geometry ( std::shared_ptr< Polygon > polygon )`**

[Geometry](#) constructor.

Parameters

in	<i>polygon</i>	Polygon

7.5.1.2 `telux::tel::Geometry::Geometry ( std::shared_ptr< Circle > circle )`

[Geometry](#) constructor.

in	circle	Circle

7.5.2 Member Function Documentation

7.5.2.1 GeometryType telux::tel::Geometry::getType () const

Get the geometry type.

Returns

[GeometryType](#).

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.5.2.2 std::shared_ptr<Polygon> telux::tel::Geometry::getPolygon () const

Get polygon geometry as warning area to perform geofencing. This method should be called only if geometry type returned by [getType\(\)](#) API is GeometryType::POLYGON

Returns

[Polygon](#) geometry object.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.5.2.3 std::shared_ptr<Circle> telux::tel::Geometry::getCircle () const

Get circle geometry as warning area to perform geofencing. This method should be called only if geometry type returned by [getType\(\)](#) API is GeometryType::CIRCLE

Returns

[Circle](#) geometry object.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.6 telux::tel::ICellBroadcastListener Class Reference

A listener class which monitors cell broadcast messages.

Public member functions

- virtual void [onIncomingMessage](#) (SlotId slotId, const std::shared_ptr< [CellBroadcastMessage](#) > cbMessage)
- virtual void [onMessageFilterChange](#) (SlotId slotId, std::vector< [CellBroadcastFilter](#) > filters)
- virtual [~ICellBroadcastListener](#) ()

A listener class which monitors cell broadcast messages.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

7.6.1 Constructors and Destructors

7.6.1.1 virtual telux::tel::ICellBroadcastListener::~~ICellBroadcastListener ()
[virtual]

7.6.2 Member Function Documentation

7.6.2.1 virtual void telux::tel::ICellBroadcastListener::onIncomingMessage (SlotId *slotId*, const std::shared_ptr< [CellBroadcastMessage](#) > *cbMessage*)
[virtual]

This function is called when device receives an incoming cell broadcast message.

Parameters

in	<i>slotId</i>	- Slot Id on which broadcast message is received.
in	<i>cbMessage</i>	- Broadcast message with information related to ETWS/CMAS notification.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.6.2.2 virtual void telux::tel::ICellBroadcastListener::onMessageFilterChange (SlotId *slotId*, std::vector< [CellBroadcastFilter](#) > *filters*) [virtual]

This function is called when there is change in broadcast configuration like updation of message filters by the client using [ICellBroadcastManager::updateMessageFilters](#).

Parameters

in	<i>slotId</i>	- Slot Id on which change in message filters is received.
in	<i>filters</i>	- Complete list of configured broadcast message filters.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.7 telux::tel::ICellBroadcastManager Class Reference

CellBroadcastManager class is primary interface to configure and activate emergency broadcast messages and receive broadcast messages.

Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual SlotId [getSlotId](#) ()=0
- virtual [telux::common::Status](#) [updateMessageFilters](#) (std::vector< [CellBroadcastFilter](#) > filters, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [requestMessageFilters](#) ([RequestFiltersResponseCallback](#) callback)=0
- virtual [telux::common::Status](#) [setActivationStatus](#) (bool activate, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [requestActivationStatus](#) ([RequestActivationStatusResponseCallback](#) callback)=0
- virtual [telux::common::Status](#) [registerListener](#) (std::weak_ptr< [ICellBroadcastListener](#) > listener)=0
- virtual [telux::common::Status](#) [deregisterListener](#) (std::weak_ptr< [ICellBroadcastListener](#) > listener)=0
- virtual [~ICellBroadcastManager](#) ()

CellBroadcastManager class is primary interface to configure and activate emergency broadcast messages and receive broadcast messages.

7.7.1 Constructors and Destructors

7.7.1.1 virtual [telux::tel::ICellBroadcastManager::~ICellBroadcastManager](#) ()
[virtual]

7.7.2 Member Function Documentation

7.7.2.1 virtual bool [telux::tel::ICellBroadcastManager::isSubsystemReady](#) () [pure virtual]

Checks the status of network subsystem and returns the result.

Returns

True if network subsystem is ready for service otherwise false.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.7.2.2 **virtual std::future<bool> telux::tel::lCellBroadcastManager::onSubsystemReady () [pure virtual]**

Wait for network subsystem to be ready.

