TelephonyManager | API reference | Android Developers

Archived: 2026-04-05 13:37:21 UTC

TelephonyManager Stay organized with collections Save and categorize content based on your preferences.

`public class TelephonyManager`
`extends Object`

Provides access to information about the telephony services on the device. Applications can use the methods in this class to determine telephony services and states, as well as to access some types of subscriber information. Applications can also register a listener to receive notification of telephony state changes.

The returned TelephonyManager will use the default subscription for all calls. To call an API for a specific subscription, use `createForSubscriptionId(int)`. e.g. `telephonyManager = defaultSubTelephonyManager.createForSubscriptionId(subId);`

Note that access to some telephony information is permission-protected. Your application cannot access the protected information unless it has the appropriate permissions declared in its manifest file. Where permissions apply, they are noted in the methods through which you access the protected information.

TelephonyManager is intended for use on devices that implement `FEATURE_TELEPHONY`. On devices that do not implement this feature, the behavior is not reliable.
Requires the `PackageManager#FEATURE_TELEPHONY` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Summary

Public methods	
`boolean`	`canChangeDtmfToneLength()` Whether the device supports configuring the DTMF tone length.
`void`	`clearSignalStrengthUpdateRequest(SignalStrengthUpdateRequest request)` Clear a `SignalStrengthUpdateRequest` from the system.
`TelephonyManager`	`createForPhoneAccountHandle(PhoneAccountHandle phoneAccountHandle)` Create a new TelephonyManager object pinned to the subscription ID associated with the given phone account.
`TelephonyManager`	`createForSubscriptionId(int subId)`

	Create a new TelephonyManager object pinned to the given subscription ID.
`boolean`	`doesSwitchMultiSimConfigTriggerReboot()` Get whether making changes to modem configurations by `switchMultiSimConfig(int)` will trigger device reboot.
`int`	`getActiveModemCount()` Returns the number of logical modems currently configured to be activated.
`List<CellInfo>`	`getAllCellInfo()` Requests all available cell information from all radios on the device including the camped/registered, serving, and neighboring cells.
`long`	`getAllowedNetworkTypesForReason(int reason)` Get the allowed network types for certain reason.
`int`	`getCallComposerStatus()` Get the user-set status for enriched calling with call composer.
`int`	`getCallState()` *This method was deprecated in API level 31. Use `getCallStateForSubscription()` to retrieve the call state for a specific telephony subscription (which allows carrier privileged apps), `TelephonyCallback.CallStateListener` for real-time call state updates, or `TelecomManager.isInCall()`, which supplies an aggregate "in call" state for the entire device.*
`int`	`getCallStateForSubscription()` Retrieve the call state for a specific subscription that was specified when this TelephonyManager instance was created.
`int`	`getCardIdForDefaultEuicc()` Get the card ID of the default eUICC card.
`PersistableBundle`	`getCarrierConfig()` Returns the carrier config of the subscription ID pinned to the TelephonyManager.
`int`	`getCarrierIdFromSimMccMnc()` Returns carrier id based on sim MCCMNC (returned by `getSimOperator()`) only.
`void`	`getCarrierRestrictionStatus(Executor executor, Consumer<Integer> resultListener)`

	Get the carrier restriction status of the device.
`CellLocation`	`getCellLocation()` *This method was deprecated in API level 26. use `getAllCellInfo()` instead, which returns a superset of this API.*
`int`	`getCurrentTtyMode()` Returns the current TTY mode of the device.
`int`	`getDataActivity()` Returns a constant indicating the type of activity on a data connection (cellular).
`int`	`getDataNetworkType()` Returns a constant indicating the radio technology (network type) currently in use on t device for data transmission.
`int`	`getDataState()` Returns a constant indicating the current data connection state (cellular).
`String`	`getDeviceId()` *This method was deprecated in API level 26. Use `getImei()` which returns IMEI for GSM or `getMeid()` which returns MEID for CDMA.*
`String`	`getDeviceId(int slotIndex)` *This method was deprecated in API level 26. Use `getImei()` which returns IMEI for GSM or `getMeid()` which returns MEID for CDMA.*
`String`	`getDeviceSoftwareVersion()` Returns the software version number for the device, for example, the IMEI/SV for GS phones.
`Map<Integer, List<EmergencyNumber>>`	`getEmergencyNumberList()` Get the emergency number list based on current locale, sim, default, modem and network.
`Map<Integer, List<EmergencyNumber>>`	`getEmergencyNumberList(int categories)` Get the per-category emergency number list based on current locale, sim, default, modem and network.
`List<String>`	`getEquivalentHomePlmns()`

	Returns a list of the equivalent home PLMNs (EF_EHPLMN) from the USIM app.
`String[]`	`getForbiddenPlmns()` Returns an array of Forbidden PLMNs from the USIM App Returns null if the query fails.
`String`	`getGroupIdLevel1()` Returns the Group Identifier Level1 for a GSM phone.
`String`	`getIccAuthentication(int appType, int authType, String data)` Returns the response of authentication for the default subscription.
`String`	`getImei(int slotIndex)` Returns the IMEI (International Mobile Equipment Identity).
`String`	`getImei()` Returns the IMEI (International Mobile Equipment Identity).
`String`	`getLine1Number()` *This method was deprecated in API level 33. use* *`SubscriptionManager.getPhoneNumber(int)`* *instead.*
`String`	`getManualNetworkSelectionPlmn()` Get the PLMN chosen for Manual Network Selection if active.
`String`	`getManufacturerCode(int slotIndex)` *This method was deprecated in API level 36. Legacy CDMA is unsupported.*
`String`	`getManufacturerCode()` *This method was deprecated in API level 36. Legacy CDMA is unsupported.*
`static long`	`getMaximumCallComposerPictureSize()` Indicates the maximum size of the call composure picture.
`String`	`getMeid()` *This method was deprecated in API level 36. Legacy CDMA is unsupported.*

String	getMeid(int slotIndex) *This method was deprecated in API level 36. Legacy CDMA is unsupported.*
String	getMmsUAProfUrl() Returns the MMS user agent profile URL.
String	getMmsUserAgent() Returns the MMS user agent.
String	getNai() Returns the Network Access Identifier (NAI).
String	getNetworkCountryIso() Returns the ISO-3166-1 alpha-2 country code equivalent of the MCC (Mobile Country Code) of the current registered operator or the cell nearby, if available.
String	getNetworkCountryIso(int slotIndex) Returns the ISO-3166-1 alpha-2 country code equivalent of the MCC (Mobile Country Code) of the current registered operator or the cell nearby, if available.
String	getNetworkOperator() Returns the numeric name (MCC+MNC) of current registered operator.
String	getNetworkOperatorName() Returns the alphabetic name of current registered operator.
int	getNetworkSelectionMode() Get the network selection mode.
void	getNetworkSlicingConfiguration(Executor executor, OutcomeReceiver<NetworkSlicingConfig, TelephonyManager.NetworkSlicingExceptio callback) Request to get the current slicing configuration including URSP rules and NSSAIs (configured, allowed and rejected).
String	getNetworkSpecifier() Returns the network specifier of the subscription ID pinned to the TelephonyManager.
int	getNetworkType()

	This method was deprecated in API level 30. use `getDataNetworkType()`
`PhoneAccountHandle`	`getPhoneAccountHandle()` Determines the `PhoneAccountHandle` associated with this TelephonyManager.
`int`	`getPhoneCount()` *This method was deprecated in API level 30. Use* `getActiveModemCount()` *instead.*
`int`	`getPhoneType()` Returns a constant indicating the device phone type.
`int`	`getPreferredOpportunisticDataSubscription()` Get preferred opportunistic data subscription Id Requires that the calling app has carrier privileges (see `hasCarrierPrivileges()`), o has either READ_PRIVILEGED_PHONE_STATE or `READ_PHONE_STATE` permission
`String`	`getPrimaryImei()` Returns the primary IMEI (International Mobile Equipment Identity) of the device as mentioned in GSMA TS.37.
`ServiceState`	`getServiceState(int includeLocationData)` Returns the current `ServiceState` information.
`ServiceState`	`getServiceState()` Returns the current `ServiceState` information.
`ServiceState`	`getServiceStateForSlot(int slotIndex)` Returns the service state information on specified SIM slot.
`SignalStrength`	`getSignalStrength()` Get the most recently available signal strength information.
`int`	`getSimCarrierId()` Returns carrier id of the current subscription.
`CharSequence`	`getSimCarrierIdName()` Returns carrier id name of the current subscription.
`String`	`getSimCountryIso()`

	Returns the ISO-3166-1 alpha-2 country code equivalent for the SIM provider's countı code.
`String`	`getSimOperator()` Returns the MCC+MNC (mobile country code + mobile network code) of the provide of the SIM.
`String`	`getSimOperatorName()` Returns the Service Provider Name (SPN).
`String`	`getSimSerialNumber()` Returns the serial number of the SIM, if applicable.
`int`	`getSimSpecificCarrierId()` Returns fine-grained carrier ID of the current subscription.
`CharSequence`	`getSimSpecificCarrierIdName()` Similar like `getSimCarrierIdName()`, returns user-facing name of the specific carrier returned by `getSimSpecificCarrierId()`.
`int`	`getSimState()` Returns a constant indicating the state of the default SIM card.
`int`	`getSimState(int slotIndex)` Returns a constant indicating the state of the device SIM card in a logical slot.
`String`	`getSubscriberId()` Returns the unique subscriber ID, for example, the IMSI for a GSM phone.
`int`	`getSubscriptionId(PhoneAccountHandle phoneAccountHandle)` Returns the subscription ID for the given phone account handle.
`int`	`getSubscriptionId()` Return an appropriate subscription ID for any situation.
`int`	`getSupportedModemCount()` Return how many logical modem can be potentially active simultaneously, in terms of hardware capability.

`long`	`getSupportedRadioAccessFamily()` Requires android.Manifest.permission.READ_PRIVILEGED_PHONE_STATE.
`String`	`getTypeAllocationCode()` Returns the Type Allocation Code from the IMEI.
`String`	`getTypeAllocationCode(int slotIndex)` Returns the Type Allocation Code from the IMEI.
`List<UiccCardInfo>`	`getUiccCardsInfo()` Gets information about currently inserted UICCs (Universal Integrated Circuit Cards) and eUICCs (embedded UICCs).
`String`	`getVisualVoicemailPackageName()` Returns the package responsible of processing visual voicemail for the subscription ID pinned to the TelephonyManager.
`String`	`getVoiceMailAlphaTag()` Retrieves the alphabetic identifier associated with the voice mail number.
`String`	`getVoiceMailNumber()` Returns the voice mail number.
`int`	`getVoiceNetworkType()` Returns the NETWORK_TYPE_xxxx for voice Requires Permission: `READ_PHONE_STATE` or `READ_BASIC_PHONE_STATE` or that the calling app has carrier privileges (see `hasCarrierPrivileges()`).
`Uri`	`getVoicemailRingtoneUri(PhoneAccountHandle accountHandle)` Returns the URI for the per-account voicemail ringtone set in Phone settings.
`boolean`	`hasCarrierPrivileges()` Has the calling application been granted carrier privileges by the carrier.
`boolean`	`hasIccCard()` This API is used to check if there is an ICC card present in the device.
`boolean`	`iccCloseLogicalChannel(int channel)`

	Closes a previously opened logical channel to the ICC card.
`byte[]`	`iccExchangeSimIO(int fileID, int command, int p1, int p2, int p3, String filePath)` Returns the response APDU for a command APDU sent through SIM_IO.
`IccOpenLogicalChannelResponse`	`iccOpenLogicalChannel(String AID, int p2)` Opens a logical channel to the ICC card.
`IccOpenLogicalChannelResponse`	`iccOpenLogicalChannel(String AID)` *This method was deprecated in API level 26. Replaced by* *`iccOpenLogicalChannel(String,int)`*
`String`	`iccTransmitApduBasicChannel(int cla, int instruction, int p1, int p2, int p3 String data)` Transmit an APDU to the ICC card over the basic channel.
`String`	`iccTransmitApduLogicalChannel(int channel, int cla, int instruction, int p1, int p2, int p3, String data)` Transmit an APDU to the ICC card over a logical channel.
`boolean`	`isConcurrentVoiceAndDataSupported()` Whether the device is currently on a technology (e.g. UMTS or LTE) which can suppo voice and data simultaneously.
`boolean`	`isDataCapable()`
`boolean`	`isDataConnectionAllowed()` Checks whether cellular data connection is allowed in the device.
`boolean`	`isDataEnabled()` Returns whether mobile data is enabled or not per user setting.
`boolean`	`isDataEnabledForReason(int reason)` Return whether data is enabled for certain reason .
`boolean`	`isDataRoamingEnabled()` Returns whether mobile data roaming is enabled on the subscription.
`boolean`	`isDeviceSmsCapable()`

`boolean`	`isDeviceVoiceCapable()`
`boolean`	`isEmergencyNumber(String number)` Identifies if the supplied phone number is an emergency number that matches a known emergency number based on current locale, SIM card(s), Android database, modem, network, or defaults.
`boolean`	`isHearingAidCompatibilitySupported()` Whether the phone supports hearing aid compatibility.
`boolean`	`isManualNetworkSelectionAllowed()` Checks if manual network selection is allowed.
`boolean`	`isModemEnabledForSlot(int slotIndex)` Indicates whether or not there is a modem stack enabled for the given SIM slot.
`int`	`isMultiSimSupported()` Returns if the usage of multiple SIM cards at the same time to register on the network (e.g. Dual Standby or Dual Active) is supported by the device and by the carrier.
`boolean`	`isNetworkRoaming()` Returns true if the device is considered roaming on the current network, for GSM purposes.
`boolean`	`isPremiumCapabilityAvailableForPurchase(int capability)` Check whether the given premium capability is available for purchase from the carrier
`boolean`	`isRadioInterfaceCapabilitySupported(String capability)` Whether the device supports a given capability on the radio interface.
`boolean`	`isRttSupported()` Determines whether the device currently supports RTT (Real-time text).
`boolean`	`isSmsCapable()` *This method was deprecated in API level 35. Replaced by `isDeviceSmsCapable()`. Starting from Android 15, SMS capability may also be overridden by carriers for a given subscription. For SMS capable device (when `isDeviceSmsCapable()` return `true`), caller should check for subscription-level SMS capability as well. See `isDeviceSmsCapable()` for details.*
`boolean`	`isTtyModeSupported()`

	This method was deprecated in API level 28. Use `TelecomManager.isTtySupported()` *instead Whether the phone supports TTY mode.*
`boolean`	`isVoiceCapable()` *This method was deprecated in API level 35. Replaced by* `isDeviceVoiceCapable()` *. Starting from Android 15, voice capability may also be overridden by carriers for a given subscription. For voice capable device (when* `isDeviceVoiceCapable()` *return* `true` *), caller should check for subscription-level voice capability as well. See* `isDeviceVoiceCapable()` *for details.*
`boolean`	`isVoicemailVibrationEnabled(PhoneAccountHandle accountHandle)` Returns whether vibration is set for voicemail notification in Phone settings.
`boolean`	`isWorldPhone()` Whether the device is a world phone.
`void`	`listen(PhoneStateListener listener, int events)` *This method was deprecated in API level 31. Use* `registerTelephonyCallback(Executor,TelephonyCallback)` *.*
`void`	`purchasePremiumCapability(int capability, Executor executor, Consumer<Intege callback)` Purchase the given premium capability from the carrier.
`void`	`rebootModem()` Reboot and re-initialize the cellular modem and related subsystems below the OS.
`void`	`registerTelephonyCallback(Executor executor, TelephonyCallback callback)` Registers a callback object to receive notification of changes in specified telephony states.
`void`	`registerTelephonyCallback(int includeLocationData, Executor executor, TelephonyCallback callback)` Registers a callback object to receive notification of changes in specified telephony states.
`void`	`requestCellInfoUpdate(Executor executor, TelephonyManager.CellInfoCallback callback)` Requests all available cell information from the current subscription for observed camped/registered, serving, and neighboring cells.

