palmsource™

Telephony and SMS

Exploring Palm OS®

Written by Christopher Bey and Brent Gossett
Technical assistance from Bertrand Aygon, Hatem Oueslati, and Alain Basty.

Exploring Palm OS: Telephony and SMS
Document Number 3117-004
November 9, 2004
For the latest version of this document, visit
http://www.palmos.com/dev/support/docs/.

PalmSource, Inc.
1240 Crossman Avenue
Sunnyvale, CA 94089
USA
www.palmsource.com

Table of Contents

About This Document

This book describes the portions of Palm OS® that interact with a mobile telephone and provide Telephony and Short Message Service (SMS) capabilities.

This book covers the Palm OS Telephony Manager and the SMS exchange library.

For information on creating mobile phone profiles using the Connection Manager, see *Exploring Palm OS: High-Level Communications*.

IMPORTANT: The *Exploring Palm OS* series is intended for developers creating native applications for Palm OS Cobalt. If you are interested in developing applications that work through PACE and that also run on earlier Palm OS releases, read the latest versions of the *Palm OS Programmer's API Reference* and *Palm OS Programmer's Companion* instead.

Who Should Read This Book

You should read this book if you are a Palm OS software developer and you want to do one of the following:

- Write an application that interfaces with a mobile telephone to send or receive calls and data, and manage phone books and message storage.
- Send or receive SMS messages using the SMS exchange library and the Exchange Manager.

You can write a full-featured application without using any of the API described in this book. Beginning Palm OS developers may want to delay reading this book until they gain a better understanding of the fundamentals of Palm OS application development. Instead, consider reading *Exploring Palm OS: Programming Basics* to gain a good understanding of event management and *Exploring Palm OS: User Interface* to learn about events generated by standard UI controls. Come back to this book when you find you need to use the telephony and SMS services.

What This Book Contains

This book contains the following information:

- Part I, "Telephony Manager," contains information on the Connection Manager:
  - Chapter 1, "Telephony Service Types," on page 3 describes the component parts of the telephony API.
  - Chapter 2, "Using the Telephony API," on page 5 describes how to use the telephony API in your applications.
  - Chapter 3, "Summary of the Telephony Manager," on page 13 summarizes the Telephony Manager functions and macros.
  - Chapter 4, "Telephony Manager Reference," on page 19 describes the telephony APIs.
- Part II, "SMS Exchange Library," contains information on the SMS exchange library API:
  - Chapter 5, "SMS Exchange Library Reference," on page 269 describes the SMS exchange library APIs.

Changes to This Book

3117-004

- Added descriptions of new Telephony Manager APIs in Chapter 4, "Telephony Manager Reference," on page 19. These additions include support for the phone MUX, GPRS, and Card Application Toolkit (CAT) features added in Palm OS Cobalt version 6.1.

3117-003

- Bug fix in signal levels returned by `TelNwkGetSignalLevel()`, and other minor corrections.

3117-002

- Minor bug fixes and editorial corrections.

3117-001

- Initial version.

Additional Resources

- Documentation

  PalmSource publishes its latest versions of this and other documents for Palm OS developers at

  http://www.palmos.com/dev/support/docs/

- Training

  PalmSource and its partners host training classes for Palm OS developers. For topics and schedules, check

  http://www.palmos.com/dev/training

- Knowledge Base

  The Knowledge Base is a fast, web-based database of technical information. Search for frequently asked questions (FAQs), sample code, white papers, and the development documentation at

  http://www.palmos.com/dev/support/kb/

Part I Telephony Manager

The Telephony Manager provides communication between Palm OS® applications and phone hardware.

1

Telephony Service Types

The telephony API organizes functions within sets called **service sets**. Each service set contains a related set of functions that may or may not be available on a particular mobile device or network. You should use the `TelIsServiceAvailable()` function to determine if a service set is supported in the current environment, and you should use the `TelIsFunctionSupported()` function to determine if a specific function is supported in the current environment.

NOTE: Sometimes a service set is supported, but not all of the functions in that service set are supported. See Testing the Telephony Environment for more information.