Returns

A future that caller can wait on to be notified when network subsystem is ready.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.7.2.3 **virtual SlotId telux::tel::lCellBroadcastManager::getSlotId () [pure virtual]**

Get associated slot for this CellBroadcastManager.

Returns

SlotId

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.7.2.4 **virtual telux::common::Status telux::tel::lCellBroadcastManager::updateMessageFilters (std::vector< CellBroadcastFilter > *filters*, telux::common::ResponseCallback *callback* = *nullptr*) [pure virtual]**

Configures the broadcast messages to be received.

Parameters

in	<i>filters</i>	List of filtered broadcast message identifiers.
in	<i>callback</i>	Optional callback to get the response of set cell broadcast filters.

Returns

Status of updateMessageIdFilters i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.7.2.5 virtual telux::common::Status telux::tel::lCellBroadcastManager::request-MessageFilters (RequestFiltersResponseCallback *callback*) [pure virtual]

Retrieve configured message filters for which broadcast messages will be received.

Parameters

in	<i>callback</i>	Callback to get the response of get cell broadcast filters.

Returns

Status of requestMessageIdFilters i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.7.2.6 virtual telux::common::Status telux::tel::lCellBroadcastManager::set-ActivationStatus (bool *activate*, telux::common::ResponseCallback *callback* = nullptr) [pure virtual]

Allows activation and deactivation of configured broadcast messages.

Parameters

in	<i>activate</i>	Activate/deactivate broadcast messages.
in	<i>callback</i>	Optional callback pointer to get the response.

Returns

Status of setActivationStatus i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.7.2.7 virtual telux::common::Status telux::tel::lCellBroadcastManager::request-ActivationStatus (RequestActivationStatusResponseCallback *callback*) [pure virtual]

Get activation status for configured broadcast messages.

in	<i>callback</i>	Callback pointer to get the response.

Returns

Status of requestActivationStatus i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.7.2.8 **virtual telux::common::Status telux::tel::ICellBroadcastManager::register-Listener (std::weak_ptr< ICellBroadcastListener > *listener*) [pure virtual]**

Register a listener for cell broadcast messages.

Parameters

in	<i>listener</i>	Pointer to ICellBroadcastListener object which receives broadcast message.

Returns

Status of registerListener i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.7.2.9 **virtual telux::common::Status telux::tel::ICellBroadcastManager::deregister-Listener (std::weak_ptr< ICellBroadcastListener > *listener*) [pure virtual]**

De-register the listener.

Parameters

in	<i>listener</i>	Listener to be de-registered

Returns

Status of deregisterListener i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.8 telux::cv2x::ICv2xListener Class Reference

Cv2x Radio Manager listeners implement this interface.

Public member functions

- virtual void [onStatusChanged](#) ([Cv2xStatus](#) status)
- virtual void [onStatusChanged](#) ([Cv2xStatusEx](#) status)
- virtual [~ICv2xListener](#) ()

Cv2x Radio Manager listeners implement this interface.

7.8.1 Constructors and Destructors

7.8.1.1 virtual telux::cv2x::ICv2xListener::~~ICv2xListener () [virtual]

Destructor for [ICv2xListener](#)

7.8.2 Member Function Documentation

7.8.2.1 virtual void telux::cv2x::ICv2xListener::onStatusChanged ([Cv2xStatus](#) *status*) [virtual]

Called when the status of the CV2X radio has changed.

Parameters

in	<i>status</i>	- CV2X radio status.

Deprecated use [onStatusChanged\(Cv2xStatusEx status\)](#)

7.8.2.2 virtual void telux::cv2x::ICv2xListener::onStatusChanged ([Cv2xStatusEx](#) *status*) [virtual]

Called when the status of the CV2X radio has changed.

Parameters

in	<i>status</i>	- CV2X radio status.

7.9 telux::data::IEspFilter Class Reference

This class represents a IP Filter for the ESP, get the new instance from [telux::data::DataFactory](#).

Public member functions

- virtual [EspInfo](#) `getEspInfo ()`=0
- virtual `telux::common::Status` `setEspInfo` (const [EspInfo](#) &espInfo)=0
- virtual `~IEspFilter` ()

This class represents a IP Filter for the ESP, get the new instance from [telux::data::DataFactory](#).