NetworkScan	requestNetworkScan(NetworkScanRequest request, Executor executor, TelephonyScanManager.NetworkScanCallback callback) Request a network scan.
NetworkScan	requestNetworkScan(int includeLocationData, NetworkScanRequest request, Executor executor, TelephonyScanManager.NetworkScanCallback callback) Request a network scan.
void	sendDialerSpecialCode(String inputCode) Send the special dialer code.
String	sendEnvelopeWithStatus(String content) Send ENVELOPE to the SIM and return the response.
void	sendUssdRequest(String ussdRequest, TelephonyManager.UssdResponseCallback callback, Handler handler) Sends an Unstructured Supplementary Service Data (USSD) request to the mobile network and informs the caller of the response via the supplied callback.
void	sendVisualVoicemailSms(String number, int port, String text, PendingIntent sentIntent) Send a visual voicemail SMS.
void	setAllowedNetworkTypesForReason(int reason, long allowedNetworkTypes) Set the allowed network types of the device and provide the reason triggering the allowed network change.
void	setCallComposerStatus(int status) Set the user-set status for enriched calling with call composer.
void	setDataEnabled(boolean enable) *This method was deprecated in API level 31. use setDataEnabledForReason with reas DATA_ENABLED_REASON_USER instead.*
void	setDataEnabledForReason(int reason, boolean enabled) Control of data connection and provide the reason triggering the data connection control.

`int`	`setForbiddenPlmns(List<String> fplmns)` Replace the contents of the forbidden PLMN SIM file with the provided values.
`boolean`	`setLine1NumberForDisplay(String alphaTag, String number)` *This method was deprecated in API level 33. use* `SubscriptionManager.setCarrierPhoneNumber(int,String)` *instead.*
`void`	`setNetworkSelectionModeAutomatic()` Sets the network selection mode to automatic.
`boolean`	`setNetworkSelectionModeManual(String operatorNumeric, boolean persistSelection, int ran)` Ask the radio to connect to the input network and change selection mode to manual.
`boolean`	`setNetworkSelectionModeManual(String operatorNumeric, boolean persistSelection)` Ask the radio to connect to the input network and change selection mode to manual.
`boolean`	`setOperatorBrandOverride(String brand)` Override the branding for the current ICCID.
`boolean`	`setPreferredNetworkTypeToGlobal()` Set the preferred network type to global mode which includes NR, LTE, CDMA, EvD and GSM/WCDMA.
`void`	`setPreferredOpportunisticDataSubscription(int subId, boolean needValidation, Executor executor, Consumer<Integer> callback)` Set preferred opportunistic data subscription id.
`void`	`setSignalStrengthUpdateRequest(SignalStrengthUpdateRequest request)` Set a `SignalStrengthUpdateRequest` to receive notification when signal quality measurements breach the specified thresholds.
`void`	`setVisualVoicemailSmsFilterSettings(VisualVoicemailSmsFilterSettings settings)` Set the visual voicemail SMS filter settings for the subscription ID pinned to the TelephonyManager.

`boolean`	`setVoiceMailNumber(String alphaTag, String number)` Sets the voice mail number.
`void`	`setVoicemailRingtoneUri(PhoneAccountHandle phoneAccountHandle, Uri uri)` *This method was deprecated in API level 28. Use* `Settings.ACTION_CHANNEL_NOTIFICATION_SETTINGS` *instead.*
`void`	`setVoicemailVibrationEnabled(PhoneAccountHandle phoneAccountHandle, boolean enabled)` *This method was deprecated in API level 28. Use* `Settings.ACTION_CHANNEL_NOTIFICATION_SETTINGS` *instead.*
`void`	`switchMultiSimConfig(int numOfSims)` Switch configs to enable multi-sim or switch back to single-sim Requires Permission: `MODIFY_PHONE_STATE` or that the calling app has carrier privileg (see `hasCarrierPrivileges()`).
`void`	`unregisterTelephonyCallback(TelephonyCallback callback)` Unregister an existing `TelephonyCallback` .
`void`	`updateAvailableNetworks(List<AvailableNetworkInfo> availableNetworks, Execut executor, Consumer<Integer> callback)` Update availability of a list of networks in the current location.
`void`	`uploadCallComposerPicture(InputStream pictureToUpload, String contentType, Executor executor, OutcomeReceiver<ParcelUuid, TelephonyManager.CallComposerException> callback)` Uploads a picture to the carrier network for use with call composer.
`void`	`uploadCallComposerPicture(Path pictureToUpload, String contentType, Executor executor, OutcomeReceiver<ParcelUuid, TelephonyManager.CallComposerException> callback)` Uploads a picture to the carrier network for use with call composer.

Inherited methods

From class `java.lang.Object`

`Object`	`clone()` Creates and returns a copy of this object.

`boolean`	`equals(Object obj)` Indicates whether some other object is "equal to" this one.
`void`	`finalize()` Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
`final Class<?>`	`getClass()` Returns the runtime class of this `Object` .
`int`	`hashCode()` Returns a hash code value for the object.
`final void`	`notify()` Wakes up a single thread that is waiting on this object's monitor.
`final void`	`notifyAll()` Wakes up all threads that are waiting on this object's monitor.
`String`	`toString()` Returns a string representation of the object.
`final void`	`wait(long timeoutMillis, int nanos)` Causes the current thread to wait until it is awakened, typically by being *notified* or *interrupted*, or until a certain amount of real time has elapsed.
`final void`	`wait(long timeoutMillis)` Causes the current thread to wait until it is awakened, typically by being *notified* or *interrupted*, or until a certain amount of real time has elapsed.
`final void`	`wait()` Causes the current thread to wait until it is awakened, typically by being *notified* or *interrupted*.

Constants

ACTION_CARRIER_MESSAGING_CLIENT_SERVICE

```
public static final String ACTION_CARRIER_MESSAGING_CLIENT_SERVICE
```

A service action that identifies a `CarrierMessagingClientService` subclass in the AndroidManifest.xml.

See `CarrierMessagingClientService` for the details.

Constant Value: "android.telephony.action.CARRIER_MESSAGING_CLIENT_SERVICE"

ACTION_CARRIER_SIGNAL_DEFAULT_NETWORK_AVAILABLE

```
public static final String ACTION_CARRIER_SIGNAL_DEFAULT_NETWORK_AVAILABLE
```

Broadcast action sent when the availability of the system default network changes.

See also:

- `This action is intended for carrier apps to set/reset carrier actions. It is only sent to the carrier apps specified in the carrier config for the subscription ID attached to this intent. The intent will have the following extra values:`
  - `{@link #EXTRA_DEFAULT_NETWORK_AVAILABLE}`
    `{@code true} if the default network is now available, {@code false} otherwise.`
  - `{@link android.telephony.SubscriptionManager#EXTRA_SUBSCRIPTION_INDEX SubscriptionManager.EXTRA_SUBSCRIPTION_INDEX}`
    `The subscription ID on which the default network availability changed.`

  `This is a protected intent that can only be sent by the system.`

Constant Value: "android.telephony.action.CARRIER_SIGNAL_DEFAULT_NETWORK_AVAILABLE"

ACTION_CARRIER_SIGNAL_PCO_VALUE

```
public static final String ACTION_CARRIER_SIGNAL_PCO_VALUE
```

Broadcast action sent when a PCO value becomes available from the modem. This action is intended for sim/account status checks and only sent to the carrier apps specified in the carrier config for the subscription ID that's attached to this intent. The intent will have the following extra values:

- `EXTRA_APN_TYPE`
  An integer indicating the apn type.
- `EXTRA_APN_PROTOCOL`
  An integer indicating the protocol of the apn connection
- `EXTRA_PCO_ID`
  An integer indicating the PCO id for the data.
- `EXTRA_PCO_VALUE`
  A byte array of PCO data read from modem.
- `SubscriptionManager.EXTRA_SUBSCRIPTION_INDEX`
  The subscription ID for which the PCO info was received.

This is a protected intent that can only be sent by the system.

Constant Value: "android.telephony.action.CARRIER_SIGNAL_PCO_VALUE"

ACTION_CARRIER_SIGNAL_REDIRECTED

```
public static final String ACTION_CARRIER_SIGNAL_REDIRECTED
```

Broadcast action sent when a data connection is redirected with validation failure. This action is intended for sim/account status checks and only sent to the carrier apps specified in the carrier config for the subscription ID that's attached to this intent. The intent will have the following extra values:

- `EXTRA_APN_TYPE`
  An integer indicating the apn type.
- `EXTRA_REDIRECTION_URL`
  A string indicating the redirection url
- `SubscriptionManager.EXTRA_SUBSCRIPTION_INDEX`
  The subscription ID on which the validation failure happened.

This is a protected intent that can only be sent by the system.

Constant Value: "android.telephony.action.CARRIER_SIGNAL_REDIRECTED"

ACTION_CARRIER_SIGNAL_REQUEST_NETWORK_FAILED

```
public static final String ACTION_CARRIER_SIGNAL_REQUEST_NETWORK_FAILED
```

Broadcast action sent when a data connection setup fails. This action is intended for sim/account status checks and only sent to the carrier apps specified in the carrier config for the subscription ID that's attached to this intent. The intent will have the following extra values:

- `EXTRA_APN_TYPE`
  An integer indicating the apn type.
- `EXTRA_DATA_FAIL_CAUSE`
  A integer indicating the data fail cause.
- `SubscriptionManager.EXTRA_SUBSCRIPTION_INDEX`
  The subscription ID on which the data setup failure happened.

This is a protected intent that can only be sent by the system.

Constant Value: "android.telephony.action.CARRIER_SIGNAL_REQUEST_NETWORK_FAILED"

ACTION_CARRIER_SIGNAL_RESET

```
public static final String ACTION_CARRIER_SIGNAL_RESET
```

Broadcast action sent when carrier apps should reset their internal state. Sent when certain events such as turning on/off mobile data, removing the SIM, etc. require carrier apps to reset their state. This action is intended to signal carrier apps to perform cleanup operations. It is only sent to the carrier apps specified in the carrier config for the subscription ID attached to this intent. The intent will have the following extra values:

- `SubscriptionManager.EXTRA_SUBSCRIPTION_INDEX`
  The subscription ID for which state should be reset.

This is a protected intent that can only be sent by the system.

Constant Value: "android.telephony.action.CARRIER_SIGNAL_RESET"

ACTION_CONFIGURE_VOICEMAIL

```
public static final String ACTION_CONFIGURE_VOICEMAIL
```

Open the voicemail settings activity to make changes to voicemail configuration.

The `EXTRA_PHONE_ACCOUNT_HANDLE` extra indicates which `PhoneAccountHandle` to configure voicemail. The `EXTRA_HIDE_PUBLIC_SETTINGS` hides settings the dialer will modify through public API if set.

Constant Value: "android.telephony.action.CONFIGURE_VOICEMAIL"

ACTION_MULTI_SIM_CONFIG_CHANGED

```
public static final String ACTION_MULTI_SIM_CONFIG_CHANGED
```

Broadcast action to be received by Broadcast receivers. Indicates multi-SIM configuration is changed. For example, it changed from single SIM capable to dual-SIM capable (DSDS or DSDA) or triple-SIM mode. It doesn't indicate how many subscriptions are actually active, or which states SIMs are, or that all steps during multi-SIM change are done. To know those information you still need to listen to SIM_STATE changes or active subscription changes. See extra of `EXTRA_ACTIVE_SIM_SUPPORTED_COUNT` for updated value.

Constant Value: "android.telephony.action.MULTI_SIM_CONFIG_CHANGED"

ACTION_NETWORK_COUNTRY_CHANGED

```
public static final String ACTION_NETWORK_COUNTRY_CHANGED
```

Broadcast intent action for network country code changes.

The `EXTRA_NETWORK_COUNTRY` extra indicates the country code of the current network returned by `getNetworkCountryIso()` .

There may be a delay of several minutes before reporting that no country is detected.

Constant Value: "android.telephony.action.NETWORK_COUNTRY_CHANGED"

ACTION_RESET_MOBILE_NETWORK_SETTINGS

```
public static final String ACTION_RESET_MOBILE_NETWORK_SETTINGS
```

Activity action: Show setting to reset mobile networks.

On devices with a settings activity to reset mobile networks, the activity should be launched without additional permissions.

On some devices, this settings activity may not exist. Callers should ensure that this case is appropriately handled.

Constant Value: "android.telephony.action.RESET_MOBILE_NETWORK_SETTINGS"

ACTION_RESPOND_VIA_MESSAGE

```
public static final String ACTION_RESPOND_VIA_MESSAGE
```

The Phone app sends this intent when a user opts to respond-via-message during an incoming call. By default, the device's default SMS app consumes this message and sends a text message to the caller. A third party app can also provide this functionality by consuming this Intent with a `Service` and sending the message using its own messaging system.

The intent contains a URI (available from `Intent.getData()`) describing the recipient, using either the `sms:`, `smsto:`, `mms:`, or `mmsto:` URI schema. Each of these URI schema carry the recipient information the same way: the path part of the URI contains the recipient's phone number or a comma-separated set of phone numbers if there are multiple recipients. For example, `smsto:2065551234`.

The intent may also contain extras for the message text (in `Intent.EXTRA_TEXT`) and a message subject (in `Intent.EXTRA_SUBJECT`).

Note: The intent-filter that consumes this Intent needs to be in a `Service` that requires the permission `Manifest.permission.SEND_RESPOND_VIA_MESSAGE`.

For example, the service that receives this intent can be declared in the manifest file with an intent filter like this:

```
<!-- Service that delivers SMS messages received from the phone "quick response" -->
<service android:name=".HeadlessSmsSendService"
         android:permission="android.permission.SEND_RESPOND_VIA_MESSAGE"
         android:exported="true" >
  <intent-filter>
    <action android:name="android.intent.action.RESPOND_VIA_MESSAGE" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:scheme="sms" />
    <data android:scheme="smsto" />
    <data android:scheme="mms" />
    <data android:scheme="mmsto" />
  </intent-filter>
</service>
```

Output: nothing.

Constant Value: "android.intent.action.RESPOND_VIA_MESSAGE"

ACTION_SECRET_CODE

```
public static final String ACTION_SECRET_CODE
```

Broadcast Action: A debug code has been entered in the dialer.

This intent is broadcast by the system and OEM telephony apps may need to receive these broadcasts. And it requires the sender to be default dialer or has carrier privileges (see `hasCarrierPrivileges()`).

These "secret codes" are used to activate developer menus by dialing certain codes. And they are of the form `*#*#<code>#*#*`. The intent will have the data URI: `android_secret_code://<code>`. It is possible that a manifest receiver would be woken up even if it is not currently running.

It is supposed to replace `Telephony.Sms.Intents.SECRET_CODE_ACTION` in the next Android version. Before that both of these two actions will be broadcast.

Constant Value: "android.telephony.action.SECRET_CODE"

ACTION_SUBSCRIPTION_CARRIER_IDENTITY_CHANGED

```
public static final String ACTION_SUBSCRIPTION_CARRIER_IDENTITY_CHANGED
```

Broadcast Action: The subscription carrier identity has changed. This intent could be sent on the following events:

- Subscription absent. Carrier identity could change from a valid id to `TelephonyManager.UNKNOWN_CARRIER_ID`.
- Subscription loaded. Carrier identity could change from `TelephonyManager.UNKNOWN_CARRIER_ID` to a valid id.
- The subscription carrier is recognized after a remote update.

The intent will have the following extra values:

- `EXTRA_CARRIER_ID` The up-to-date carrier id of the current subscription id.
- `EXTRA_CARRIER_NAME` The up-to-date carrier name of the current subscription.
- `EXTRA_SUBSCRIPTION_ID` The subscription id associated with the changed carrier identity.

This is a protected intent that can only be sent by the system.