Each function in the telephony API is prefixed with `Tel`; each telephony service set adds an additional 3 characters to the prefix. Table 1.1 describes the telephony service sets.

Table 1.1 Telephony API service sets

Service set	Functionality	Service prefix
Basic	Basic functions that are always available.	`Tel`
Configuration	Services that allow you to configure phones, including SMS configuration.	`TelCfg`
Emergency calls	Emergency call handling.	`TelEmc`
Information	Functions to retrieve information about the current phone.	`TelInf`

Table 1.1 Telephony API service sets *(continued)*

Service set	Functionality	Service prefix
Network	Functions the provide network-oriented services, including authorized networks, current network, signal level, and search mode information.	`TelNwk`
OEM	A function that allows hardware manufacturers to extend the Telephony Manager. Each manufacturer can provide a specific set of OEM functions for a particular device.	`TelOem`
Phone book	Functions to access the phone's SIM and address book, including the ability to create, view, and delete phone book entries.	`TelPhb`
Power	Power supply related functions.	`TelPow`
Security	Functions that provide PIN code management and related services for phone and SIM security-related features.	`TelSty`
Short Message Service	Services to handle Short Message Service (SMS) and to enable the reading, sending, and deleting of short messages.	`TelSms`
Sound	Phone sound management related to muting.	`TelSnd`
Speech calls	Functions to handle the sending and receiving of speech calls. This service also includes functions that handle playing DTMF tones.	`TelSpc`

2 Using the Telephony API

This chapter describes how to use the Telephony API.

Note that the only network supported in this release is GSM/GPRS.

Opening the Telephony Manager Library

Before you can use the Telephony Manager library, you must open it by calling `TelOpen()` or `TelOpenPhoneProfile()`. The library is automatically loaded by the system upon the first telephony function call.

When opened, the Telephony Manager library uses the Connection Manager to open an internal component known as the Telephony Server, which interfaces to the phone drivers. The Telephony Server retrieves information about the needed drivers through the Connection Manager profile.

The particular Connection Manager phone profile that is used depends on how you open the Telephony Manager:

- If you call `TelOpen()`, the phone profile is automatically selected by the Telephony Manager via a call to `CncProfileFindFirst()`. This finds the first telephony profile that is usable and available in the list of telephony profiles.
- If you call `TelOpenPhoneProfile()`, you select the phone profile by passing its identifier to this function.

Closing the Telephony Manager Library

When you are done with the library, you should close it by calling the `TelClose()` function, which releases any resources associated with your use of the Telephony Manager.

Using Synchronous and Asynchronous Calls

Almost all of the telephony functions can be called either synchronously or asynchronously. If you call a function synchronously, it blocks until it completes or an error occurs.

If you call a function asynchronously, it returns immediately and your application receives an event to notify it that the function has completed. The event that you receive contains status and other information returned by the function. For more information about telephony events, see "Telephony Events" on page 10.

You can cancel an asynchronous function call that is in progress by calling `TelCancel()`.

This section provides a simple example of calling the `TelNwkGetStatus()` function both synchronously and asynchronously to illustrate the difference.

When you call a function synchronously, you need to test the result value returned by the function to determine if the call was successful. For example, the code in Listing 2.1 calls the `TelNwkGetStatus()` function synchronously.

Listing 2.1 Calling a function synchronously

```
status_t err = errNone;
int32_t sTelDescId;
uint8_t sNetworkStatus;
err = TelNwkGetStatus(sTelDescId, &sNetworkStatus, NULL)
printf("Result of getting network status is %d", err);
```

To call the same function asynchronously, you specify a transaction ID in the call, instead of specifying `NULL` as the last argument. The transaction ID (`sNwkStatusTransId` in Listing 2.2) is a pointer to an unsigned integer value that is filled in by the Telephony Manager with a value associated with the asynchronous operation that is begun. This same ID value is found in the `transId` field of the event you receive when the operation completes.