7.9.1 Constructors and Destructors**7.9.1.1 virtual `telux::data::IEspFilter::~~IEspFilter ( )` [virtual]**

Destructor for [IEspFilter](#)

7.9.2 Member Function Documentation**7.9.2.1 virtual `EspInfo` `telux::data::IEspFilter::getEspInfo ( )` [pure virtual]**

Get the ESP header info

Returns

[telux::data::EspInfo](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.9.2.2 virtual `telux::common::Status` `telux::data::IEspFilter::setEspInfo ( const EspInfo & espInfo )` [pure virtual]

sets the ICMP header info

Parameters

in	<i>espInfo</i>	EspInfo structure telux::data::EspInfo

Returns

Immediate status of [setEspInfo\(\)](#) request sent i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.10 telux::data::IcmpFilter Class Reference

This class represents a IP Filter for the ICMP, get the new instance from [telux::data::DataFactory](#).

Public member functions

- virtual [IcmpInfo](#) [getIcmpInfo](#) ()=0
- virtual [telux::common::Status](#) [setIcmpInfo](#) (const [IcmpInfo](#) &icmpInfo)=0
- virtual [~IcmpFilter](#) ()

This class represents a IP Filter for the ICMP, get the new instance from [telux::data::DataFactory](#).

7.10.1 Constructors and Destructors

7.10.1.1 virtual telux::data::IcmpFilter::~~IcmpFilter () [virtual]

Destructor for [IcmpFilter](#)

7.10.2 Member Function Documentation

7.10.2.1 virtual IcmpInfo telux::data::IcmpFilter::getIcmpInfo () [pure virtual]

Get the ICMP header info

Returns

[telux::data::IcmpInfo](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.10.2.2 virtual telux::common::Status telux::data::IcmpFilter::setIcmpInfo (const IcmpInfo & icmpInfo) [pure virtual]

sets the ICMP header info

Parameters

in	<i>icmpInfo</i>	TcpInfo structure telux::data::IcmpInfo

Returns

Immediate status of [setIcmpInfo\(\)](#) request sent i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.11 telux::tel::IImsservingSystemListener Class Reference

A listener class for monitoring changes in IMS Serving System manager, including IMS registration status change. Override the methods for the state that you wish to receive updates for.

Public member functions

- virtual void [onServiceStatusChange](#) ([telux::common::ServiceStatus](#) status)
- virtual void [onImssRegStatusChange](#) ([ImssRegistrationInfo](#) status)
- virtual [~IImsservingSystemListener](#) ()

A listener class for monitoring changes in IMS Serving System manager, including IMS registration status change. Override the methods for the state that you wish to receive updates for.

The methods in listener can be invoked from multiple different threads. The implementation should be thread safe.

7.11.1 Constructors and Destructors

7.11.1.1 virtual [telux::tel::IImsservingSystemListener::~~IImsservingSystemListener](#) (
) [[virtual](#)]

7.11.2 Member Function Documentation

7.11.2.1 virtual void [telux::tel::IImsservingSystemListener::onServiceStatusChange](#) (
 [telux::common::ServiceStatus](#) *status*) [[virtual](#)]

This function is called when service status changes.

Parameters

in	<i>status</i>	- telux::common::ServiceStatus

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

Reimplemented from [telux::common::IServiceStatusListener](#).

7.11.2.2 **virtual void telux::tel::ImsServingSystemListener::onImsRegStatusChange (ImsRegistrationInfo *status*) [virtual]**

This function is called whenever any IMS service configuration is changed.

<i>in</i>	<i>status</i>	Indicates which registration status is the IMS service changed to. telux::tel::ImsRegistrationInfo .

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.12 telux::tel::ImsServingSystemManager Class Reference

IMS Serving System Manager is the primary interface for IMS related operations Allows to query IMS registration status.

Public member functions

- virtual [telux::common::ServiceStatus getServiceStatus \(\)](#)=0
- virtual [telux::common::Status requestRegistrationInfo \(ImsRegistrationInfoCb callback\)](#)=0
- virtual [telux::common::Status registerListener \(std::weak_ptr< telux::tel::ImsServingSystemListener > listener\)](#)=0
- virtual [telux::common::Status deregisterListener \(std::weak_ptr< telux::tel::ImsServingSystemListener > listener\)](#)=0
- virtual [~ImsServingSystemManager \(\)](#)

IMS Serving System Manager is the primary interface for IMS related operations Allows to query IMS registration status.