Constant Value: "android.telephony.action.SUBSCRIPTION_CARRIER_IDENTITY_CHANGED"

ALLOWED_NETWORK_TYPES_REASON_CARRIER

```
public static final int ALLOWED_NETWORK_TYPES_REASON_CARRIER
```

To indicate allowed network type change is requested by carrier. Carrier configuration won't affect the settings configured through other reasons and will result in allowing network types that are in both configurations (i.e intersection of both sets).

Constant Value: 2 (0x00000002)

ALLOWED_NETWORK_TYPES_REASON_USER

```
public static final int ALLOWED_NETWORK_TYPES_REASON_USER
```

To indicate allowed network type change is requested by user.

Constant Value: 0 (0x00000000)

APPTYPE_CSIM

```
public static final int APPTYPE_CSIM
```

UICC application type is CSIM

Constant Value: 4 (0x00000004)

APPTYPE_ISIM

```
public static final int APPTYPE_ISIM
```

UICC application type is ISIM

Constant Value: 5 (0x00000005)

APPTYPE_RUIM

```
public static final int APPTYPE_RUIM
```

UICC application type is RUIM

Constant Value: 3 (0x00000003)

APPTYPE_SIM

```
public static final int APPTYPE_SIM
```

UICC application type is SIM

Constant Value: 1 (0x00000001)

APPTYPE_UNKNOWN

```
public static final int APPTYPE_UNKNOWN
```

UICC application type is unknown or not specified

Constant Value: 0 (0x00000000)

APPTYPE_USIM

```
public static final int APPTYPE_USIM
```

UICC application type is USIM

Constant Value: 2 (0x00000002)

AUTHTYPE_EAP_AKA

```
public static final int AUTHTYPE_EAP_AKA
```

Authentication type for UICC challenge is EAP AKA. See RFC 4187 for details.

Constant Value: 129 (0x00000081)

AUTHTYPE_EAP_SIM

```
public static final int AUTHTYPE_EAP_SIM
```

Authentication type for UICC challenge is EAP SIM. See RFC 4186 for details.

Constant Value: 128 (0x00000080)

AUTHTYPE_GBA_BOOTSTRAP

```
public static final int AUTHTYPE_GBA_BOOTSTRAP
```

Authentication type for GBA Bootstrap Challenge. Pass this authentication type into the `getIccAuthentication(int, int, String)` API to perform a GBA Bootstrap challenge (BSF), with `data` (generated according to the procedure defined in 3GPP 33.220 Section 5.3.2 step.4) in base64 encoding. This method will return the Bootstrapping response in base64 encoding when ICC authentication is completed. Ref 3GPP 33.220 Section 5.3.2.

Constant Value: 132 (0x00000084)

AUTHTYPE_GBA_NAF_KEY_EXTERNAL

```
public static final int AUTHTYPE_GBA_NAF_KEY_EXTERNAL
```

Authentication type for GBA Network Application Functions (NAF) key External Challenge. Pass this authentication type into the `getIccAuthentication(int, int, String)` API to perform a GBA Network Applications Functions (NAF) key External challenge using the NAF_ID parameter as the `data` in base64 encoding. This method will return the Ks_Ext_Naf key in base64 encoding when ICC authentication is completed. Ref 3GPP 33.220 Section 5.3.2.

Constant Value: 133 (0x00000085)

CALL_COMPOSER_STATUS_BUSINESS_ONLY

```
public static final int CALL_COMPOSER_STATUS_BUSINESS_ONLY
```

Call composer status **Business Only** from user setting.

Constant Value: 2 (0x00000002)

CALL_COMPOSER_STATUS_OFF

```
public static final int CALL_COMPOSER_STATUS_OFF
```

Call composer status **OFF** from user setting.

Constant Value: 0 (0x00000000)

CALL_COMPOSER_STATUS_ON

```
public static final int CALL_COMPOSER_STATUS_ON
```

Call composer status **ON** from user setting.

Constant Value: 1 (0x00000001)

CALL_STATE_IDLE

```
public static final int CALL_STATE_IDLE
```

Device call state: No activity.

Constant Value: 0 (0x00000000)

CALL_STATE_OFFHOOK

```
public static final int CALL_STATE_OFFHOOK
```

Device call state: Off-hook. At least one call exists that is dialing, active, or on hold, and no calls are ringing or waiting.

Constant Value: 2 (0x00000002)

CALL_STATE_RINGING

```
public static final int CALL_STATE_RINGING
```

Device call state: Ringing. A new call arrived and is ringing or waiting. In the latter case, another call is already active.

Constant Value: 1 (0x00000001)

CARRIER_RESTRICTION_STATUS_NOT_RESTRICTED

```
public static final int CARRIER_RESTRICTION_STATUS_NOT_RESTRICTED
```

The device is not restricted to a carrier

Constant Value: 1 (0x00000001)

CARRIER_RESTRICTION_STATUS_RESTRICTED

```
public static final int CARRIER_RESTRICTION_STATUS_RESTRICTED
```

The device is restricted to a carrier.

Constant Value: 2 (0x00000002)

CARRIER_RESTRICTION_STATUS_RESTRICTED_TO_CALLER

```
public static final int CARRIER_RESTRICTION_STATUS_RESTRICTED_TO_CALLER
```

The device is restricted to the carrier of the calling application.

Constant Value: 3 (0x00000003)

CARRIER_RESTRICTION_STATUS_UNKNOWN

```
public static final int CARRIER_RESTRICTION_STATUS_UNKNOWN
```

Carrier restriction status value is unknown, in case modem did not provide any information about carrier restriction status.

Constant Value: 0 (0x00000000)

CDMA_ROAMING_MODE_AFFILIATED

```
public static final int CDMA_ROAMING_MODE_AFFILIATED
```

This constant was deprecated in API level 36.
Legacy CDMA is unsupported.

Value for `CarrierConfigManager.KEY_CDMA_ROAMING_MODE_INT` which permits roaming on affiliated networks.

Constant Value: 1 (0x00000001)

CDMA_ROAMING_MODE_ANY

```
public static final int CDMA_ROAMING_MODE_ANY
```

This constant was deprecated in API level 36.
Legacy CDMA is unsupported.

Value for `CarrierConfigManager.KEY_CDMA_ROAMING_MODE_INT` which permits roaming on any network.

Constant Value: 2 (0x00000002)

CDMA_ROAMING_MODE_HOME

```
public static final int CDMA_ROAMING_MODE_HOME
```

This constant was deprecated in API level 36.
Legacy CDMA is unsupported.

Value for `CarrierConfigManager.KEY_CDMA_ROAMING_MODE_INT` which only permits connections on home networks.

Constant Value: 0 (0x00000000)

CDMA_ROAMING_MODE_RADIO_DEFAULT

```
public static final int CDMA_ROAMING_MODE_RADIO_DEFAULT
```

This constant was deprecated in API level 36.
Legacy CDMA is unsupported.

Value for `CarrierConfigManager.KEY_CDMA_ROAMING_MODE_INT` which leaves the roaming mode set to the radio default or to the user's preference if they've indicated one.

Constant Value: -1 (0xffffffff)

DATA_ACTIVITY_DORMANT

```
public static final int DATA_ACTIVITY_DORMANT
```

Data connection is active, but physical link is down

Constant Value: 4 (0x00000004)

DATA_ACTIVITY_IN

```
public static final int DATA_ACTIVITY_IN
```

Data connection activity: Currently receiving IP PPP traffic.

Constant Value: 1 (0x00000001)

DATA_ACTIVITY_INOUT

```
public static final int DATA_ACTIVITY_INOUT
```

Data connection activity: Currently both sending and receiving IP PPP traffic.

Constant Value: 3 (0x00000003)

DATA_ACTIVITY_NONE

```
public static final int DATA_ACTIVITY_NONE
```

Data connection activity: No traffic.

Constant Value: 0 (0x00000000)

DATA_ACTIVITY_OUT

```
public static final int DATA_ACTIVITY_OUT
```

Data connection activity: Currently sending IP PPP traffic.

Constant Value: 2 (0x00000002)

DATA_CONNECTED

```
public static final int DATA_CONNECTED
```

Data connection state: Connected. IP traffic should be available.

Constant Value: 2 (0x00000002)

DATA_CONNECTING

```
public static final int DATA_CONNECTING
```

Data connection state: Currently setting up a data connection.

Constant Value: 1 (0x00000001)

DATA_DISCONNECTED

```
public static final int DATA_DISCONNECTED
```

Data connection state: Disconnected. IP traffic not available.

Constant Value: 0 (0x00000000)

DATA_DISCONNECTING

```
public static final int DATA_DISCONNECTING
```

Data connection state: Disconnecting. IP traffic may be available but will cease working imminently.

Constant Value: 4 (0x00000004)

DATA_ENABLED_REASON_CARRIER

```
public static final int DATA_ENABLED_REASON_CARRIER
```

To indicate enable or disable carrier data by the system based on carrier signalling or carrier privileged apps. Carrier data on/off won't affect user settings but will bypass the settings and turns off data internally if set to `false`.

Constant Value: 2 (0x00000002)

DATA_ENABLED_REASON_OVERRIDE

```
public static final int DATA_ENABLED_REASON_OVERRIDE
```

To indicate data was enabled or disabled due to mobile data policy overrides. Note that this is not a valid reason for `setDataEnabledForReason(int, boolean)` and is only used to indicate that data enabled was changed due to an override.

Constant Value: 4 (0x00000004)

DATA_ENABLED_REASON_POLICY

```
public static final int DATA_ENABLED_REASON_POLICY
```

To indicate that data control due to policy. Usually used when data limit is passed. Policy data on/off won't affect user settings but will bypass the settings and turns off data internally if set to `false`.

Constant Value: 1 (0x00000001)

DATA_ENABLED_REASON_THERMAL

```
public static final int DATA_ENABLED_REASON_THERMAL
```

To indicate enable or disable data by thermal service. Thermal data on/off won't affect user settings but will bypass the settings and turns off data internally if set to `false`.

Constant Value: 3 (0x00000003)

DATA_ENABLED_REASON_UNKNOWN

```
public static final int DATA_ENABLED_REASON_UNKNOWN
```

To indicate that data was enabled or disabled due to an unknown reason. Note that this is not a valid reason for `setDataEnabledForReason(int, boolean)` and is only used to indicate that data enabled was changed.

Constant Value: -1 (0xffffffff)

DATA_ENABLED_REASON_USER

```
public static final int DATA_ENABLED_REASON_USER
```

To indicate that user enabled or disabled data.

Constant Value: 0 (0x00000000)

DATA_HANDOVER_IN_PROGRESS

```
public static final int DATA_HANDOVER_IN_PROGRESS
```

Data connection state: Handover in progress. The connection is being transited from cellular network to IWLAN, or from IWLAN to cellular network.

Constant Value: 5 (0x00000005)

DATA_SUSPENDED

```
public static final int DATA_SUSPENDED
```

Data connection state: Suspended. The connection is up, but IP traffic is temporarily unavailable. For example, in a 2G network, data activity may be suspended when a voice call arrives.

Constant Value: 3 (0x00000003)

DATA_UNKNOWN

```
public static final int DATA_UNKNOWN
```

Data connection state: Unknown. Used before we know the state.

Constant Value: -1 (0xffffffff)

DEFAULT_PORT_INDEX

```
public static final int DEFAULT_PORT_INDEX
```

Default port index for a UICC. On physical SIM cards the only available port is 0. See `UiccPortInfo` for more information on ports. See `EuiccManager.isSimPortAvailable(int)` for information on how portIndex is used on eUICCs.

Constant Value: 0 (0x00000000)

ERI_FLASH

```
public static final int ERI_FLASH
```

This constant was deprecated in API level 36.
Legacy CDMA is unsupported.

ERI (Enhanced Roaming Indicator) is FLASH i.e value 2 defined by 3GPP2 C.R1001-H v1.0 Table 8.1-1.

Constant Value: 2 (0x00000002)

ERI_OFF

```
public static final int ERI_OFF
```

This constant was deprecated in API level 36.
Legacy CDMA is unsupported.

ERI (Enhanced Roaming Indicator) is OFF i.e value 1 defined by 3GPP2 C.R1001-H v1.0 Table 8.1-1.

Constant Value: 1 (0x00000001)

ERI_ON

```
public static final int ERI_ON
```

This constant was deprecated in API level 36.
Legacy CDMA is unsupported.

ERI (Enhanced Roaming Indicator) is ON i.e value 0 defined by 3GPP2 C.R1001-H v1.0 Table 8.1-1.

Constant Value: 0 (0x00000000)

```
public static final String EXTRA_ACTIVE_SIM_SUPPORTED_COUNT
```

The number of active SIM supported by current multi-SIM config. It's not related to how many SIM/subscriptions are currently active. Same value will be returned by `getActiveModemCount()` . For single SIM mode, it's 1. For DSDS or DSDA mode, it's 2. For triple-SIM mode, it's 3. Extra of `ACTION_MULTI_SIM_CONFIG_CHANGED` . type: integer

Constant Value: "android.telephony.extra.ACTIVE_SIM_SUPPORTED_COUNT"

```
public static final String EXTRA_APN_PROTOCOL
```

An integer extra containing the protocol of the apn connection. Sent with the `ACTION_CARRIER_SIGNAL_PCO_VALUE` broadcast. See the `PROTOCOL_*` constants in `ApnSetting` for a list of possible values.

Constant Value: "android.telephony.extra.APN_PROTOCOL"

```
public static final String EXTRA_CALL_VOICEMAIL_INTENT
```

The intent to call voicemail.

Constant Value: "android.telephony.extra.CALL_VOICEMAIL_INTENT"

```
public static final String EXTRA_DEFAULT_NETWORK_AVAILABLE
```

A boolean extra indicating the availability of the default network. Sent with the `ACTION_CARRIER_SIGNAL_DEFAULT_NETWORK_AVAILABLE` broadcast.

Constant Value: "android.telephony.extra.DEFAULT_NETWORK_AVAILABLE"

EXTRA_EMERGENCY_CALL_TO_SATELLITE_HANDOVER_TYPE

```
public static final String EXTRA_EMERGENCY_CALL_TO_SATELLITE_HANDOVER_TYPE
```

Integer extra key used with `EVENT_DISPLAY_EMERGENCY_MESSAGE` which indicates the type of handover from emergency call to satellite messaging.

Will be either android.telephony.satellite.SatelliteManager#EMERGENCY_CALL_TO_SATELLITE_HANDOVER_TYPE_SOS or android.telephony.satellite.SatelliteManager#EMERGENCY_CALL_TO_SATELLITE_HANDOVER_TYPE_T911

Set in the extras for the `EVENT_DISPLAY_EMERGENCY_MESSAGE` connection event.

Constant Value: "android.telephony.extra.EMERGENCY_CALL_TO_SATELLITE_HANDOVER_TYPE"

```
public static final String EXTRA_EMERGENCY_CALL_TO_SATELLITE_LAUNCH_INTENT
```

Extra key used with the `EVENT_DISPLAY_EMERGENCY_MESSAGE` for a `PendingIntent` which will be launched by the Dialer app.

Constant Value: "android.telephony.extra.EMERGENCY_CALL_TO_SATELLITE_LAUNCH_INTENT"

```
public static final String EXTRA_HIDE_PUBLIC_SETTINGS
```

The boolean value indicating whether the voicemail settings activity launched by `ACTION_CONFIGURE_VOICEMAIL` should hide settings accessible through public API. This is used by dialer implementations which provides their own voicemail settings UI, but still needs to expose device specific voicemail settings to the user.

Constant Value: "android.telephony.extra.HIDE_PUBLIC_SETTINGS"

```
public static final String EXTRA_IS_REFRESH
```

Boolean value representing whether the `TelephonyManager.ACTION_SHOW_VOICEMAIL_NOTIFICATION` is new or a refresh of an existing notification. Notification refresh happens after reboot or connectivity changes. The user has already been notified for the voicemail so it should not alert the user, and should not be shown again if the user has dismissed it.