Listing 2.2 Calling a function asynchronously

```
status_t err = errNone;
int32_t sTelDescId;
uint8_t sNetworkStatus;
uint16_t sNwkStatusTransId = kTelInvalidTransId;
err = TelNwkGetStatus(sTelDescId, &sNetworkStatus, &sNwkStatusTransId)
...
// Telephony application event handler; called from event loop
static void MyProcessTelephonyEvent(TelEventType *eventP)
{
  switch( eventP->functionId )
  {
...
    case kTelNwkGetStatusMessage:
      printf("Result of function call is %d", eventP->returnCode);
      printf("Network status is %d", eventP->paramP);
      break;
...
  }
```

Using Data Structures With Variably-sized Fields

Many of the telephony functions use data structures that have variably-sized buffer fields. For example, the `TelNwkGetLocation()` function uses the `TelNwkLocationType` structure, which contains two such fields.

```
typedef struct _TelNwkLocationType {
  char *areaCodeP;
  size_t areaCodeSize;
  char *cellIdP;
  size_t cellIdSize;
} TelNwkLocationType;
```

The `areaCodeP` and `cellIdP` buffers are variable-sized strings that you allocate in the heap. When you initialize one of these structures to pass to the `TelNwkGetLocation()` function, you must preallocate the buffers and store the allocated size in the corresponding size fields.

The following code sample initializes a `TelNwkLocationType` data structure and passes it to the `TelNwkGetLocation()` function to retrieve the network location.

```
#define maxAreaCodeSize 5
#define maxCellIdSize 30
TelNwkLocationType myLoc;

myLoc.areaCodeP = MemPtrNew(maxAreaCodeSize);
myLoc.areaCodeSize = maxAreaCodeSize;
myLoc.cellIdP = MemPtrNew(maxCellIdSize);
myLoc.cellIdSize = maxCellIdSize;
err = TelNwkGetLocation(sTelDescId, &myLoc, NULL);
```

Upon return from the function, the buffer fields are filled in, and the size fields contain the actual number of bytes that were stored into the buffer fields.

If the allocated size of a buffer is not large enough to contain the entire value, the function does the following:

- Returns the `telErrBufferSize` error.

- Fills the buffer with as much data as it can, and truncates the data that does not fit. If the data ends with a null terminator and is truncated, the null terminator is retained.
- Sets the value of the size field to the actual size required to contain all of the data.

Note that for string buffers, the size includes the byte required for the null terminator character.

NOTE: When you call a function asynchronously, the `telErrBufferSize` error is returned in the `returnCode` field of the event you receive upon completion of the function's execution.

Also, when you call a function asynchronously, it is your responsibility to ensure that any data structures used by the function remain in memory until you receive the completion event. At that time, you are responsible for freeing the memory for any buffers you allocated.

Testing the Telephony Environment

Before running your application, you need to verify that the environment in which it is running (the Palm Powered™ device and the telephone) supports the facilities that your application needs. The Telephony Manager allows you to determine if a specific service set is available with `TelIsServiceAvailable()`, and also allows you to determine if a specific function call is supported with `TelIsFunctionSupported()`.

Alternatively, there are a series of macros that you can use to check if a service is available (`TelIs`*`ServiceName`*`ServiceAvailable()`), or if a function is available (`TelIs`*`FunctionName`*`Supported()`).

The code excerpt in Listing 2.3 shows how to use the macros to verify that the environment supports particular phone book capabilities. The code first tests for the availability of the phone book service set, and then determines if several specific functions are supported.

Listing 2.3 Testing for the presence of specific capabilities

```
// Test if phone book capabilities are present
err = TelIsPhbServiceAvailable(sTelDescId);
if (err != errNone)
  return err;

// Check that this phone supports adding entries
err = TelIsPhbAddEntrySupported(sTelDescId);
if (err != errNone)
  return err;

// Check that this phone supports selecting a phone book
err = TelIsPhbSetPhonebookSupported(sTelDescId);
if (err != errNone)
  return err;

// Check that this phone supports getting entries list
err = TelIsPhbGetEntriesSupported(sTelDescId);
if (err != errNone)
  return err;

// Check that this phone supports getting an entry
err = TelIsPhbGetEntrySupported(sTelDescId);
if (err != errNone)
  return err;

// Check that this phone supports deleting an entry
err = TelIsPhbDeleteEntrySupported(sTelDescId);
return err;
```

Telephony Events

The Telephony Manager sends telephony events to an application through its event loop or via notifications sent by the Notification Manager.