7.12.1 Constructors and Destructors

7.12.1.1 **virtual telux::tel::ImsServingSystemManager::~~ImsServingSystemManager () [virtual]**

7.12.2 Member Function Documentation

7.12.2.1 **virtual telux::common::ServiceStatus telux::tel::IImServingSystemManager::getServiceStatus () [pure virtual]**

This status indicates whether the [IImServingSystemManager](#) object is in a usable state.

Returns

SERVICE_AVAILABLE - If IMS Serving System manager is ready for service.

SERVICE_UNAVAILABLE - If IMS Serving System manager is temporarily unavailable.

SERVICE_FAILED - If IMS Serving System manager encountered an irrecoverable failure.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.12.2.2 **virtual telux::common::Status telux::tel::IImServingSystemManager::requestRegistrationInfo (ImsRegistrationInfoCb *callback*) [pure virtual]**

Request IMS registration information.

Parameters

in	<i>callback</i>	Callback pointer to get the response of requestRegistrationInfo.

Returns

Status of requestRegistrationInfo i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.12.2.3 **virtual telux::common::Status telux::tel::IImServingSystemManager::registerListener (std::weak_ptr< telux::tel::IImServingSystemListener > *listener*) [pure virtual]**

Add a listener to listen for specific events in the IMS Serving System subsystem.

Parameters

in	<i>listener</i>	Pointer to IImServingSystemListener object that processes the notification

Returns

Status of registerListener i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.12.2.4 **virtual telux::common::Status telux::tel::llmsServingSystemManager- ::deregisterListener (std::weak_ptr< telux::tel::llmsServingSystemListener > *listener*) [pure virtual]**

Remove a previously added listener.

Parameters

<i>in</i>	<i>listener</i>	Listener to be removed.

Returns

Status of deregisterListener i.e. success or suitable error code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.13 telux::data::llpFilter Class Reference

A IP filter class to add specific filters like what data will be allowed from the modem to the application processor. Only data packets that match the filter will be sent to the apps processor. Also used to configure Firewall rules.

Public member functions

- virtual [IPv4Info](#) getIPv4Info ()=0
- virtual [telux::common::Status](#) setIPv4Info (const [IPv4Info](#) &ipv4Info)=0
- virtual [IPv6Info](#) getIPv6Info ()=0
- virtual [telux::common::Status](#) setIPv6Info (const [IPv6Info](#) &ipv6Info)=0
- virtual [IpProtocol](#) getIpProtocol ()=0
- virtual [IpFamilyType](#) getIpFamily ()=0
- virtual [~llpFilter](#) ()

A IP filter class to add specific filters like what data will be allowed from the modem to the application processor. Only data packets that match the filter will be sent to the apps processor. Also used to configure Firewall rules.

7.13.1 Constructors and Destructors

7.13.1.1 virtual telux::data::IIPFilter::~~IIPFilter () [virtual]

Destructor for [IIPFilter](#)

7.13.2 Member Function Documentation

7.13.2.1 virtual IPv4Info telux::data::IIPFilter::getIPv4Info () [pure virtual]

Get the IPv4 header info

Returns

[telux::data::IPv4Info](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.13.2.2 virtual telux::common::Status telux::data::IIPFilter::setIPv4Info (const IPv4Info & *ipv4Info*) [pure virtual]

sets the IPv4 header info

Parameters

in	<i>ipv4Info</i>	IPv4 structure telux::data::IPv4Info

Returns

Immediate status of [setIPv4Info\(\)](#) request sent i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.13.2.3 virtual IPv6Info telux::data::IIPFilter::getIPv6Info () [pure virtual]

Get the IPv6 header info

Returns

[telux::data::IPv6Info](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.13.2.4 **virtual telux::common::Status telux::data::llpFilter::setIPv6Info (const IPv6Info & *ipv6Info*) [pure virtual]**

sets the IPv6 header info

Parameters

in	<i>ipv6Info</i>	IPv6 structure telux::data::IPv6Info

Returns

Immediate status of [setIPv6Info\(\)](#) request sent i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.13.2.5 **virtual IpProtocol telux::data::llpFilter::getIpProtocol () [pure virtual]**

Get the IpProtocol Number

Returns

[telux::data::IpProtocol](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.13.2.6 **virtual IpFamilyType telux::data::llpFilter::getIpFamily () [pure virtual]**

Get the IP family type

Returns

[telux::data::IpFamilyType](#)

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.14 **telux::loc::ILocationConfigurator Class Reference**

[ILocationConfigurator](#) allows general engine configurations (example: TUNC, PACE etc), configuration of specific engines like SPE (example: minSVElevation, minGPSWeek etc) or DRE, deletion of warm and

cold aiding data, NMEA configuration and support for XTRA feature. [ILocationConfigurator](#) APIs strictly adheres to the principle of single client per process.