Constant Value: "android.telephony.extra.IS_REFRESH"

```
public static final String EXTRA_LAST_KNOWN_NETWORK_COUNTRY
```

The extra used with an `ACTION_NETWORK_COUNTRY_CHANGED` to specify the last known the country code in ISO-3166-1 alpha-2 format. This might be an empty string when the country code was never available. The last known country code persists across reboot.

Retrieve with `android.content.Intent.getStringExtra(String)`.

Constant Value: "android.telephony.extra.LAST_KNOWN_NETWORK_COUNTRY"

```
public static final String EXTRA_LAUNCH_VOICEMAIL_SETTINGS_INTENT
```

The intent to launch voicemail settings.

Constant Value: "android.telephony.extra.LAUNCH_VOICEMAIL_SETTINGS_INTENT"

```
public static final String EXTRA_NOTIFICATION_COUNT
```

The number of voice messages associated with the notification.

Constant Value: "android.telephony.extra.NOTIFICATION_COUNT"

```
public static final String EXTRA_PCO_ID
```

An integer extra indicating the ID for the PCO data. Sent with the `ACTION_CARRIER_SIGNAL_PCO_VALUE` broadcast.

Constant Value: "android.telephony.extra.PCO_ID"

```
public static final String EXTRA_PCO_VALUE
```

A byte array extra containing PCO data read from the modem. Sent with the `ACTION_CARRIER_SIGNAL_PCO_VALUE` broadcast.

Constant Value: "android.telephony.extra.PCO_VALUE"

```
public static final String EXTRA_REDIRECTION_URL
```

String extra containing the redirection URL sent with `ACTION_CARRIER_SIGNAL_REDIRECTED`.

Constant Value: "android.telephony.extra.REDIRECTION_URL"

```
public static final String EXTRA_SUBSCRIPTION_ID
```

An int extra used with `ACTION_SUBSCRIPTION_CARRIER_IDENTITY_CHANGED` to indicate the subscription which has changed; or in general whenever a subscription ID needs specified.

Constant Value: "android.telephony.extra.SUBSCRIPTION_ID"

```
public static final String EXTRA_VOICEMAIL_NUMBER
```

The voicemail number.

Constant Value: "android.telephony.extra.VOICEMAIL_NUMBER"

INCLUDE_LOCATION_DATA_COARSE

```
public static final int INCLUDE_LOCATION_DATA_COARSE
```

Include coarse location data. Indicates whether the caller would not like to receive location related information which will be sent if the caller already possess `Manifest.permission.ACCESS_COARSE_LOCATION` and do not renounce the permissions.

Constant Value: 1 (0x00000001)

INCLUDE_LOCATION_DATA_FINE

```
public static final int INCLUDE_LOCATION_DATA_FINE
```

Include fine location data. Indicates whether the caller would not like to receive location related information which will be sent if the caller already possess `Manifest.permission.ACCESS_FINE_LOCATION` and do not renounce the permissions.

Constant Value: 2 (0x00000002)

INCLUDE_LOCATION_DATA_NONE

```
public static final int INCLUDE_LOCATION_DATA_NONE
```

Specifies to not include any location related data. Indicates whether the caller would not like to receive location related information which will be sent if the caller already possess `Manifest.permission.ACCESS_COARSE_LOCATION` and do not renounce the permissions.

Constant Value: 0 (0x00000000)

MULTISIM_ALLOWED

```
public static final int MULTISIM_ALLOWED
```

The usage of multiple SIM cards at the same time to register on the network (e.g. Dual Standby or Dual Active) is supported.

Constant Value: 0 (0x00000000)

MULTISIM_NOT_SUPPORTED_BY_CARRIER

```
public static final int MULTISIM_NOT_SUPPORTED_BY_CARRIER
```

The usage of multiple SIM cards at the same time to register on the network (e.g. Dual Standby or Dual Active) is supported by the hardware, but restricted by the carrier.

Constant Value: 2 (0x00000002)

MULTISIM_NOT_SUPPORTED_BY_HARDWARE

```
public static final int MULTISIM_NOT_SUPPORTED_BY_HARDWARE
```

The usage of multiple SIM cards at the same time to register on the network (e.g. Dual Standby or Dual Active) is not supported by the hardware.

Constant Value: 1 (0x00000001)

NETWORK_SELECTION_MODE_AUTO

```
public static final int NETWORK_SELECTION_MODE_AUTO
```

Constant Value: 1 (0x00000001)

NETWORK_SELECTION_MODE_MANUAL

```
public static final int NETWORK_SELECTION_MODE_MANUAL
```

Constant Value: 2 (0x00000002)

NETWORK_SELECTION_MODE_UNKNOWN

```
public static final int NETWORK_SELECTION_MODE_UNKNOWN
```

Constant Value: 0 (0x00000000)

NETWORK_TYPE_1xRTT

```
public static final int NETWORK_TYPE_1xRTT
```

This constant was deprecated in API level 36.
Legacy CDMA is unsupported.

Current network is 1xRTT

Constant Value: 7 (0x00000007)

NETWORK_TYPE_BITMASK_1xRTT

```
public static final long NETWORK_TYPE_BITMASK_1xRTT
```

network type bitmask indicating the support of radio tech 1xRTT.

Constant Value: 64 (0x0000000000000040)

NETWORK_TYPE_BITMASK_CDMA

```
public static final long NETWORK_TYPE_BITMASK_CDMA
```

This constant was deprecated in API level 36.
Legacy CDMA is unsupported.

network type bitmask indicating the support of radio tech CDMA(IS95A/IS95B).

Constant Value: 8 (0x0000000000000008)

NETWORK_TYPE_BITMASK_EDGE

```
public static final long NETWORK_TYPE_BITMASK_EDGE
```

network type bitmask indicating the support of radio tech EDGE.

Constant Value: 2 (0x0000000000000002)

NETWORK_TYPE_BITMASK_EHRPD

```
public static final long NETWORK_TYPE_BITMASK_EHRPD
```

This constant was deprecated in API level 36.
Legacy CDMA is unsupported.

network type bitmask indicating the support of radio tech EHRPD.

Constant Value: 8192 (0x0000000000002000)

NETWORK_TYPE_BITMASK_EVDO_0

```
public static final long NETWORK_TYPE_BITMASK_EVDO_0
```

network type bitmask indicating the support of radio tech EVDO 0.

Constant Value: 16 (0x0000000000000010)

NETWORK_TYPE_BITMASK_EVDO_A

```
public static final long NETWORK_TYPE_BITMASK_EVDO_A
```

network type bitmask indicating the support of radio tech EVDO A.

Constant Value: 32 (0x0000000000000020)

NETWORK_TYPE_BITMASK_EVDO_B

```
public static final long NETWORK_TYPE_BITMASK_EVDO_B
```

network type bitmask indicating the support of radio tech EVDO B.

Constant Value: 2048 (0x0000000000000800)

NETWORK_TYPE_BITMASK_GPRS

```
public static final long NETWORK_TYPE_BITMASK_GPRS
```

network type bitmask indicating the support of radio tech GPRS.

Constant Value: 1 (0x0000000000000001)

NETWORK_TYPE_BITMASK_GSM

```
public static final long NETWORK_TYPE_BITMASK_GSM
```

network type bitmask indicating the support of radio tech GSM.

Constant Value: 32768 (0x0000000000008000)

NETWORK_TYPE_BITMASK_HSDPA

```
public static final long NETWORK_TYPE_BITMASK_HSDPA
```

network type bitmask indicating the support of radio tech HSDPA.

Constant Value: 128 (0x0000000000000080)

NETWORK_TYPE_BITMASK_HSPA

```
public static final long NETWORK_TYPE_BITMASK_HSPA
```

network type bitmask indicating the support of radio tech HSPA.

Constant Value: 512 (0x0000000000000200)

NETWORK_TYPE_BITMASK_HSPAP

```
public static final long NETWORK_TYPE_BITMASK_HSPAP
```

network type bitmask indicating the support of radio tech HSPAP.

Constant Value: 16384 (0x0000000000004000)

NETWORK_TYPE_BITMASK_HSUPA

```
public static final long NETWORK_TYPE_BITMASK_HSUPA
```

network type bitmask indicating the support of radio tech HSUPA.

Constant Value: 256 (0x0000000000000100)

NETWORK_TYPE_BITMASK_IWLAN

```
public static final long NETWORK_TYPE_BITMASK_IWLAN
```

network type bitmask indicating the support of radio tech IWLAN.

Constant Value: 131072 (0x0000000000020000)

NETWORK_TYPE_BITMASK_LTE

```
public static final long NETWORK_TYPE_BITMASK_LTE
```

network type bitmask indicating the support of radio tech LTE.

Constant Value: 4096 (0x0000000000001000)

NETWORK_TYPE_BITMASK_LTE_CA

```
public static final long NETWORK_TYPE_BITMASK_LTE_CA
```

This constant was deprecated in API level 34.
Please use `NETWORK_TYPE_BITMASK_LTE` instead. Deprecated in Android U.

NOT USED; this bitmask is exposed accidentally. If used, will be converted to `NETWORK_TYPE_BITMASK_LTE`. network type bitmask indicating the support of radio tech LTE CA (carrier aggregation).

Constant Value: 262144 (0x0000000000040000)

NETWORK_TYPE_BITMASK_NR

```
public static final long NETWORK_TYPE_BITMASK_NR
```

network type bitmask indicating the support of radio tech NR(New Radio) 5G.

Constant Value: 524288 (0x0000000000080000)

NETWORK_TYPE_BITMASK_TD_SCDMA

```
public static final long NETWORK_TYPE_BITMASK_TD_SCDMA
```

network type bitmask indicating the support of radio tech TD_SCDMA.

Constant Value: 65536 (0x0000000000010000)

NETWORK_TYPE_BITMASK_UMTS

```
public static final long NETWORK_TYPE_BITMASK_UMTS
```

network type bitmask indicating the support of radio tech UMTS.

Constant Value: 4 (0x0000000000000004)

NETWORK_TYPE_BITMASK_UNKNOWN

```
public static final long NETWORK_TYPE_BITMASK_UNKNOWN
```

network type bitmask unknown.

Constant Value: 0 (0x0000000000000000)

NETWORK_TYPE_CDMA

```
public static final int NETWORK_TYPE_CDMA
```

This constant was deprecated in API level 36.
Legacy CDMA is unsupported.

Current network is CDMA: Either IS95A or IS95B

Constant Value: 4 (0x00000004)

NETWORK_TYPE_EDGE

```
public static final int NETWORK_TYPE_EDGE
```

Current network is EDGE

Constant Value: 2 (0x00000002)

NETWORK_TYPE_EHRPD

```
public static final int NETWORK_TYPE_EHRPD
```

This constant was deprecated in API level 36.
Legacy CDMA is unsupported.

Current network is eHRPD

Constant Value: 14 (0x0000000e)

NETWORK_TYPE_EVDO_0

```
public static final int NETWORK_TYPE_EVDO_0
```

This constant was deprecated in API level 36.
Legacy CDMA is unsupported.

Current network is EVDO revision 0

Constant Value: 5 (0x00000005)

NETWORK_TYPE_EVDO_A

```
public static final int NETWORK_TYPE_EVDO_A
```

This constant was deprecated in API level 36.
Legacy CDMA is unsupported.

Current network is EVDO revision A

Constant Value: 6 (0x00000006)

NETWORK_TYPE_EVDO_B

```
public static final int NETWORK_TYPE_EVDO_B
```

This constant was deprecated in API level 36.
Legacy CDMA is unsupported.

Current network is EVDO revision B

Constant Value: 12 (0x0000000c)

NETWORK_TYPE_GPRS

```
public static final int NETWORK_TYPE_GPRS
```

Current network is GPRS

Constant Value: 1 (0x00000001)

NETWORK_TYPE_GSM

```
public static final int NETWORK_TYPE_GSM
```

Current network is GSM

Constant Value: 16 (0x00000010)

NETWORK_TYPE_HSDPA

```
public static final int NETWORK_TYPE_HSDPA
```

Current network is HSDPA

Constant Value: 8 (0x00000008)

NETWORK_TYPE_HSPA

```
public static final int NETWORK_TYPE_HSPA
```

Current network is HSPA

Constant Value: 10 (0x0000000a)

NETWORK_TYPE_HSPAP

```
public static final int NETWORK_TYPE_HSPAP
```

Current network is HSPA+

Constant Value: 15 (0x0000000f)

NETWORK_TYPE_HSUPA

```
public static final int NETWORK_TYPE_HSUPA
```

Current network is HSUPA

Constant Value: 9 (0x00000009)

NETWORK_TYPE_IDEN

```
public static final int NETWORK_TYPE_IDEN
```

This constant was deprecated in API level 34.
Legacy network type no longer being used starting in Android U.

Current network is iDen

Constant Value: 11 (0x0000000b)

NETWORK_TYPE_IWLAN

```
public static final int NETWORK_TYPE_IWLAN
```

Current network is IWLAN

Constant Value: 18 (0x00000012)

NETWORK_TYPE_LTE

```
public static final int NETWORK_TYPE_LTE
```

Current network is LTE

Constant Value: 13 (0x0000000d)

NETWORK_TYPE_NR

```
public static final int NETWORK_TYPE_NR
```

Current network is NR (New Radio) 5G. This will only be returned for 5G SA. For 5G NSA, the network type will be `NETWORK_TYPE_LTE`.

Constant Value: 20 (0x00000014)

NETWORK_TYPE_TD_SCDMA

```
public static final int NETWORK_TYPE_TD_SCDMA
```

Current network is TD_SCDMA

Constant Value: 17 (0x00000011)

NETWORK_TYPE_UMTS

```
public static final int NETWORK_TYPE_UMTS
```

Current network is UMTS

Constant Value: 3 (0x00000003)

NETWORK_TYPE_UNKNOWN

```
public static final int NETWORK_TYPE_UNKNOWN
```

Network type is unknown

Constant Value: 0 (0x00000000)

PHONE_TYPE_CDMA

```
public static final int PHONE_TYPE_CDMA
```

This constant was deprecated in API level 36.
Legacy CDMA is unsupported.

Phone radio is CDMA.

Constant Value: 2 (0x00000002)

PHONE_TYPE_GSM

```
public static final int PHONE_TYPE_GSM
```

Phone radio is GSM.

Constant Value: 1 (0x00000001)

PHONE_TYPE_NONE

```
public static final int PHONE_TYPE_NONE
```

No phone radio.

Constant Value: 0 (0x00000000)

PHONE_TYPE_SIP

```
public static final int PHONE_TYPE_SIP
```

Phone is via SIP.

Constant Value: 3 (0x00000003)

PREMIUM_CAPABILITY_PRIORITIZE_LATENCY

```
public static final int PREMIUM_CAPABILITY_PRIORITIZE_LATENCY
```

A premium capability that boosts the network to allow for real-time interactive traffic by prioritizing low latency communication. Corresponds to `NetworkCapabilities.NET_CAPABILITY_PRIORITIZE_LATENCY`.

Constant Value: 34 (0x00000022)

PURCHASE_PREMIUM_CAPABILITY_RESULT_ALREADY_IN_PROGRESS

```
public static final int PURCHASE_PREMIUM_CAPABILITY_RESULT_ALREADY_IN_PROGRESS
```

Purchase premium capability failed because a request was already made and is in progress. This may have been requested by either the same app or another app. Subsequent attempts will return the same error until the previous request completes.

Constant Value: 4 (0x00000004)

PURCHASE_PREMIUM_CAPABILITY_RESULT_ALREADY_PURCHASED

```
public static final int PURCHASE_PREMIUM_CAPABILITY_RESULT_ALREADY_PURCHASED
```

Purchase premium capability failed because it is already purchased and available. Subsequent attempts will return the same error until the performance boost expires.