Events sent via the event loop are mainly for the completion of asynchronous function calls. Applications can call `TelEvtGetEvent()` to receive both system events and telephony events in the main event loop. To receive only telephony events, use `TelEvtGetTelephonyEvent()`. Telephony events have the event type `kTelTelephonyEvent`.

Events sent via notifications are for many other kinds of telephony events such as an incoming SMS message, a call connection, battery status change, etc. These are communicated via a notification of the type `kTelTelephonyNotification`. Applications that want to receive such telephony notifications must register with the Notification Manager.

Sleep and Wake

When the Telephony Server exchanges data with the mobile phone, the device is prevented from sleeping (`EvtResetAutoOffTimer()` is called internally). This means that the device won't go to sleep when data is being sent or received.

If the user switches off the device during data exchange, the connection is stopped, and all of the pending commands are canceled.

3

Summary of the Telephony Manager

Telephony Manager Functions and Macros

Basic Functions

`TelCancel()`	`TelIsOemServiceAvailable()`
`TelClose()`	`TelIsPhbServiceAvailable()`
`TelCncClose()`	`TelIsPowServiceAvailable()`
`TelCncGetStatus()`	`TelIsServiceAvailable()`
`TelCncOpen()`	`TelIsSmsServiceAvailable()`
`TelEvtGetEvent()`	`TelIsSndServiceAvailable()`
`TelEvtGetTelephonyEvent()`	`TelIsSpcServiceAvailable()`
`TelIsCfgServiceAvailable()`	`TelIsStyServiceAvailable()`
`TelIsEmcServiceAvailable()`	`TelOpen()`
`TelIsFunctionSupported()`	`TelOpenPhoneProfile()`
`TelIsInfServiceAvailable()`	`TelTestPhoneDriver()`
`TelIsNwkServiceAvailable()`	`TelUiManageError()`

Telephony Manager Functions and Macros *(continued)*	
Card Application Toolkit	
TelCardGetFile()	TelCatNotifyCardOfEvent()
TelCatCallAction()	TelCatSetCmdResponse()
TelCatGetCmdParameters()	TelCatSetConfig()
TelCatGetConfig()	TelCatTerminate()
TelCatMenuSelection()	
Configuration	
TelCfgGetAlertSoundMode()	TelCfgGetVoiceMailNumber()
TelCfgGetCallForwarding()	TelCfgSetAlertSoundMode()
TelCfgGetCallIdRestrictionStatus()	TelCfgSetCallForwarding()
TelCfgGetLoudspeakerVolumeLevel()	TelCfgSetCallIdRestrictionStatus()
TelCfgGetLoudspeakerVolumeLevelRange()	TelCfgSetLoudspeakerVolumeLevel()
TelCfgGetPhoneNumber()	TelCfgSetPhoneNumber()
TelCfgGetRingerSoundLevel()	TelCfgSetRingerSoundLevel()
TelCfgGetRingerSoundLevelRange()	TelCfgSetSmsCenter()
TelCfgGetSmsCenter()	TelCfgSetVibratorMode()
TelCfgGetVibratorMode()	TelCfgSetVoiceMailNumber()
Emergency Call	
TelEmcDial()	

Telephony Manager Functions and Macros *(continued)*

GPRS

TelGprsGetAttach()	TelGprsGetQosRequested()
TelGprsGetAvailableContextId()	TelGprsGetSmsService()
TelGprsGetContext()	TelGprsSetAttach()
TelGprsGetDataCounter()	TelGprsSetContext()
TelGprsGetDefinedCids()	TelGprsSetEventReporting()
TelGprsGetEventReporting()	TelGprsSetNwkRegistration()
TelGprsGetNwkRegistration()	TelGprsSetPdpActivation()
TelGprsGetPdpActivation()	TelGprsSetQosMinimum()
TelGprsGetPdpAddress()	TelGprsSetQosRequested()
TelGprsGetQosCurrent()	TelGprsSetSmsService()
TelGprsGetQosMinimum()	