Public Types

- using [GetMinGpsWeekCallback](#) = std::function< void(uint16_t minGpsWeek, [telux::common::ErrorCode](#) error)>
- using [GetMinSVElevationCallback](#) = std::function< void(uint8_t minSVElevation, [telux::common::ErrorCode](#) error)>
- using [GetRobustLocationCallback](#) = std::function< void(const [telux::loc::RobustLocationConfiguration](#) rLConfig, [telux::common::ErrorCode](#) error)>
- using [GetXtraStatusCallback](#) = std::function< void(const [telux::loc::XtraStatus](#) xtraStatus, [telux::common::ErrorCode](#) error)>

Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status](#) [configureCTunc](#) (bool enable, [telux::common::ResponseCallback](#) callback=nullptr, float timeUncertainty=DEFAULT_TUNC_THRESHOLD, uint32_t energyBudget=DEFAULT_TUNC_ENERGY_THRESHOLD)=0
- virtual [telux::common::Status](#) [configurePACE](#) (bool enable, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [deleteAllAidingData](#) ([telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [configureLeverArm](#) (const [LeverArmConfigInfo](#) &info, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [configureConstellations](#) (const [SvBlackList](#) &list, [telux::common::ResponseCallback](#) callback=nullptr, bool resetToDefault=false)=0
- virtual [telux::common::Status](#) [configureRobustLocation](#) (bool enable, bool enableForE911=false, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [requestRobustLocation](#) ([GetRobustLocationCallback](#) cb)=0
- virtual [telux::common::Status](#) [configureMinGpsWeek](#) (uint16_t minGpsWeek, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [requestMinGpsWeek](#) ([GetMinGpsWeekCallback](#) cb)=0
- virtual [telux::common::Status](#) [configureMinSVElevation](#) (uint8_t minSVElevation, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [requestMinSVElevation](#) ([GetMinSVElevationCallback](#) cb)=0
- virtual [telux::common::Status](#) [deleteAidingData](#) ([AidingData](#) aidingDataMask, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [configureDR](#) (const [DREngineConfiguration](#) &config,

`telux::common::ResponseCallback callback=nullptr)=0`

- virtual `telux::common::Status configureNmeaTypes` (const `NmeaSentenceConfig` nmeaType, `telux::common::ResponseCallback` callback=nullptr)=0
- virtual `telux::common::Status configureNmea` (const `NmeaConfig` configParams, `telux::common::ResponseCallback` callback=nullptr)=0
- virtual `telux::common::Status provideConsentForTerrestrialPositioning` (bool userConsent, `telux::common::ResponseCallback` callback=nullptr)=0
- virtual `telux::common::Status registerLocationInjector` (std::weak_ptr< `ILocationInjectionListener` > listener)=0
- virtual `telux::common::Status deregisterLocationInjector` (std::weak_ptr< `ILocationInjectionListener` > listener)=0
- virtual `telux::common::Status injectLocationData` (const `ExternalLocationInfo` &location)=0
- virtual `telux::common::Status configureEngineIntegrityRisk` (const `EngineType` engineType, uint32_t integrityRisk, `telux::common::ResponseCallback` callback=nullptr)=0
- virtual `telux::common::Status configureXtraParams` (bool enable, const `XtraConfig` configParams, `telux::common::ResponseCallback` callback=nullptr)=0
- virtual `telux::common::Status requestXtraStatus` (`GetXtraStatusCallback` callback)=0
- virtual `telux::common::Status registerListener` (`LocConfigIndications` indicationList, std::weak_ptr< `ILocationConfigListener` > listener)=0
- virtual `telux::common::Status deRegisterListener` (`LocConfigIndications` indicationList, std::weak_ptr< `ILocationConfigListener` > listener)=0
- virtual `~ILocationConfigurator` ()

`ILocationConfigurator` allows general engine configurations (example: TUNC, PACE etc), configuration of specific engines like SPE (example: minSVElevation, minGPSWeek etc) or DRE, deletion of warm and cold aiding data, NMEA configuration and support for XTRA feature. `ILocationConfigurator` APIs strictly adheres to the principle of single client per process.