Constant Value: 3 (0x00000003)

PURCHASE_PREMIUM_CAPABILITY_RESULT_CARRIER_DISABLED

```
public static final int PURCHASE_PREMIUM_CAPABILITY_RESULT_CARRIER_DISABLED
```

Purchase premium capability failed because the carrier disabled or does not support the capability, as specified in `CarrierConfigManager.KEY_SUPPORTED_PREMIUM_CAPABILITIES_INT_ARRAY`. Subsequent attempts will return the same error until the carrier enables the feature.

Constant Value: 7 (0x00000007)

PURCHASE_PREMIUM_CAPABILITY_RESULT_FEATURE_NOT_SUPPORTED

```
public static final int PURCHASE_PREMIUM_CAPABILITY_RESULT_FEATURE_NOT_SUPPORTED
```

Purchase premium capability failed because the device does not support the feature. Subsequent attempts will return the same error.

Constant Value: 10 (0x0000000a)

PURCHASE_PREMIUM_CAPABILITY_RESULT_NETWORK_NOT_AVAILABLE

```
public static final int PURCHASE_PREMIUM_CAPABILITY_RESULT_NETWORK_NOT_AVAILABLE
```

Purchase premium capability failed because the network is not available. Subsequent attempts will return the same error until network conditions change.

Constant Value: 12 (0x0000000c)

PURCHASE_PREMIUM_CAPABILITY_RESULT_NOT_DEFAULT_DATA_SUBSCRIPTION

```
public static final int PURCHASE_PREMIUM_CAPABILITY_RESULT_NOT_DEFAULT_DATA_SUBSCRIPTION
```

Purchase premium capability failed because the request was not made on the default data subscription, indicated by `SubscriptionManager.getDefaultDataSubscriptionId()`. Subsequent attempts will return the same error until the request is made on the default data subscription.

Constant Value: 14 (0x0000000e)

PURCHASE_PREMIUM_CAPABILITY_RESULT_NOT_FOREGROUND

```
public static final int PURCHASE_PREMIUM_CAPABILITY_RESULT_NOT_FOREGROUND
```

Purchase premium capability failed because the requesting application is not in the foreground. Subsequent attempts will return the same error until the requesting application moves to the foreground.

Constant Value: 5 (0x00000005)

PURCHASE_PREMIUM_CAPABILITY_RESULT_PENDING_NETWORK_SETUP

```
public static final int PURCHASE_PREMIUM_CAPABILITY_RESULT_PENDING_NETWORK_SETUP
```

Purchase premium capability was successful and is waiting for the network to setup the slicing configuration. If the setup is complete within the time specified by `CarrierConfigManager.KEY_PREMIUM_CAPABILITY_NETWORK_SETUP_TIME_MILLIS_LONG`, subsequent requests will return `PURCHASE_PREMIUM_CAPABILITY_RESULT_ALREADY_PURCHASED` until the purchase expires. If the setup is not complete within the time specified above, applications can request the premium capability again.

Constant Value: 15 (0x0000000f)

PURCHASE_PREMIUM_CAPABILITY_RESULT_REQUEST_FAILED

```
public static final int PURCHASE_PREMIUM_CAPABILITY_RESULT_REQUEST_FAILED
```

Purchase premium capability failed because the telephony service is unavailable or there was an error in the phone process. Subsequent attempts will return the same error until request conditions are satisfied.

Constant Value: 11 (0x0000000b)

SET_OPPORTUNISTIC_SUB_INACTIVE_SUBSCRIPTION

```
public static final int SET_OPPORTUNISTIC_SUB_INACTIVE_SUBSCRIPTION
```

The subscription is not valid. It must be an active opportunistic subscription.

Constant Value: 2 (0x00000002)

SET_OPPORTUNISTIC_SUB_NO_OPPORTUNISTIC_SUB_AVAILABLE

```
public static final int SET_OPPORTUNISTIC_SUB_NO_OPPORTUNISTIC_SUB_AVAILABLE
```

The subscription is not valid. It must be an opportunistic subscription.

Constant Value: 3 (0x00000003)

SET_OPPORTUNISTIC_SUB_REMOTE_SERVICE_EXCEPTION

```
public static final int SET_OPPORTUNISTIC_SUB_REMOTE_SERVICE_EXCEPTION
```

Subscription service happened remote exception.

Constant Value: 4 (0x00000004)

SET_OPPORTUNISTIC_SUB_SUCCESS

```
public static final int SET_OPPORTUNISTIC_SUB_SUCCESS
```

No error. Operation succeeded.

Constant Value: 0 (0x00000000)

SET_OPPORTUNISTIC_SUB_VALIDATION_FAILED

```
public static final int SET_OPPORTUNISTIC_SUB_VALIDATION_FAILED
```

Validation failed when trying to switch to preferred subscription.

Constant Value: 1 (0x00000001)

SIM_STATE_ABSENT

```
public static final int SIM_STATE_ABSENT
```

SIM card state: no SIM card is available in the device

Constant Value: 1 (0x00000001)

SIM_STATE_CARD_IO_ERROR

```
public static final int SIM_STATE_CARD_IO_ERROR
```

SIM card state: SIM Card Error, present but faulty

Constant Value: 8 (0x00000008)

SIM_STATE_CARD_RESTRICTED

```
public static final int SIM_STATE_CARD_RESTRICTED
```

SIM card state: SIM Card restricted, present but not usable due to carrier restrictions.

Constant Value: 9 (0x00000009)

SIM_STATE_NETWORK_LOCKED

```
public static final int SIM_STATE_NETWORK_LOCKED
```

SIM card state: Locked: requires a network PIN to unlock

Constant Value: 4 (0x00000004)

SIM_STATE_NOT_READY

```
public static final int SIM_STATE_NOT_READY
```

SIM card state: SIM Card is NOT READY

Constant Value: 6 (0x00000006)

SIM_STATE_PERM_DISABLED

```
public static final int SIM_STATE_PERM_DISABLED
```

SIM card state: SIM Card Error, permanently disabled

Constant Value: 7 (0x00000007)

SIM_STATE_PIN_REQUIRED

```
public static final int SIM_STATE_PIN_REQUIRED
```

SIM card state: Locked: requires the user's SIM PIN to unlock

Constant Value: 2 (0x00000002)

SIM_STATE_PUK_REQUIRED

```
public static final int SIM_STATE_PUK_REQUIRED
```

SIM card state: Locked: requires the user's SIM PUK to unlock

Constant Value: 3 (0x00000003)

SIM_STATE_READY

```
public static final int SIM_STATE_READY
```

SIM card state: Ready

Constant Value: 5 (0x00000005)

SIM_STATE_UNKNOWN

```
public static final int SIM_STATE_UNKNOWN
```

SIM card state: Unknown. Signifies that the SIM is in transition between states. For example, when the user inputs the SIM pin under PIN_REQUIRED state, a query for sim status returns this state before turning to SIM_STATE_READY. These are the ordinal value of IccCardConstants.State.

Constant Value: 0 (0x00000000)

TTY_MODE_FULL

```
public static final int TTY_MODE_FULL
```

TTY (teletypewriter) mode is on. The speaker is off and the microphone is muted. The user will communicate with the remote party by sending and receiving text messages.

Constant Value: 1 (0x00000001)

TTY_MODE_HCO

```
public static final int TTY_MODE_HCO
```

TTY (teletypewriter) mode is in hearing carryover mode (HCO). The microphone is muted but the speaker is on. The user will communicate with the remote party by sending text messages and hearing an audible reply.

Constant Value: 2 (0x00000002)

TTY_MODE_OFF

```
public static final int TTY_MODE_OFF
```

TTY (teletypewriter) mode is off.

Constant Value: 0 (0x00000000)

TTY_MODE_VCO

```
public static final int TTY_MODE_VCO
```

TTY (teletypewriter) mode is in voice carryover mode (VCO). The speaker is off but the microphone is still on. User will communicate with the remote party by speaking and receiving text message replies.

Constant Value: 3 (0x00000003)

UNKNOWN_CARRIER_ID

```
public static final int UNKNOWN_CARRIER_ID
```

An unknown carrier id. It could either be subscription unavailable or the subscription carrier cannot be recognized. Unrecognized carriers here means `MCC+MNC` cannot be identified.

Constant Value: -1 (0xffffffff)

UNSUPPORTED_CARD_ID

```
public static final int UNSUPPORTED_CARD_ID
```

A UICC card identifier used if the device does not support the operation. For example, `getCardIdForDefaultEuicc()` returns this value if the device has no eUICC, or the eUICC cannot be read.

Constant Value: -1 (0xffffffff)

UPDATE_AVAILABLE_NETWORKS_ABORTED

```
public static final int UPDATE_AVAILABLE_NETWORKS_ABORTED
```

The request is aborted.

Constant Value: 2 (0x00000002)

UPDATE_AVAILABLE_NETWORKS_DISABLE_MODEM_FAIL

```
public static final int UPDATE_AVAILABLE_NETWORKS_DISABLE_MODEM_FAIL
```

Disable modem fail.

Constant Value: 5 (0x00000005)

UPDATE_AVAILABLE_NETWORKS_ENABLE_MODEM_FAIL

```
public static final int UPDATE_AVAILABLE_NETWORKS_ENABLE_MODEM_FAIL
```

Enable modem fail.

Constant Value: 6 (0x00000006)

UPDATE_AVAILABLE_NETWORKS_INVALID_ARGUMENTS

```
public static final int UPDATE_AVAILABLE_NETWORKS_INVALID_ARGUMENTS
```

The parameter passed in is invalid.

Constant Value: 3 (0x00000003)

UPDATE_AVAILABLE_NETWORKS_MULTIPLE_NETWORKS_NOT_SUPPORTED

```
public static final int UPDATE_AVAILABLE_NETWORKS_MULTIPLE_NETWORKS_NOT_SUPPORTED
```

Carrier app does not support multiple available networks.

Constant Value: 7 (0x00000007)

UPDATE_AVAILABLE_NETWORKS_NO_CARRIER_PRIVILEGE

```
public static final int UPDATE_AVAILABLE_NETWORKS_NO_CARRIER_PRIVILEGE
```

No carrier privilege.

Constant Value: 4 (0x00000004)

UPDATE_AVAILABLE_NETWORKS_NO_OPPORTUNISTIC_SUB_AVAILABLE

```
public static final int UPDATE_AVAILABLE_NETWORKS_NO_OPPORTUNISTIC_SUB_AVAILABLE
```

The subscription is not valid. It must be an opportunistic subscription.

Constant Value: 8 (0x00000008)

UPDATE_AVAILABLE_NETWORKS_REMOTE_SERVICE_EXCEPTION

```
public static final int UPDATE_AVAILABLE_NETWORKS_REMOTE_SERVICE_EXCEPTION
```

There is no OpportunisticNetworkService.

Constant Value: 9 (0x00000009)

UPDATE_AVAILABLE_NETWORKS_SERVICE_IS_DISABLED

```
public static final int UPDATE_AVAILABLE_NETWORKS_SERVICE_IS_DISABLED
```

OpportunisticNetworkService is disabled.

Constant Value: 10 (0x0000000a)

UPDATE_AVAILABLE_NETWORKS_SUCCESS

```
public static final int UPDATE_AVAILABLE_NETWORKS_SUCCESS
```

No error. Operation succeeded.

Constant Value: 0 (0x00000000)

UPDATE_AVAILABLE_NETWORKS_UNKNOWN_FAILURE

```
public static final int UPDATE_AVAILABLE_NETWORKS_UNKNOWN_FAILURE
```

There is a unknown failure happened.

Constant Value: 1 (0x00000001)

VVM_TYPE_CVVM

```
public static final String VVM_TYPE_CVVM
```

A flavor of OMTP protocol with a different mobile originated (MO) format

Constant Value: "vvm_type_cvvm"

VVM_TYPE_OMTP

```
public static final String VVM_TYPE_OMTP
```

The OMTP protocol.

Constant Value: "vvm_type_omtp"

Fields

Public methods

createForPhoneAccountHandle

```
public TelephonyManager createForPhoneAccountHandle (PhoneAccountHandle phoneAccountHandle)
```

Create a new TelephonyManager object pinned to the subscription ID associated with the given phone account.

Parameters	
`phoneAccountHandle`	`PhoneAccountHandle`
Returns	
`TelephonyManager`	a TelephonyManager that uses the given phone account for all calls, or `null` if the phone account does not correspond to a valid subscription ID.

createForSubscriptionId

```
public TelephonyManager createForSubscriptionId (int subId)
```

Create a new TelephonyManager object pinned to the given subscription ID.

Returns	
`TelephonyManager`	a TelephonyManager that uses the given subId for all calls.

getActiveModemCount

```
public int getActiveModemCount ()
```

Returns the number of logical modems currently configured to be activated.

- Returns 0 if none of voice, sms, data is supported.

- Returns 1 for Single standby mode (Single SIM functionality).
- Returns 2 for Dual standby mode (Dual SIM functionality).
- Returns 3 for Tri standby mode (Tri SIM functionality).

getAllCellInfo

```
public List<CellInfo> getAllCellInfo ()
```

Requests all available cell information from all radios on the device including the camped/registered, serving, and neighboring cells.

The response can include one or more `CellInfoGsm`, `CellInfoCdma`, `CellInfoTdscdma`, `CellInfoLte`, and `CellInfoWcdma` objects, in any combination. It is typical to see instances of one or more of any these in the list. In addition, zero or more of the returned objects may be considered registered; that is, their `CellInfo.isRegistered()` methods may return true, indicating that the cell is being used or would be used for signaling communication if necessary.

Beginning with `Android Q`, if this API results in a change of the cached CellInfo, that change will be reported via `onCellInfoChanged()`.

Apps targeting `Android Q` or higher will no longer trigger a refresh of the cached CellInfo by invoking this API. Instead, those apps will receive the latest cached results, which may not be current. Apps targeting `Android Q` or higher that wish to request updated CellInfo should call `requestCellInfoUpdate()`; however, in all cases, updates will be rate-limited and are not guaranteed. To determine the recency of CellInfo data, callers should check `CellInfo#getTimeStamp()`.

This method returns valid data for devices with `FEATURE_TELEPHONY`. In cases where only partial information is available for a particular CellInfo entry, unavailable fields will be reported as `CellInfo.UNAVAILABLE`. All reported cells will include at least a valid set of technology-specific identification info and a power level measurement.

This method is preferred over using `getCellLocation()`.
Requires `Manifest.permission.ACCESS_FINE_LOCATION`
Requires the `PackageManager#FEATURE_TELEPHONY_RADIO_ACCESS` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

getAllowedNetworkTypesForReason

```
public long getAllowedNetworkTypesForReason (int reason)
```

Get the allowed network types for certain reason. `getAllowedNetworkTypesForReason(int)` returns allowed network type for a specific reason.

Requires permission: android.Manifest.READ_PRIVILEGED_PHONE_STATE or that the calling app has carrier privileges (see `hasCarrierPrivileges()`).
Requires android.Manifest.permission.READ_PRIVILEGED_PHONE_STATE.
Requires the TelephonyManager#CAPABILITY_USES_ALLOWED_NETWORK_TYPES_BITMASK feature which can be detected using `TelephonyManager.isRadioInterfaceCapabilitySupported(String)`.

getCardIdForDefaultEuicc

```
public int getCardIdForDefaultEuicc ()
```

Get the card ID of the default eUICC card. If the eUICCs have not yet been loaded, returns `UNINITIALIZED_CARD_ID`. If there is no eUICC or the device does not support card IDs for eUICCs, returns `UNSUPPORTED_CARD_ID`.