Information

TelInfGetCallsDuration()	TelInfResetCallsDuration()
TelInfGetCallsList()	TelInfResetCallsList()
TelInfGetIdentification()	

Telephony Manager Functions and Macros *(continued)*	
Network Interface	
TelNwkAddPreferredOperator()	TelNwkGetRegistrationMode()
TelNwkCancelUssd()	TelNwkGetSignalLevel()
TelNwkCheckUssd()	TelNwkGetStatus()
TelNwkDeletePreferredOperator()	TelNwkGetType()
TelNwkGetLocation()	TelNwkReceiveUssd()
TelNwkGetOperator()	TelNwkSendUssd()
TelNwkGetOperators()	TelNwkSetOperator()
TelNwkGetPreferredOperators()	TelNwkSetRegistration()
TelNwkGetProviderId()	
OEM	
TelOemCall()	
Phone Book	
TelPhbAddEntry()	TelPhbGetPhonebook()
TelPhbDeleteEntry()	TelPhbGetPhonebooks()
TelPhbGetEntries()	TelPhbSetPhonebook()
TelPhbGetEntry()	
Phone MUX	
TelMuxChanAllocate()	TelMuxChanSetId()
TelMuxChanFree()	TelMuxEnable()
Power Management	
TelPowGetBatteryChargeLevel()	TelPowSetPhoneFunctionality()
TelPowGetBatteryConnectionStatus()	

Telephony Manager Functions and Macros *(continued)*

Short Message Service

TelSmsDeleteMessage()	TelSmsReadMessage()
TelSmsGetDataMaxSize()	TelSmsReadMessages()
TelSmsGetStorage()	TelSmsSendMessage()
TelSmsGetStorages()	TelSmsSetStorage()
TelSmsGetUniquePartId()	

Sound

TelSndGetMuteStatus()	TelSndSetMuteStatus()

Speech Calls

TelSpcAcceptCall()	TelSpcHoldActiveCalls()
TelSpcAddHeldCall()	TelSpcInitiateCall()
TelSpcGetCall()	TelSpcPlayTone()
TelSpcGetCalls()	TelSpcPrivateCall()
TelSpcGetToneDuration()	TelSpcReleaseCall()
TelSpcGetToneDurationRange()	TelSpcSetToneDuration()

Security

TelStyChangeFacilityPassword()	TelStyGetFacility()
TelStyEnterAuthentication()	TelStyLockFacility()
TelStyGetAuthenticationStatus()	TelStyUnlockFacility()
TelStyGetFacilities()	

4

Telephony Manager Reference

This chapter describes the Telephony Manager APIs and is divided into the following sections:

The header files `TelephonyLib.h` and `TelephonyLibTypes.h` declare the API that this chapter describes.

Telephony Manager Structures and Types

TelCardFileType Struct

Purpose Holds the content of and information about a file on a card.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCardFileType {
   uint16_t *pathP;
   uint8_t *bufP;
   size_t bufSize;
   size_t byteCount;
   uint16_t partOffset;
   uint16_t partSize;
   uint16_t fileSize;
   uint8_t fileStruct;
   uint8_t mode;
   uint8_t pathCount;
   uint8_t recId;
   uint8_t recSize;
   uint8_t pad;
} TelCardFileType, *TelCardFilePtr
```

Fields

`pathP`
: A pointer to the absolute path of the file to read in the SIM. For example:
`{ 0x3F00, 0x7F20, 0x6F21 }`.

  Consists of file identifiers from the Master File to the Elementary File to be accessed.

`bufP`
: A pointer to a buffer to be filled in with the content of the requested file.

`bufSize`
: The size of the `bufP` buffer.

`byteCount`
: The number of bytes in the `bufP` buffer. This is the number of bytes that were actually read from the file.

`partOffset`
: The offset of the part of the file that was requested.

`partSize`
: The size of the requested part of the file.

`fileSize`
: The Elementary File size.

`fileStruct`
: The Elementary File structure. One of the values described in "Card Elementary File Structures" on page 94.

`mode`
: The file access mode. One of the values described in "Card Elementary File Access Modes" on page 94.

`pathCount`
: The number of file identifiers in `pathP`.

`recId`
: The identifier of the record to be read. Values range from 1 to 254.

`recSize`
: The size of a record in bytes. This value is 0 if the file is not a Linear Fixed or a Cyclic Elementary File.

`pad`
: Padding bytes

Comments Used by `TelCardGetFile()`.

TelCatBufferType Struct

Purpose Specifies the parameters that the Card Application Toolkit's Send Data, Send DTMF, Send USSD, Send SS, Run AT Command commands use.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCatBufferType {
   uint8_t *bufferP;
   uint8_t bufferSize;
   uint8_t other;
   uint16_t pad;
} TelCatBufferType
```