7.14.1 Member Typedef Documentation

7.14.1.1 `using telux::loc::ILocationConfigurator::GetMinGpsWeekCallback = std::function<void(uint16_t minGpsWeek, telux::common::ErrorCode error)>`

This function is called with the response to requestMinGpsWeek API.

Parameters

in	<i>minGpsWeek</i>	- minimum gps week.
in	<i>error</i>	- Return code which indicates whether the operation succeeded or not.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.14.1.2 using telux::loc::ILocationConfigurator::GetMinSVElevationCallback = std::function<void(uint8_t minSVElevation, telux::common::ErrorCode error)>

This function is called with the response to requestMinSVElevation API.

Parameters

in	<i>minSVElevation</i>	- minimum SV Elevation angle in units of degree.
in	<i>error</i>	- Return code which indicates whether the operation succeeded or not.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.14.1.3 using telux::loc::ILocationConfigurator::GetRobustLocationCallback = std::function<void(const telux::loc:: RobustLocationConfiguration rLConfig, telux::common::ErrorCode error)>

This function is called with the response to requestRobustLocation API.

Parameters

in	<i>rLConfig</i>	- robust location settings information.
in	<i>error</i>	- Return code which indicates whether the operation succeeded or not.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.14.1.4 using telux::loc::ILocationConfigurator::GetXtraStatusCallback = std::function<void(const telux::loc::XtraStatus xtraStatus, telux::common::ErrorCode error)>

This function is called with the response to requestXtraStatus API.

Parameters

in	<i>xtraStatus</i>	- Information pertaining to Xtra assistance data.
in	<i>error</i>	- Return code which indicates whether the operation succeeded or not.

7.15 telux::loc::ILocationManager Class Reference

[ILocationManager](#) provides interface to register and remove listeners. It also allows to set and get configuration/ criteria for position reports. The new APIs(registerListenerEx, deRegisterListenerEx, startDetailedReports, startBasicReports) and old/deprecated APIs(registerListener, removeListener, setPositionReportTimeout, setHorizontalAccuracyLevel, setMinIntervalForReports) should not be used interchangeably, either the new APIs should be used or the old APIs should be used.

Public Types

- using [GetEnergyConsumedCallback](#) = std::function< void([telux::loc::GnssEnergyConsumedInfo](#) energyConsumed, [telux::common::ErrorCode](#) error)>
- using [GetTerrestrialInfoCallback](#) = std::function< void(const std::shared_ptr< [ILocationInfoBase](#) > terrestrialInfo)>

Public member functions

- virtual bool [isSubsystemReady](#) ()=0
- virtual std::future< bool > [onSubsystemReady](#) ()=0
- virtual [telux::common::Status](#) [registerListenerEx](#) (std::weak_ptr< [ILocationListener](#) > listener)=0
- virtual [telux::common::Status](#) [deRegisterListenerEx](#) (std::weak_ptr< [ILocationListener](#) > listener)=0
- virtual [telux::common::Status](#) [startDetailedReports](#) (uint32_t interval, [telux::common::ResponseCallback](#) callback=nullptr, [GnssReportTypeMask](#) reportMask=DEFAULT_GNSS_REPORT)=0
- virtual [telux::common::Status](#) [startDetailedEngineReports](#) (uint32_t interval, [LocReqEngine](#) engineType, [telux::common::ResponseCallback](#) callback=nullptr, [GnssReportTypeMask](#) reportMask=DEFAULT_GNSS_REPORT)=0
- virtual [telux::common::Status](#) [startBasicReports](#) (uint32_t distanceInMeters, uint32_t intervalInMs, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [registerForSystemInfoUpdates](#) (std::weak_ptr< [ILocationSystemInfoListener](#) > listener, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [deRegisterForSystemInfoUpdates](#) (std::weak_ptr< [ILocationSystemInfoListener](#) > listener, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [requestEnergyConsumedInfo](#) ([GetEnergyConsumedCallback](#) cb)=0
- virtual [telux::common::Status](#) [stopReports](#) ([telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [getTerrestrialPosition](#) (uint32_t timeoutMsec, [TerrestrialTechnology](#) techMask, [GetTerrestrialInfoCallback](#) cb, [telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [telux::common::Status](#) [cancelTerrestrialPositionRequest](#) ([telux::common::ResponseCallback](#) callback=nullptr)=0
- virtual [~ILocationManager](#) ()

[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.