The card ID is a unique identifier associated with a UICC or eUICC card. Card IDs are unique to a device, and always refer to the same UICC or eUICC card unless the device goes through a factory reset.
Requires the `PackageManager#FEATURE_TELEPHONY_EUICC` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Returns	
`int`	card ID of the default eUICC card, if loaded.

getCarrierIdFromSimMccMnc

```
public int getCarrierIdFromSimMccMnc ()
```

Returns carrier id based on sim MCCMNC (returned by `getSimOperator()`) only. This is used for fallback when configurations/logic for exact carrier id `getSimCarrierId()` are not found. Android carrier id table here can be updated out-of-band, its possible a MVNO (Mobile Virtual Network Operator) carrier was not fully recognized and assigned to its MNO (Mobile Network Operator) carrier id by default. After carrier id table update, a new carrier id was assigned. If apps don't take the update with the new id, it might be helpful to always fallback by using carrier id based on MCCMNC if there is no match.
Requires the `PackageManager#FEATURE_TELEPHONY_SUBSCRIPTION` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Returns	
`int`	matching carrier id from sim MCCMNC. Return `UNKNOWN_CARRIER_ID` if the subscription is unavailable or the carrier cannot be identified.

getDeviceId

```
public String getDeviceId ()
```

This method was deprecated in API level 26.
Use `getImei()` which returns IMEI for GSM or `getMeid()` which returns MEID for CDMA.

Returns the unique device ID, for example, the IMEI for GSM and the MEID or ESN for CDMA phones. Return null if device ID is not available.

Starting with API level 29, persistent device identifiers are guarded behind additional restrictions, and apps are recommended to use resettable identifiers (see Best practices for unique identifiers). This method can be invoked if one of the following requirements is met:

- If the calling app has been granted the READ_PRIVILEGED_PHONE_STATE permission; this is a privileged permission that can only be granted to apps preloaded on the device.
- If the calling app is the device owner of a fully-managed device, a profile owner of an organization-owned device, or their delegates (see `DevicePolicyManager.getEnrollmentSpecificId()`).
- If the calling app has carrier privileges (see `hasCarrierPrivileges()`) on any active subscription.
- If the calling app is the default SMS role holder (see `RoleManager.isRoleHeld(String)`).

If the calling app does not meet one of these requirements then this method will behave as follows:

- If the calling app's target SDK is API level 28 or lower and the app has the READ_PHONE_STATE permission then null is returned.
- If the calling app's target SDK is API level 28 or lower and the app does not have the READ_PHONE_STATE permission, or if the calling app is targeting API level 29 or higher, then a SecurityException is thrown.

.
Requires android.Manifest.permission.READ_PRIVILEGED_PHONE_STATE

getDeviceId

```
public String getDeviceId (int slotIndex)
```

This method was deprecated in API level 26.
Use `getImei()` which returns IMEI for GSM or `getMeid()` which returns MEID for CDMA.

Returns the unique device ID of a subscription, for example, the IMEI for GSM and the MEID for CDMA phones. Return null if device ID is not available.

Starting with API level 29, persistent device identifiers are guarded behind additional restrictions, and apps are recommended to use resettable identifiers (see Best practices for unique identifiers). This method can be invoked if one of the following requirements is met:

- If the calling app has been granted the READ_PRIVILEGED_PHONE_STATE permission; this is a privileged permission that can only be granted to apps preloaded on the device.
- If the calling app is the device owner of a fully-managed device, a profile owner of an organization-owned device, or their delegates (see `DevicePolicyManager.getEnrollmentSpecificId()`).
- If the calling app has carrier privileges (see `hasCarrierPrivileges()`) on any active subscription.
- If the calling app is the default SMS role holder (see `RoleManager.isRoleHeld(String)`).

If the calling app does not meet one of these requirements then this method will behave as follows:

- If the calling app's target SDK is API level 28 or lower and the app has the READ_PHONE_STATE permission then null is returned.
- If the calling app's target SDK is API level 28 or lower and the app does not have the READ_PHONE_STATE permission, or if the calling app is targeting API level 29 or higher, then a SecurityException is thrown.

.
Requires android.Manifest.permission.READ_PRIVILEGED_PHONE_STATE

Parameters	
`slotIndex`	`int` : of which deviceID is returned

getImei

```
public String getImei (int slotIndex)
```

Returns the IMEI (International Mobile Equipment Identity). Return null if IMEI is not available.

Starting with API level 29, persistent device identifiers are guarded behind additional restrictions, and apps are recommended to use resettable identifiers (see Best practices for unique identifiers). This method can be invoked if one of the following requirements is met:

- If the calling app has been granted the READ_PRIVILEGED_PHONE_STATE permission; this is a privileged permission that can only be granted to apps preloaded on the device.
- If the calling app is the device owner of a fully-managed device, a profile owner of an organization-owned device, or their delegates (see `DevicePolicyManager.getEnrollmentSpecificId()`).
- If the calling app has carrier privileges (see `hasCarrierPrivileges()`) on any active subscription.
- If the calling app is the default SMS role holder (see `RoleManager.isRoleHeld(String)`).
- If the calling app has been granted the `Manifest.permission.USE_ICC_AUTH_WITH_DEVICE_IDENTIFIER` permission.

If the calling app does not meet one of these requirements then this method will behave as follows:

- If the calling app's target SDK is API level 28 or lower and the app has the READ_PHONE_STATE permission then null is returned.
- If the calling app's target SDK is API level 28 or lower and the app does not have the READ_PHONE_STATE permission, or if the calling app is targeting API level 29 or higher, then a SecurityException is thrown.

.
Requires android.Manifest.permission.READ_PRIVILEGED_PHONE_STATE

Parameters	
`slotIndex`	`int` : of which IMEI is returned

getImei

```
public String getImei ()
```

Returns the IMEI (International Mobile Equipment Identity). Return null if IMEI is not available. See `getImei(int)` for details on the required permissions and behavior when the caller does not hold sufficient permissions.
Requires android.Manifest.permission.READ_PRIVILEGED_PHONE_STATE

getManufacturerCode

```
public String getManufacturerCode (int slotIndex)
```

This method was deprecated in API level 36.
Legacy CDMA is unsupported.

Returns the Manufacturer Code from the MEID. Return null if Manufacturer Code is not available.
Requires the `PackageManager#FEATURE_TELEPHONY_CDMA` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Parameters	
`slotIndex`	`int` : of which Type Allocation Code is returned

getMeid

```
public String getMeid ()
```

This method was deprecated in API level 36.
Legacy CDMA is unsupported.

Returns the MEID (Mobile Equipment Identifier). Return null if MEID is not available.

Starting with API level 29, persistent device identifiers are guarded behind additional restrictions, and apps are recommended to use resettable identifiers (see Best practices for unique identifiers). This method can be invoked if one of the following requirements is met:

- If the calling app has been granted the READ_PRIVILEGED_PHONE_STATE permission; this is a privileged permission that can only be granted to apps preloaded on the device.
- If the calling app is the device owner of a fully-managed device, a profile owner of an organization-owned device, or their delegates (see `DevicePolicyManager.getEnrollmentSpecificId()`).
- If the calling app has carrier privileges (see `hasCarrierPrivileges()`) on any active subscription.
- If the calling app is the default SMS role holder (see `RoleManager.isRoleHeld(String)`).

If the calling app does not meet one of these requirements then this method will behave as follows:

- If the device is running Android 25Q2 or later, then null is returned.
- If the calling app's target SDK is API level 28 or lower and the app has the READ_PHONE_STATE permission then null is returned.
- If the calling app's target SDK is API level 28 or lower and the app does not have the READ_PHONE_STATE permission, or if the calling app is targeting API level 29 or higher, then a SecurityException is thrown.

.
Requires android.Manifest.permission.READ_PRIVILEGED_PHONE_STATE.
Requires the `PackageManager#FEATURE_TELEPHONY_CDMA` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

getMeid

```
public String getMeid (int slotIndex)
```

This method was deprecated in API level 36.
Legacy CDMA is unsupported.

Returns the MEID (Mobile Equipment Identifier). Return null if MEID is not available.

Starting with API level 29, persistent device identifiers are guarded behind additional restrictions, and apps are recommended to use resettable identifiers (see Best practices for unique identifiers). This method can be invoked if one of the following requirements is met:

- If the calling app has been granted the READ_PRIVILEGED_PHONE_STATE permission; this is a privileged permission that can only be granted to apps preloaded on the device.
- If the calling app is the device owner of a fully-managed device, a profile owner of an organization-owned device, or their delegates (see `DevicePolicyManager.getEnrollmentSpecificId()`).
- If the calling app has carrier privileges (see `hasCarrierPrivileges()`) on any active subscription.
- If the calling app is the default SMS role holder (see `RoleManager.isRoleHeld(String)`).

If the calling app does not meet one of these requirements then this method will behave as follows:

- If the calling app's target SDK is API level 28 or lower and the app has the READ_PHONE_STATE permission then null is returned.
- If the calling app's target SDK is API level 28 or lower and the app does not have the READ_PHONE_STATE permission, or if the calling app is targeting API level 29 or higher, then a SecurityException is thrown.

.
Requires android.Manifest.permission.READ_PRIVILEGED_PHONE_STATE.
Requires the `PackageManager#FEATURE_TELEPHONY_CDMA` feature which can be detected using `PackageManager.hasSystemFeature(String)` .

Parameters	
`slotIndex`	`int` : of which MEID is returned

getNai

```
public String getNai ()
```

Returns the Network Access Identifier (NAI). Return null if NAI is not available.

Starting with API level 29, persistent device identifiers are guarded behind additional restrictions, and apps are recommended to use resettable identifiers (see Best practices for unique identifiers). This method can be invoked if one of the following requirements is met:

- If the calling app has been granted the READ_PRIVILEGED_PHONE_STATE permission; this is a privileged permission that can only be granted to apps preloaded on the device.
- If the calling app is the device owner of a fully-managed device, a profile owner of an organization-owned device, or their delegates (see `DevicePolicyManager.getEnrollmentSpecificId()`).
- If the calling app has carrier privileges (see `hasCarrierPrivileges()`).
- If the calling app is the default SMS role holder (see `RoleManager.isRoleHeld(String)`).

If the calling app does not meet one of these requirements then this method will behave as follows:

- If the calling app's target SDK is API level 28 or lower and the app has the READ_PHONE_STATE permission then null is returned.
- If the calling app's target SDK is API level 28 or lower and the app does not have the READ_PHONE_STATE permission, or if the calling app is targeting API level 29 or higher, then a SecurityException is thrown.

.
Requires android.Manifest.permission.READ_PRIVILEGED_PHONE_STATE.
Requires the `PackageManager#FEATURE_TELEPHONY_SUBSCRIPTION` feature which can be detected using `PackageManager.hasSystemFeature(String)` .

getNetworkCountryIso

```
public String getNetworkCountryIso (int slotIndex)
```

Returns the ISO-3166-1 alpha-2 country code equivalent of the MCC (Mobile Country Code) of the current registered operator or the cell nearby, if available. This is same as `getNetworkCountryIso()` but allowing specifying the SIM slot index. This is used for accessing network country info from the SIM slot that does not have SIM inserted.

Requires the `PackageManager#FEATURE_TELEPHONY_RADIO_ACCESS` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Parameters	
`slotIndex`	`int`: the SIM slot index to get network country ISO.
Returns	
`String`	the lowercase 2 character ISO-3166-1 alpha-2 country code, or empty string if not available. This value cannot be `null`.

getNetworkSlicingConfiguration

```
public void getNetworkSlicingConfiguration (Executor executor,
                OutcomeReceiver<NetworkSlicingConfig, TelephonyManager.NetworkSlicingException> callback)
```

Request to get the current slicing configuration including URSP rules and NSSAIs (configured, allowed and rejected). This method can be invoked if one of the following requirements is met:

- If the calling app has been granted the READ_PRIVILEGED_PHONE_STATE permission; this is a privileged permission that can only be granted to apps preloaded on the device.
- If the calling app has carrier privileges (see `hasCarrierPrivileges()`).

This will be invalid if the device does not support android.telephony.TelephonyManager#CAPABILITY_SLICING_CONFIG_SUPPORTED.
Requires the `TelephonyManager#CAPABILITY_SLICING_CONFIG_SUPPORTED` feature which can be detected using `TelephonyManager.isRadioInterfaceCapabilitySupported(String)`.
Requires android.Manifest.permission.READ_PRIVILEGED_PHONE_STATE

Parameters	
`executor`	`Executor`: the executor on which callback will be invoked. This value cannot be `null`. Callback and listener events are dispatched through this `Executor`, providing an easy way to control which thread is used. To dispatch events through the main thread of your application, you can use `Context.getMainExecutor()`. Otherwise, provide an `Executor` that dispatches to an appropriate thread.
`callback`	`OutcomeReceiver`: a callback to receive the current slicing configuration. This value cannot be `null`.

getPhoneCount

```
public int getPhoneCount ()
```

This method was deprecated in API level 30.
Use `getActiveModemCount()` instead.

Returns the number of phones available.

- Returns 0 if none of voice, sms, data is supported.
- Returns 1 for Single standby mode (Single SIM functionality).

- Returns 2 for Dual standby mode (Dual SIM functionality).
- Returns 3 for Tri standby mode (Tri SIM functionality).

getPrimaryImei

```
public String getPrimaryImei ()
```

Returns the primary IMEI (International Mobile Equipment Identity) of the device as mentioned in GSMA TS.37. `getImei(int)` returns the IMEI that belongs to the selected slotID whereas this API `getPrimaryImei()` returns primary IMEI of the device. A single SIM device with only one IMEI will be set by default as primary IMEI. A multi-SIM device with multiple IMEIs will have one of the IMEIs set as primary as mentioned in GSMA TS37_2.2_REQ_8.

Requires one of the following permissions

- If the calling app has been granted the READ_PRIVILEGED_PHONE_STATE permission; this is a privileged permission that can only be granted to apps preloaded on the device.
- If the calling app is the device owner of a fully-managed device, a profile owner of an organization-owned device, or their delegates (see `DevicePolicyManager.getEnrollmentSpecificId()`).
- If the calling app has carrier privileges (see `hasCarrierPrivileges()`) on any active subscription.
- If the calling app is the default SMS role holder (see `RoleManager.isRoleHeld(String)`).
- If the calling app has been granted the `Manifest.permission.USE_ICC_AUTH_WITH_DEVICE_IDENTIFIER` permission.

Returns	
`String`	Primary IMEI of type string. This value cannot be `null` .
Throws	
`SecurityException`	if the caller does not have the required permission/privileges

getSimCarrierId

```
public int getSimCarrierId ()
```

Returns carrier id of the current subscription.

To recognize a carrier (including MVNO) as a first-class identity, Android assigns each carrier with a canonical integer a.k.a. carrier id. The carrier ID is an Android platform-wide identifier for a carrier. AOSP maintains carrier ID assignments in here

Apps which have carrier-specific configurations or business logic can use the carrier id as an Android platform-wide identifier for carriers.
Requires the `PackageManager#FEATURE_TELEPHONY_SUBSCRIPTION` feature which can be detected using `PackageManager.hasSystemFeature(String)` .

Returns	
`int`	Carrier id of the current subscription. Return `UNKNOWN_CARRIER_ID` if the subscription is unavailable or the carrier cannot be identified.

getSimCarrierIdName

```
public CharSequence getSimCarrierIdName ()
```

Returns carrier id name of the current subscription.

Carrier id name is a user-facing name of carrier id returned by `getSimCarrierId()`, usually the brand name of the subsidiary (e.g. T-Mobile). Each carrier could configure multiple `SPN` but should have a single carrier name. Carrier name is not a canonical identity, use `getSimCarrierId()` instead.

The returned carrier name is unlocalized.
Requires the `PackageManager#FEATURE_TELEPHONY_SUBSCRIPTION` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Returns	
`CharSequence`	Carrier name of the current subscription. Return `null` if the subscription is unavailable or the carrier cannot be identified.

getSimSerialNumber

```
public String getSimSerialNumber ()
```

Returns the serial number of the SIM, if applicable. Return null if it is unavailable.