Fields

`bufferP`
: A pointer to the data buffer.

`bufferSize`
: The size of `bufferP` in bytes.

`other`
: Other parameter specific to the command, if any.

`pad`
: Padding bytes.

Comments Used by `TelCatGetCmdParameters()` and `TelCatSetCmdResponse()` depending on the `cmdId` field of the `TelCatCmdParamsType` or `TelCatCmdResponseType` structure.

TelCatCmdParamsType Struct

Purpose Holds the parameters of a proactive card command.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCatCmdParamsType {
   MemPtr cmdParamP;
   size_t cmdParamSize;
   char *textP;
   uint8_t textSize;
   uint8_t iconId;
   uint8_t cmdId;
   Boolean explicitIcon;
   Boolean noResponse;
   uint8_t other1;
   uint16_t other2;
} TelCatCmdParamsType
```

Fields

`cmdParamP`
: A pointer to a structure associated with the command in the `cmdId` field. Almost all CAT commands use this field to hold parameters.

`cmdParamSize`
: Size of the parameter buffer for the command specified in the `cmdId` field.

`textP`
: A pointer to the text to display.

`textSize`
: The size of the `textP` buffer.

`iconId`
: The icon identifier.

`cmdId`
: The command ID. One of the values described in "Card Command IDs" on page 92.

`explicitIcon`
: If `true`, indicates that the icon is explicit.

`noResponse`
: If `true`, the command does not need a response.

`other1`
: Other command-dependent parameter.

`other2`
Other command-dependent parameter.

Comments Used by `TelCatGetCmdParameters()`.

TelCatCmdResponseType Struct

Purpose Holds the response of a proactive card command.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCatCmdResponseType {
   char *respP;
   uint32_t other;
   size_t respSize;
   uint8_t cmdId;
   uint8_t respType;
   uint8_t resCode;
   uint8_t addInfo;
} TelCatCmdResponseType
```

Fields

`respP`
A pointer to a buffer that holds the response text.

`other`
Other command-dependent parameter.

`respSize`
The size in bytes of the response text in `respP`.

`cmdId`
The command ID. One of the values described in "Card Command IDs" on page 92.

`respType`
The expected response type. One of the values described in "Card Get Inkey and Get Input Command Response Types" on page 97.

`resCode`
The result codes applicable to the command specified in the `cmdId` field. One of the values described in "Card General Result Codes" on page 95.

`addInfo`
An "additional information" code. One of the `kTelCatAdd<xxx>` values, depending on the command.

Comments Used by `TelCatSetCmdResponse()`.

TelCatConfigType Struct

Purpose Holds information about Card Application Toolkit (CAT) features and the language setting.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCatConfigType {
   uint8_t *profileP;
   uint32_t profileSize;
   char lanCode[2];
   uint8_t mode;
   uint8_t padding;
} TelCatConfigType, *TelCatConfigPtr
```

Fields

`profileP`
A pointer to a buffer that holds standard Terminal Profile parameters.

`profileSize`
The size of the `profileP` buffer in bytes.

`lanCode`
An ISO 639 language code.

`mode`
Enable or disable the presentation of CAT unsolicited result codes. Set this field to 1 to enable. For example, enable this mode for a browser.

`padding`
Padding bytes.

Comments Used by `TelCatGetConfig()` and `TelCatSetConfig()`.

TelCatDisplayTextType Struct

Purpose Specifies the parameters that the Card Application Toolkit's Display Text command uses.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCatDisplayTextType {
   Boolean priority;
   Boolean clearAfterDelay;
   Boolean immediateResponse;
} TelCatDisplayTextType
```

Fields

`priority`
If `true`, then the priority level is high; otherwise, the priority level is normal.