7.15.1 Member Typedef Documentation

7.15.1.1 using telux::loc::ILocationManager::GetEnergyConsumedCallback = std::function<void(telux::loc::GnssEnergyConsumedInfo energyConsumed, telux::common::ErrorCode error)>

This function is called with the response to getEnergyConsumedInfoUpdate API.

Parameters

in	<i>energyConsumed</i>	- Information regarding energy consumed by Gnss engine.
in	<i>error</i>	- Return code which indicates whether the operation succeeded or not.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.15.1.2 using telux::loc::ILocationManager::GetTerrestrialInfoCallback = std::function<void(const std::shared_ptr<ILocationInfoBase> terrestrialInfo)>

This function is called with the response to getTerrestrialPosition API.

Parameters

in	<i>terrestrialInfo</i>	- basic position related information.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.16 telux::common::IServiceStatusListener Class Reference

Public member functions

- virtual void [onServiceStatusChange](#) ([ServiceStatus](#) status)
- virtual [~IServiceStatusListener](#) ()

7.16.1 Constructors and Destructors

7.16.1.1 virtual telux::common::IServiceStatusListener::~~IServiceStatusListener () [virtual]

7.16.2 Member Function Documentation

7.16.2.1 virtual void telux::common::IServiceStatusListener::onServiceStatusChange (ServiceStatus *status*) [virtual]

This function is called when service status changes.

Parameters

in	<i>status</i>	- ServiceStatus

Reimplemented in [telux::tel::ImsServingSystemListener](#).

7.17 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](#).

7.17.1 Constructors and Destructors

7.17.1.1 virtual telux::data::ITcpFilter::~~ITcpFilter () [virtual]

Destructor for [ITcpFilter](#)

7.17.2 Member Function Documentation

7.17.2.1 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.

7.17.2.2 virtual telux::common::Status telux::data::ITcpFilter::setTcpInfo (const TcpInfo & tcpInfo) [pure virtual]

sets the TCP header info

Parameters

in	tcpInfo	TcpInfo structure telux::data::TcpInfo

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.

7.18 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](#).

7.18.1 Constructors and Destructors

7.18.1.1 virtual telux::data::IUdpFilter::~~IUdpFilter () [virtual]

Destructor for [IUdpFilter](#)

7.18.2 Member Function Documentation

7.18.2.1 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.

7.18.2.2 **virtual telux::common::Status telux::data::UdpFilter::setUdpInfo (const UdpInfo & *udpInfo*) [pure virtual]**

sets the UDP header info

Parameters

in	<i>udpInfo</i>	UdpInfo structure telux::data::UdpInfo

Returns

Immediate status of [setUdpInfo\(\)](#) request sent i.e. success or suitable status code.

Note

Eval: This is a new API and is being evaluated. It is subject to change and could break backwards compatibility.

7.19 telux::tel::Polygon Class Reference

Public member functions

- [Polygon](#) (std::vector< [Point](#) > vertices)
- std::vector< [Point](#) > [getVertices](#) ()

This class represents a simple polygon with different points.

7.19.1 Constructors and Destructors

7.19.1.1 **telux::tel::Polygon::Polygon (std::vector< Point > *vertices*)**

[Polygon](#) constructor.

Parameters

in	<i>vertices</i>	List of telux::tel::Point

7.19.2 Member Function Documentation

7.19.2.1 `std::vector<Point> telux::tel::Polygon::getVertices ( )`

Get vertices of polygon.

Returns

List of [telux::tel::Point](#).

7.20 `telux::tel::WarningAreaInfo` Class Reference

Public member functions

- [WarningAreaInfo](#) (int maxWaitTime, std::vector< [Geometry](#) > geometries)
- int [getGeoFenceMaxWaitTime](#) ()
- std::vector< [Geometry](#) > [getGeometries](#) ()

This class represents warning area information for alert.

7.20.1 Constructors and Destructors

7.20.1.1 `telux::tel::WarningAreaInfo::WarningAreaInfo ( int maxWaitTime, std::vector< Geometry > geometries )`

Warning Area Information constructor.

Parameters

in	<i>maxWaitTime</i>	Maximum wait time allowed to determine position for alert Range is 0 to 255 where 0 means Zero wait time, 1 - 254 is Geo-Fencing Maximum Wait Time in seconds and 255 means use device default wait time.
in	<i>geometries</i>	Geometries to perform geofencing on alert

7.20.2 Member Function Documentation

7.20.2.1 `int telux::tel::WarningAreaInfo::getGeoFenceMaxWaitTime ( )`

Get maximum wait time allowed to determine position for alert.