Starting with API level 29, persistent device identifiers are guarded behind additional restrictions, and apps are recommended to use resettable identifiers (see Best practices for unique identifiers). This method can be invoked if one of the following requirements is met:

- If the calling app has been granted the READ_PRIVILEGED_PHONE_STATE permission; this is a privileged permission that can only be granted to apps preloaded on the device.
- If the calling app is the device owner of a fully-managed device, a profile owner of an organization-owned device, or their delegates (see `DevicePolicyManager.getEnrollmentSpecificId()`).
- If the calling app has carrier privileges (see `hasCarrierPrivileges()`).
- If the calling app is the default SMS role holder (see `RoleManager.isRoleHeld(String)`).

If the calling app does not meet one of these requirements then this method will behave as follows:

- If the calling app's target SDK is API level 28 or lower and the app has the READ_PHONE_STATE permission then null is returned.
- If the calling app's target SDK is API level 28 or lower and the app does not have the READ_PHONE_STATE permission, or if the calling app is targeting API level 29 or higher, then a SecurityException is thrown.

.
Requires android.Manifest.permission.READ_PRIVILEGED_PHONE_STATE.
Requires the `PackageManager#FEATURE_TELEPHONY_SUBSCRIPTION` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

getSimSpecificCarrierId

```
public int getSimSpecificCarrierId ()
```

Returns fine-grained carrier ID of the current subscription. A specific carrier ID can represent the fact that a carrier may be in effect an aggregation of other carriers (ie in an MVNO type scenario) where each of these specific carriers which are used to make up the actual carrier service may have different carrier configurations. A specific carrier ID could also be used, for example, in a scenario where a carrier requires different carrier configuration for different service offering such as a prepaid plan. the specific carrier ID would be used for configuration purposes, but apps wishing to know about the carrier itself should use the regular carrier ID returned by `getSimCarrierId()`. e.g, Tracfone SIMs could return different specific carrier ID based on IMSI from current subscription while carrier ID remains the same.

For carriers without fine-grained specific carrier ids, return `getSimCarrierId()`

Specific carrier ids are defined in the same way as carrier id here except each with a "parent" id linking to its top-level carrier id.
Requires the `PackageManager#FEATURE_TELEPHONY_SUBSCRIPTION` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Returns	
`int`	Returns fine-grained carrier id of the current subscription. Return `UNKNOWN_CARRIER_ID` if the subscription is unavailable or the carrier cannot be identified.

getSubscriberId

```
public String getSubscriberId ()
```

Returns the unique subscriber ID, for example, the IMSI for a GSM phone. Return null if it is unavailable.

Starting with API level 29, persistent device identifiers are guarded behind additional restrictions, and apps are recommended to use resettable identifiers (see Best practices for unique identifiers). This method can be invoked if one of the following requirements is met:

- If the calling app has been granted the READ_PRIVILEGED_PHONE_STATE permission; this is a privileged permission that can only be granted to apps preloaded on the device.
- If the calling app is the device owner of a fully-managed device, a profile owner of an organization-owned device, or their delegates (see `DevicePolicyManager.getEnrollmentSpecificId()`).
- If the calling app has carrier privileges (see `hasCarrierPrivileges()`).
- If the calling app is the default SMS role holder (see `RoleManager.isRoleHeld(String)`).
- If the calling app has been granted the `Manifest.permission.USE_ICC_AUTH_WITH_DEVICE_IDENTIFIER` permission.

If the calling app does not meet one of these requirements then this method will behave as follows:

- If the calling app's target SDK is API level 28 or lower and the app has the READ_PHONE_STATE permission then null is returned.
- If the calling app's target SDK is API level 28 or lower and the app does not have the READ_PHONE_STATE permission, or if the calling app is targeting API level 29 or higher, then a SecurityException is thrown.

.
Requires android.Manifest.permission.READ_PRIVILEGED_PHONE_STATE.
Requires the `PackageManager#FEATURE_TELEPHONY_SUBSCRIPTION` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

getSupportedModemCount

```
public int getSupportedModemCount ()
```

Return how many logical modem can be potentially active simultaneously, in terms of hardware capability. It might return different value from `getActiveModemCount()`. For example, for a dual-SIM capable device operating in single SIM mode (only one logical modem is turned on), `getActiveModemCount()` returns 1 while this API returns 2.

getTypeAllocationCode

```
public String getTypeAllocationCode ()
```

Returns the Type Allocation Code from the IMEI. Return null if Type Allocation Code is not available.

getTypeAllocationCode

```
public String getTypeAllocationCode (int slotIndex)
```

Returns the Type Allocation Code from the IMEI. Return null if Type Allocation Code is not available.

Parameters	
`slotIndex`	`int`: of which Type Allocation Code is returned

hasCarrierPrivileges

```
public boolean hasCarrierPrivileges ()
```

Has the calling application been granted carrier privileges by the carrier. If any of the packages in the calling UID has carrier privileges, the call will return true. This access is granted by the owner of the UICC card and does not depend on the registered carrier. Note that this API applies to both physical and embedded subscriptions and is a superset of the checks done in SubscriptionManager#canManageSubscription.
Requires the `PackageManager#FEATURE_TELEPHONY_SUBSCRIPTION` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Returns	
`boolean`	true if the app has carrier privileges.

hasIccCard

```
public boolean hasIccCard ()
```

This API is used to check if there is an ICC card present in the device. An ICC card is a smart card that contains a subscriber identity module (SIM) and is used to identify and authenticate users to a mobile network. Note: In case of embedded SIM there is an ICC card always present irrespective of whether an active SIM profile is present or not so this API would always return true.
Requires the `PackageManager#FEATURE_TELEPHONY_SUBSCRIPTION` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Returns	
`boolean`	true if a ICC card is present.

iccCloseLogicalChannel

```
public boolean iccCloseLogicalChannel (int channel)
```

Closes a previously opened logical channel to the ICC card. Input parameters equivalent to TS 27.007 AT+CCHC command. It is strongly recommended that callers of this API should firstly create new TelephonyManager instance by calling `TelephonyManager.createForSubscriptionId(int)`. Failure to do so can result in unpredictable and detrimental behavior like callers can end up talking to the wrong SIM card.

Requires Permission: `MODIFY_PHONE_STATE` or that the calling app has carrier privileges (see `hasCarrierPrivileges()`). Requires the `PackageManager#FEATURE_TELEPHONY_SUBSCRIPTION` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Parameters	
`channel`	`int`: is the channel id to be closed as returned by a successful iccOpenLogicalChannel.

Returns	
`boolean`	true if the channel was closed successfully.

iccOpenLogicalChannel

```
public IccOpenLogicalChannelResponse iccOpenLogicalChannel (String AID,
                int p2)
```

Opens a logical channel to the ICC card. This operation wraps two APDU instructions:

- MANAGE CHANNEL to open a logical channel
- SELECT the given `AID` using the given `p2`

Per Open Mobile API Specification v3.2 section 6.2.7.h, only p2 values of 0x00, 0x04, 0x08, and 0x0C are guaranteed to be supported. If the SELECT command's status word is not '9000', '62xx', or '63xx', the status word will be considered an error and the channel shall not be opened. Input parameters equivalent to TS 27.007 AT+CCHO command. It is strongly recommended that callers of this should firstly create a new TelephonyManager instance by calling `TelephonyManager.createForSubscriptionId(int)`. Failure to do so can result in unpredictable and detrimental behavior like callers can end up talking to the wrong SIM card.

Requires Permission: `MODIFY_PHONE_STATE` or that the calling app has carrier privileges (see `hasCarrierPrivileges()`). Requires the `PackageManager#FEATURE_TELEPHONY_SUBSCRIPTION` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Parameters	
`AID`	`String`: Application id. See ETSI 102.221 and 101.220.
`p2`	`int`: P2 parameter (described in ISO 7816-4).

iccTransmitApduBasicChannel

```
public String iccTransmitApduBasicChannel (int cla,
                int instruction,
                int p1,
                int p2,
                int p3,
                String data)
```

Transmit an APDU to the ICC card over the basic channel. Input parameters equivalent to TS 27.007 AT+CSIM command.

Requires Permission: `MODIFY_PHONE_STATE` or that the calling app has carrier privileges (see `hasCarrierPrivileges()`). Requires the `PackageManager#FEATURE_TELEPHONY_SUBSCRIPTION` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Parameters	
`cla`	`int` : Class of the APDU command.
`instruction`	`int` : Instruction of the APDU command.
`p1`	`int` : P1 value of the APDU command.
`p2`	`int` : P2 value of the APDU command.
`p3`	`int` : P3 value of the APDU command. If p3 is negative a 4 byte APDU is sent to the SIM.
`data`	`String` : Data to be sent with the APDU.
Returns	
`String`	The APDU response from the ICC card with the status appended at the end.

iccTransmitApduLogicalChannel

```
public String iccTransmitApduLogicalChannel (int channel,
                int cla,
                int instruction,
                int p1,
                int p2,
                int p3,
                String data)
```

Transmit an APDU to the ICC card over a logical channel. Input parameters equivalent to TS 27.007 AT+CGLA command. It is strongly recommended that callers of this API should firstly create a new TelephonyManager instance by calling `TelephonyManager.createForSubscriptionId(int)`. Failure to do so can result in unpredictable and detrimental behavior like callers can end up talking to the wrong SIM card.

Requires Permission: `MODIFY_PHONE_STATE` or that the calling app has carrier privileges (see `hasCarrierPrivileges()`). Requires the `PackageManager#FEATURE_TELEPHONY_SUBSCRIPTION` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Parameters

`channel`	`int` : is the channel id to be closed as returned by a successful iccOpenLogicalChannel.
`cla`	`int` : Class of the APDU command.
`instruction`	`int` : Instruction of the APDU command.
`p1`	`int` : P1 value of the APDU command.
`p2`	`int` : P2 value of the APDU command.
`p3`	`int` : P3 value of the APDU command. If p3 is negative a 4 byte APDU is sent to the SIM.
`data`	`String` : Data to be sent with the APDU.

Returns	
`String`	The APDU response from the ICC card with the status appended at the end.

isConcurrentVoiceAndDataSupported

```
public boolean isConcurrentVoiceAndDataSupported ()
```

Whether the device is currently on a technology (e.g. UMTS or LTE) which can support voice and data simultaneously. This can change based on location or network condition.
Requires the `PackageManager#FEATURE_TELEPHONY_DATA` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Returns	
`boolean`	`true` if simultaneous voice and data supported, and `false` otherwise.

isDataCapable

```
public boolean isDataCapable ()
```

Returns	
`boolean`	true if the current device is "data capable" over a radio on the device. "Data capable" means that this device supports packet-switched data connections over the telephony network.

isDeviceSmsCapable

```
public boolean isDeviceSmsCapable ()
```

Returns	
`boolean`	true if the current device supports SMS service. If true, this means that the device supports both sending and receiving SMS via the telephony network.

	Note: Voicemail waiting SMS, cell broadcasting SMS, and MMS are disabled when device doesn't support SMS. Starting from Android 15, SMS capability may also be overridden by carriers for a given subscription on an SMS capable device. To check if a subscription is "SMS capable", call method `SubscriptionInfo.getServiceCapabilities()` and check if `SubscriptionManager.SERVICE_CAPABILITY_SMS` is included.

isDeviceVoiceCapable

```
public boolean isDeviceVoiceCapable ()
```

Returns	
`boolean`	true if the current device is "voice capable". "Voice capable" means that this device supports circuit-switched or IMS packet switched (i.e. voice) phone calls over the telephony network, and is allowed to display the in-call UI while a cellular voice call is active. This will be false on "data only" devices which can't make voice calls and don't support any in-call UI. Note: the meaning of this flag is subtly different from the PackageManager .FEATURE_TELEPHONY system feature, which is available on any device with a telephony radio, even if the device is data-only. Starting from Android 15, voice capability may also be overridden by carrier for a given subscription on a voice capable device. To check if a subscription is "voice capable", call method `SubscriptionInfo.getServiceCapabilities()` and check if `SubscriptionManager.SERVICE_CAPABILITY_VOICE` is included.

isEmergencyNumber

```
public boolean isEmergencyNumber (String number)
```

Identifies if the supplied phone number is an emergency number that matches a known emergency number based on current locale, SIM card(s), Android database, modem, network, or defaults.

This method assumes that only dialable phone numbers are passed in; non-dialable numbers are not considered emergency numbers. A dialable phone number consists only of characters/digits identified by `PhoneNumberUtils.isDialable(char)` .

The subscriptions which the identification would be based on, are all the active subscriptions, no matter which subscription could be used to create TelephonyManager.

Parameters	
`number`	`String` : - the number to look up. This value cannot be `null` .
Returns	

`boolean`	`true` if the given number is an emergency number based on current locale, SIM card(s), Android database, modem, network or defaults; `false` otherwise.

isRadioInterfaceCapabilitySupported

```
public boolean isRadioInterfaceCapabilitySupported (String capability)
```

Whether the device supports a given capability on the radio interface. If the capability is not in the set of radio interface capabilities, false is returned.
Requires the `PackageManager#FEATURE_TELEPHONY_RADIO_ACCESS` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Returns	
`boolean`	the availability of the capability

isRttSupported

```
public boolean isRttSupported ()
```

Determines whether the device currently supports RTT (Real-time text). Based both on carrier support for the feature and device firmware support.
Requires the `PackageManager#FEATURE_TELEPHONY_IMS` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Returns	
`boolean`	`true` if the device and carrier both support RTT, `false` otherwise.

isSmsCapable

```
public boolean isSmsCapable ()
```

This method was deprecated in API level 35.
Replaced by `isDeviceSmsCapable()`. Starting from Android 15, SMS capability may also be overridden by carriers for a given subscription. For SMS capable device (when `isDeviceSmsCapable()` return `true`), caller should check for subscription-level SMS capability as well. See `isDeviceSmsCapable()` for details.

Returns	
`boolean`	true if the current device supports sms service. If true, this means that the device supports both sending and receiving sms via the telephony network. Note: Voicemail waiting sms, cell broadcasting sms, and MMS are disabled when device doesn't support sms.

isTtyModeSupported

```
public boolean isTtyModeSupported ()
```

This method was deprecated in API level 28.
Use `TelecomManager.isTtySupported()` instead Whether the phone supports TTY mode.

Returns	
`boolean`	`true` if the device supports TTY mode, and `false` otherwise.

isVoiceCapable

```
public boolean isVoiceCapable ()
```

This method was deprecated in API level 35.
Replaced by `isDeviceVoiceCapable()`. Starting from Android 15, voice capability may also be overridden by carriers for a given subscription. For voice capable device (when `isDeviceVoiceCapable()` return `true`), caller should check for subscription-level voice capability as well. See `isDeviceVoiceCapable()` for details.

Returns	
`boolean`	true if the current device is "voice capable". "Voice capable" means that this device supports circuit-switched (i.e. voice) phone calls over the telephony network, and is allowed to display the in-call UI while a cellular voice call is active. This will be false on "data only" devices which can't make voice calls and don't support any in-call UI. Note: the meaning of this flag is subtly different from the PackageManager.FEATURE_TELEPHONY system feature, which is available on any device with a telephony radio, even if the device is data-only.

listen

```
public void listen (PhoneStateListener listener,
                int events)
```

This method was deprecated in API level 31.
Use `registerTelephonyCallback(Executor,TelephonyCallback)`.

Registers a listener object to receive notification of changes in specified telephony states.

To register a listener, pass a `PhoneStateListener` and specify at least one telephony state of interest in the events argument. At registration, and when a specified telephony state changes, the telephony manager invokes the appropriate callback method on the listener object and passes the current (updated) values.

To un-register a listener, pass the listener object and set the events argument to `LISTEN_NONE` (0). If this TelephonyManager object has been created with `createForSubscriptionId(int)`, applies to the given subId. Otherwise, applies to `SubscriptionManager.getDefaultSubscriptionId()`. To listen events for multiple subIds, pass a separate listener object to each TelephonyManager object created with `createForSubscriptionId(int)`. Note: if you call this method while in the middle of a binder transaction, you **must** call `Binder.clearCallingIdentity()` before calling this method. A `SecurityException` will be thrown otherwise. This API should be used sparingly -- large numbers of listeners will cause

system instability. If a process has registered too many listeners without unregistering them, it may encounter an `IllegalStateException` when trying to register more listeners.