`clearAfterDelay`
If `true`, then clear the text after a delay; otherwise, wait for the user's action.

`immediateResponse`
If `true`, then send a response to the card as soon as possible.

Comments Used by `TelCatGetCmdParameters()` and `TelCatSetCmdResponse()` depending on the `cmdId` field of the `TelCatCmdParamsType` or `TelCatCmdResponseType` structure.

TelCatEventToCardType Struct

Purpose Specifies to the card an event that occurred in Palm OS®.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCatEventToCardType {
   uint8_t evtCode;
   char lanCode[2];
   uint8_t browserTerminationCause;
} TelCatEventToCardType
```

Fields

`evtCode`
An event download code. One of the values described in "Card Set Up Event List Command Events" on page 101.

`lanCode`
An ISO 639 language code.

`browserTerminationCause`
A browser termination cause code. One of the values described in "Card Browser Termination Cause Codes" on page 91.

Comments Used by `TelCatNotifyCardOfEvent()`.

TelCatGetInkeyType Struct

Purpose Specifies the parameters that the Card Application Toolkit's Get Inkey command uses.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCatGetInkeyType {
   Boolean helpInfo;
   uint8_t respType;
   uint16_t pad;
} TelCatGetInkeyType
```

Fields

`helpInfo`
If `true`, then help information is provided by the card.

`respType`
The expected response type. One of the values described in "Card Get Inkey and Get Input Command Response Types" on page 97.

`pad`
Padding bytes.

Comments Used by `TelCatGetCmdParameters()` and `TelCatSetCmdResponse()` depending on the `cmdId` field of the `TelCatCmdParamsType` or `TelCatCmdResponseType` structure.

TelCatGetInputType Struct

Purpose Specifies the parameters that the Card Application Toolkit's Get Input command uses.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCatGetInputType {
   char *defRespP;
   size_t defRespSize;
   Boolean hideUserInput;
   Boolean helpInfo;
   uint8_t minRespLength;
   uint8_t maxRespLength;
   uint8_t respType;
   uint8_t pad1;
   uint16_t pad2;
} TelCatGetInputType
```

Fields

`defRespP`
A pointer to the default response text to propose.

`defRespSize`
The size of `defRespP` in bytes.

`hideUserInput`
If `true`, then mask the data entered by the user.

`helpInfo`
If `true`, then help information is provided by the card.

`minRespLength`
The minimum response length, in characters.

`maxRespLength`
The maximum response length, in characters.

`respType`
The expected response type. One of the values described in "Card Get Inkey and Get Input Command Response Types" on page 97.

`pad1`
Padding bytes.

`pad2`
Padding bytes.

Comments Used by `TelCatGetCmdParameters()` and `TelCatSetCmdResponse()` depending on the `cmdId` field of the `TelCatCmdParamsType` or `TelCatCmdResponseType` structure.

TelCatItemListType Struct

Purpose Specifies the parameters that the Card Application Toolkit's Select Item and Setup Menu commands use.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCatItemListType {
   TelCatItemType *itemsP;
   uint8_t itemCount;
   Boolean softKey;
   Boolean helpInfo;
   uint8_t defItemId;
} TelCatItemListType
```

Fields

`itemsP`
: A pointer to a list of menu items. Each item is defined by a `TelCatItemType` structure.

`itemCount`
: The number of items in `itemsP`.

`softKey`
: If `true`, then the item can be selected by tapping on its icon.

`helpInfo`
: If `true`, then help information is provided by the card.

`defItemId`
: The identifier of the item that should be pre-selected.

Comments Used by `TelCatGetCmdParameters()` and `TelCatSetCmdResponse()` depending on the `cmdId` field of the `TelCatCmdParamsType` or `TelCatCmdResponseType` structure.