Returns

Maximum wait time for alert in seconds.

7.20.2.2 `std::vector<Geometry> telux::tel::WarningAreaInfo::getGeometries ( )`

Get geometries to perform geofencing on alert.

Returns

List of [telux::tel::Geometry](#).

LEGAL INFORMATION

Your access to and use of this material, along with any documents, software, specifications, reference board files, drawings, diagnostics and other information contained herein (collectively this “Material”), is subject to your (including the corporation or other legal entity you represent, collectively “You” or “Your”) acceptance of the terms and conditions (“Terms of Use”) set forth below. If You do not agree to these Terms of Use, you may not use this Material and shall immediately destroy any copy thereof.

1) Legal Notice.

This Material is being made available to You solely for Your internal use with those products and service offerings of Qualcomm Technologies, Inc. (“Qualcomm Technologies”), its affiliates and/or licensors described in this Material, and shall not be used for any other purposes. If this Material is marked as “Qualcomm Internal Use Only”, no license is granted to You herein, and You must immediately (a) destroy or return this Material to Qualcomm Technologies, and (b) report Your receipt of this Material to qualcomm.support@qti.qualcomm.com. This Material may not be altered, edited, or modified in any way without Qualcomm Technologies’ prior written approval, nor may it be used for any machine learning or artificial intelligence development purpose which results, whether directly or indirectly, in the creation or development of an automated device, program, tool, algorithm, process, methodology, product and/or other output. Unauthorized use or disclosure of this Material or the information contained herein is strictly prohibited, and You agree to indemnify Qualcomm Technologies, its affiliates and licensors for any damages or losses suffered by Qualcomm Technologies, its affiliates and/or licensors for any such unauthorized uses or disclosures of this Material, in whole or part.

Qualcomm Technologies, its affiliates and/or licensors retain all rights and ownership in and to this Material. No license to any trademark, patent, copyright, mask work protection right or any other intellectual property right is either granted or implied by this Material or any information disclosed herein, including, but not limited to, any license to make, use, import or sell any product, service or technology offering embodying any of the information in this Material.

THIS MATERIAL IS BEING PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE. TO THE MAXIMUM EXTENT PERMITTED BY LAW, QUALCOMM TECHNOLOGIES, ITS AFFILIATES AND/OR LICENSORS SPECIFICALLY DISCLAIM ALL WARRANTIES OF TITLE, MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, COMPLETENESS OR ACCURACY, AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MOREOVER, NEITHER QUALCOMM TECHNOLOGIES, NOR ANY OF ITS AFFILIATES AND/OR LICENSORS, SHALL BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY EXPENSES, LOSSES, USE, OR ACTIONS HOWSOEVER INCURRED OR UNDERTAKEN BY YOU IN RELIANCE ON THIS MATERIAL.

Certain product kits, tools and other items referenced in this Material may require You to accept additional terms and conditions before accessing or using those items.

Technical data specified in this Material may be subject to U.S. and other applicable export control laws. Transmission contrary to U.S. and any other applicable law is strictly prohibited.

Nothing in this Material is an offer to sell any of the components or devices referenced herein.

This Material is subject to change without further notification.

In the event of a conflict between these Terms of Use and the Website Terms of Use on www.qualcomm.com, the *Qualcomm Privacy Policy* referenced on www.qualcomm.com, or other legal statements or notices found on prior pages of the Material, these Terms of Use will control. In the event of a conflict between these Terms of Use and any other agreement (written or click-through, including, without limitation any non-disclosure agreement) executed by You and Qualcomm Technologies or a Qualcomm Technologies affiliate and/or licensor with respect to Your access to and use of this Material, the other agreement will control.

These Terms of Use shall be governed by and construed and enforced in accordance with the laws of the State of California, excluding the U.N. Convention on International Sale of Goods, without regard to conflict of laws principles. Any dispute, claim or controversy arising out of or relating to these Terms of Use, or the breach or validity hereof, shall be adjudicated only by a court of competent jurisdiction in the county of San Diego, State of California, and You hereby consent to the personal jurisdiction of such courts for that purpose.

2) Trademark and Product Attribution Statements.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the U.S. and/or elsewhere. The Bluetooth® word mark is a registered trademark owned by Bluetooth SIG, Inc. Other product and brand names referenced in this Material may be trademarks or registered trademarks of their respective owners.

Snapdragon and Qualcomm branded products referenced in this Material are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.