Parameters	
`listener`	`PhoneStateListener` : The `PhoneStateListener` object to register (or unregister)
`events`	`int` : The telephony state(s) of interest to the listener, as a bitwise-OR combination of `PhoneStateListener` LISTEN_ flags.

registerTelephonyCallback

```
public void registerTelephonyCallback (Executor executor,
                TelephonyCallback callback)
```

Registers a callback object to receive notification of changes in specified telephony states.

To register a callback, pass a `TelephonyCallback` which implements interfaces of events. For example, FakeServiceStateCallback extends `TelephonyCallback` implements `TelephonyCallback.ServiceStateListener` . At registration, and when a specified telephony state changes, the telephony manager invokes the appropriate callback method on the callback object and passes the current (updated) values.

Note: Be aware of the permission requirements stated on the `TelephonyCallback` listeners you implement. Your application must be granted these permissions in order to register a `TelephonyCallback` which requires them; a `SecurityException` will be thrown if you do not hold the required permissions for all `TelephonyCallback` listeners you implement.

If this TelephonyManager object has been created with `createForSubscriptionId(int)` , applies to the given subId. Otherwise, applies to `SubscriptionManager.getDefaultSubscriptionId()` . To register events for multiple subIds, pass a separate callback object to each TelephonyManager object created with `createForSubscriptionId(int)` . Note: if you call this method while in the middle of a binder transaction, you **must** call `Binder.clearCallingIdentity()` before calling this method. A `SecurityException` will be thrown otherwise. This API should be used sparingly -- large numbers of callbacks will cause system instability. If a process has registered too many callbacks without unregistering them, it may encounter an `IllegalStateException` when trying to register more callbacks.

Parameters	
`executor`	`Executor` : The executor of where the callback will execute. This value cannot be `null` . Callback and listener events are dispatched through this `Executor` , providing an easy way to control which thread is used. To dispatch events through the main thread of your application, you can use `Context.getMainExecutor()` . Otherwise, provide an `Executor` that dispatches to an appropriate thread.
`callback`	`TelephonyCallback` : The `TelephonyCallback` object to register. The caller should hold a reference to the callback. The framework only holds a weak reference. This value cannot be `null` .

registerTelephonyCallback

```
public void registerTelephonyCallback (int includeLocationData,
                Executor executor,
                TelephonyCallback callback)
```

Registers a callback object to receive notification of changes in specified telephony states.

To register a callback, pass a `TelephonyCallback` which implements interfaces of events. For example, FakeServiceStateCallback extends `TelephonyCallback` implements `TelephonyCallback.ServiceStateListener`. At registration, and when a specified telephony state changes, the telephony manager invokes the appropriate callback method on the callback object and passes the current (updated) values.

If this TelephonyManager object has been created with `createForSubscriptionId(int)`, applies to the given subId. Otherwise, applies to `SubscriptionManager.getDefaultSubscriptionId()`. To register events for multiple subIds, pass a separate callback object to each TelephonyManager object created with `createForSubscriptionId(int)`. Note: if you call this method while in the middle of a binder transaction, you **must** call `Binder.clearCallingIdentity()` before calling this method. A `SecurityException` will be thrown otherwise. This API should be used sparingly -- large numbers of callbacks will cause system instability. If a process has registered too many callbacks without unregistering them, it may encounter an `IllegalStateException` when trying to register more callbacks.

There's another way to renounce permissions with a custom context `AttributionSource.Builder#setRenouncedPermissions(Set<String>)` but only for system apps. To avoid confusion, calling this method supersede renouncing permissions with a custom context.

Parameters	
`includeLocationData`	`int` : Specifies if the caller would like to receive location related information. Value is one of the following: • `INCLUDE_LOCATION_DATA_NONE` • `INCLUDE_LOCATION_DATA_COARSE` • `INCLUDE_LOCATION_DATA_FINE`
`executor`	`Executor` : The executor of where the callback will execute. This value cannot be `null`. Callback and listener events are dispatched through this `Executor`, providing an easy way to control which thread is used. To dispatch events through the main thread of your application, you can use `Context.getMainExecutor()`. Otherwise, provide an `Executor` that dispatches to an appropriate thread.
`callback`	`TelephonyCallback` : The `TelephonyCallback` object to register. The caller should hold a reference to the callback. The framework only holds a weak reference. This value cannot be `null`.

requestNetworkScan

```
public NetworkScan requestNetworkScan (NetworkScanRequest request,
                Executor executor,
                TelephonyScanManager.NetworkScanCallback callback)
```

Request a network scan. This method is asynchronous, so the network scan results will be returned by callback. The returned NetworkScan will contain a callback method which can be used to stop the scan.

Requires Permission: `MODIFY_PHONE_STATE` or that the calling app has carrier privileges (see `hasCarrierPrivileges()`) and `Manifest.permission.ACCESS_FINE_LOCATION`. If the system-wide location switch is off, apps may still call this API, with the following constraints:

1. The app must hold the `android.permission.NETWORK_SCAN` permission.
2. The app must not supply any specific bands or channels to scan.
3. The app must only specify MCC/MNC pairs that are associated to a SIM in the device.
4. Returned results will have no meaningful info other than signal strength and MCC/MNC info.

.

Requires `Manifest.permission.MODIFY_PHONE_STATE` and `Manifest.permission.ACCESS_FINE_LOCATION`
Requires the `PackageManager#FEATURE_TELEPHONY_RADIO_ACCESS` feature which can be detected using `PackageManager.hasSystemFeature(String)` .

Parameters	
`request`	`NetworkScanRequest` : Contains all the RAT with bands/channels that need to be scanned.
`executor`	`Executor` : The executor through which the callback should be invoked. Since the scan request may trigger multiple callbacks and they must be invoked in the same order as they are received by the platform, the user should provide an executor which executes tasks one at a time in serial order.
`callback`	`TelephonyScanManager.NetworkScanCallback` : Returns network scan results or errors.

Returns	
`NetworkScan`	A NetworkScan obj which contains a callback which can be used to stop the scan.

requestNetworkScan

```
public NetworkScan requestNetworkScan (int includeLocationData,
                NetworkScanRequest request,
                Executor executor,
                TelephonyScanManager.NetworkScanCallback callback)
```

Request a network scan. This method is asynchronous, so the network scan results will be returned by callback. The returned NetworkScan will contain a callback method which can be used to stop the scan.

Requires Permission: `MODIFY_PHONE_STATE` or that the calling app has carrier privileges (see `hasCarrierPrivileges()`) and `Manifest.permission.ACCESS_FINE_LOCATION` if includeLocationData is set to `INCLUDE_LOCATION_DATA_FINE` . If the system-wide location switch is off, apps may still call this API, with the following constraints:

1. The app must hold the `android.permission.NETWORK_SCAN` permission.
2. The app must not supply any specific bands or channels to scan.
3. The app must only specify MCC/MNC pairs that are associated to a SIM in the device.
4. Returned results will have no meaningful info other than signal strength and MCC/MNC info.

.

Requires `Manifest.permission.MODIFY_PHONE_STATE`
Requires the `PackageManager#FEATURE_TELEPHONY_RADIO_ACCESS` feature which can be detected using `PackageManager.hasSystemFeature(String)` .

Parameters	
`includeLocationData`	`int` : Specifies if the caller would like to receive location related information. If this parameter is set to `INCLUDE_LOCATION_DATA_FINE` then the application will be checked for `Manifest.permission.ACCESS_FINE_LOCATION` permission and available location related

	information received during network scan will be sent to the caller. Value is one of the following: • `INCLUDE_LOCATION_DATA_NONE` • `INCLUDE_LOCATION_DATA_COARSE` • `INCLUDE_LOCATION_DATA_FINE`
`request`	`NetworkScanRequest` : Contains all the RAT with bands/channels that need to be scanned. This value cannot be `null` .
`executor`	`Executor` : The executor through which the callback should be invoked. Since the scan request may trigger multiple callbacks and they must be invoked in the same order as they are received by the platform, the user should provide an executor which executes tasks one at a time in serial order. This value cannot be `null` .
`callback`	`TelephonyScanManager.NetworkScanCallback` : Returns network scan results or errors. This value cannot be `null` .

Returns	
`NetworkScan`	A NetworkScan obj which contains a callback which can be used to stop the scan. This value may be `null` .

sendVisualVoicemailSms

```
public void sendVisualVoicemailSms (String number,
                int port,
                String text,
                PendingIntent sentIntent)
```

Send a visual voicemail SMS. The caller must be the current default dialer. A `VisualVoicemailService` uses this method to send a command via SMS to the carrier's visual voicemail server. Some examples for carriers using the OMTP standard include activating and deactivating visual voicemail, or requesting the current visual voicemail provisioning status. See the OMTP Visual Voicemail specification for more information on the format of these SMS messages.

Requires Permission: `SEND_SMS`
Requires the `PackageManager#FEATURE_TELEPHONY_CALLING` feature which can be detected using `PackageManager.hasSystemFeature(String)` .

Parameters	
`number`	`String` : The destination number.
`port`	`int` : The destination port for data SMS, or 0 for text SMS.
`text`	`String` : The message content. For data sms, it will be encoded as a UTF-8 byte stream.
`sentIntent`	`PendingIntent` : The sent intent passed to the `SmsManager`

See also:

- `SmsManager.sendDataMessage(String,String,short,byte[],PendingIntent,PendingIntent)`

- `SmsManager.sendTextMessage(String,String,String,PendingIntent,PendingIntent)`

setForbiddenPlmns

```
public int setForbiddenPlmns (List<String> fplmns)
```

Replace the contents of the forbidden PLMN SIM file with the provided values. Passing an empty list will clear the contents of the EFfplmn file. If the provided list is shorter than the size of EFfplmn, then the list will be padded up to the file size with 'FFFFFF'. (required by 3GPP TS 31.102 spec 4.2.16) If the list is longer than the size of EFfplmn, then the file will be written from the beginning of the list up to the file size.

Requires Permission: `MODIFY_PHONE_STATE` or that the calling app has carrier privileges (see `hasCarrierPrivileges()`).
Requires `Manifest.permission.MODIFY_PHONE_STATE`
Requires the `PackageManager#FEATURE_TELEPHONY_SUBSCRIPTION` feature which can be detected using `PackageManager.hasSystemFeature(String)` .

Parameters	
`fplmns`	`List` : a list of PLMNs to be forbidden. This value cannot be `null` .
Returns	
`int`	number of PLMNs that were successfully written to the SIM FPLMN list. This may be less than the number of PLMNs passed in where the SIM file does not have enough room for all of the values passed in. Return -1 in the event of an unexpected failure

setOperatorBrandOverride

```
public boolean setOperatorBrandOverride (String brand)
```

Override the branding for the current ICCID. Once set, whenever the SIM is present in the device, the service provider name (SPN) and the operator name will both be replaced by the brand value input. To unset the value, the same function should be called with a null brand value.

Requires Permission: `Manifest.permission.MODIFY_PHONE_STATE` or that the calling app has carrier privileges (see `hasCarrierPrivileges()`).
Requires the `PackageManager#FEATURE_TELEPHONY_SUBSCRIPTION` feature which can be detected using `PackageManager.hasSystemFeature(String)` .

Parameters	
`brand`	`String` : The brand name to display/set.
Returns	
`boolean`	true if the operation was executed correctly.

setPreferredOpportunisticDataSubscription

```
public void setPreferredOpportunisticDataSubscription (int subId,
                boolean needValidation,
```

```
                Executor executor,
                Consumer<Integer> callback)
```

Set preferred opportunistic data subscription id. Switch internet data to preferred opportunistic data subscription id. This api can result in lose of internet connectivity for short period of time while internet data is handed over.

Requires that the calling app has carrier privileges on both primary and secondary subscriptions (see `hasCarrierPrivileges()`), or has permission `MODIFY_PHONE_STATE`.
Requires the `PackageManager#FEATURE_TELEPHONY_DATA` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Parameters	
`subId`	`int`: which opportunistic subscription `SubscriptionManager.getOpportunisticSubscriptions` is preferred for cellular data. Pass `SubscriptionManager.DEFAULT_SUBSCRIPTION_ID` to unset the preference
`needValidation`	`boolean`: whether validation is needed before switch happens.
`executor`	`Executor`: The executor of where the callback will execute. This value may be `null`. Callback and listener events are dispatched through this `Executor`, providing an easy way to control which thread is used. To dispatch events through the main thread of your application, you can use `Context.getMainExecutor()`. Otherwise, provide an `Executor` that dispatches to an appropriate thread.
`callback`	`Consumer`: Callback will be triggered once it succeeds or failed. See the `SET_OPPORTUNISTIC_SUB_*` constants for more details. Pass null if don't care about the result.

uploadCallComposerPicture

```
public void uploadCallComposerPicture (InputStream pictureToUpload,
                String contentType,
                Executor executor,
                OutcomeReceiver<ParcelUuid, TelephonyManager.CallComposerException> callback)
```

Uploads a picture to the carrier network for use with call composer. This method allows a dialer app to upload a picture to the carrier network that can then later be attached to an outgoing call. In order to attach the picture to a call, use the `ParcelUuid` returned from `callback` upon successful upload as the value to `TelecomManager.EXTRA_OUTGOING_PICTURE`. This functionality is only available to the app filling the `RoleManager.ROLE_DIALER` role on the device. This functionality is only available when `CarrierConfigManager.KEY_SUPPORTS_CALL_COMPOSER_BOOL` is set to `true` in the bundle returned from `getCarrierConfig()`.
Requires the `PackageManager#FEATURE_TELEPHONY_CALLING` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Parameters	
`pictureToUpload`	`InputStream`: An `InputStream` that supplies the bytes representing the picture to upload. The client bears responsibility for closing this stream after `callback` is called with success or failure. Additionally, if the stream supplies more bytes than the return value of `getMaximumCallComposerPictureSize()`, the upload will be aborted and the callback will be

	called with an exception containing `CallComposerException.ERROR_FILE_TOO_LARGE`. This value cannot be `null`.
`contentType`	`String`: The MIME type of the picture you're uploading (e.g. image/jpeg). The list of acceptable content types can be found at 3GPP TS 26.141 sections 4.2 and 4.3. This value cannot be `null`.
`executor`	`Executor`: The `Executor` on which the `pictureToUpload` stream will be read, as well as on which the callback will be called. Callback and listener events are dispatched through this `Executor`, providing an easy way to control which thread is used. To dispatch events through the main thread of your application, you can use `Context.getMainExecutor()`. Otherwise, provide an `Executor` that dispatches to an appropriate thread. This value cannot be `null`.
`callback`	`OutcomeReceiver`: A callback called when the upload operation terminates, either in success or in error. This value cannot be `null`.

uploadCallComposerPicture

```
public void uploadCallComposerPicture (Path pictureToUpload,
                String contentType,
                Executor executor,
                OutcomeReceiver<ParcelUuid, TelephonyManager.CallComposerException> callback)
```

Uploads a picture to the carrier network for use with call composer.
Requires the `PackageManager#FEATURE_TELEPHONY_CALLING` feature which can be detected using `PackageManager.hasSystemFeature(String)`.

Parameters	
`pictureToUpload`	`Path`: Path to a local file containing the picture to upload. This value cannot be `null`.
`contentType`	`String`: The MIME type of the picture you're uploading (e.g. image/jpeg). This value cannot be `null`.
`executor`	`Executor`: The `Executor` on which the `pictureToUpload` file will be read from disk, as well as on which `callback` will be called. Callback and listener events are dispatched through this `Executor`, providing an easy way to control which thread is used. To dispatch events through the main thread of your application, you can use `Context.getMainExecutor()`. Otherwise, provide an `Executor` that dispatches to an appropriate thread. This value cannot be `null`.
`callback`	`OutcomeReceiver`: A callback called when the upload operation terminates, either in success or in error. This value cannot be `null`.