TelCatItemType Struct

Purpose Specifies the parameters that the Card Application Toolkit's Select Item and Setup Menu commands use.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCatItemType {
   char *nameP;
   size_t nameSize;
   uint8_t id;
   uint8_t iconId;
   Boolean expIcon;
   uint8_t nextActionInd;
} TelCatItemType
```

Fields

`nameP`
: A pointer to the item name.

`nameSize`
: The size of `nameP` in bytes.

`id`
: The item identifier.

`iconId`
: The icon identifier.

`expIcon`
: If `true`, the icon is explicit.

`nextActionInd`
: The identifier of the next command for this item.

Comments Used by `TelCatGetCmdParameters()` and `TelCatSetCmdResponse()` depending on the `cmdId` field of the `TelCatCmdParamsType` or `TelCatCmdResponseType` structure.

TelCatLaunchBrowserType Struct

Purpose Specifies the parameters that the Card Application Toolkit's Launch Browser command uses.

Declared In `TelephonyLibTypes.h`

Prototype
```
typedef struct _TelCatLaunchBrowserType {
   char *urlP;
   size_t urlSize;
   char *gatewayP;
   size_t gatewaySize;
   uint16_t *filePathP;
   uint8_t *prefBearersP;
   uint8_t fileIdCount;
   uint8_t prefBearerCount;
   uint8_t condition;
   uint8_t browserId;
} TelCatLaunchBrowserType
```

Fields

`urlP`
A pointer to the URL.

`urlSize`
The size of `urlP` in bytes.

`gatewayP`
A pointer to the gateway name or proxy identity to be used.

`gatewaySize`
The size of `gatewayP` in bytes.

`filePathP`
A pointer to the concatenated absolute paths of the provisioning Elementary File. This field is `NULL` if no specific file has been specified.

`prefBearersP`
A pointer to a prioritized list of bearer codes. The values are described in "Card Launch Browser Command Bearer Codes" on page 97.

`fileIdCount`
The number of file identifiers in `filePathP`.

`prefBearerCount`
The number of items in `prefBearersP`.

`condition`
The conditions under which to launch the browser. One of the values described in "Card Launch Browser Command Conditions" on page 98.

`browserId`
The browser ID.

Comments Used by `TelCatGetCmdParameters()` and `TelCatSetCmdResponse()` depending on the `cmdId` field of the `TelCatCmdParamsType` or `TelCatCmdResponseType` structure.

TelCatMenuSelectionType Struct

Purpose Specifies a menu selection and the application on the card it applies to.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCatMenuSelectionType {
   uint8_t evtCode;
   uint8_t appId;
   uint16_t pad;
} TelCatMenuSelectionType
```

Fields

`evtCode`
A Menu Selection event code. One of the values described in "Card Menu Selection Event Codes" on page 98.

`appId`
Identifier of the application the menu selection applies to.

`pad`
Padding bytes.

Comments Used by `TelCatMenuSelection()`.

TelCatOpenChanType Struct

Purpose Specifies the parameters that the Card Application Toolkit's Open Channel command uses.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCatOpenChanType {
   char *addressP;
   char *subAddressP;
   char *otherAddressP;
   char *destinationAddressP;
   char *loginP;
   char *passwordP;
   uint8_t *bearerParamsP;
   char *accessPointP;
   uint32_t duration1;
   uint32_t duration2;
   uint16_t bufferSize;
   uint16_t transportPort;
   Boolean onDemand;
   uint8_t bearerCode;
   uint8_t otherAddressType;
   uint8_t destinationAddressType;
   uint8_t transportType;
   uint8_t addressSize;
   uint8_t subAddressSize;
   uint8_t otherAddressSize;
   uint8_t bearerParamsSize;
   uint8_t loginSize;
   uint8_t passwordSize;
   uint8_t destinationAddressSize;
   uint8_t accessPointSize;
   uint8_t pad1;
   uint16_t pad2;
} TelCatOpenChanType
```

Fields

`addressP`
A pointer to the address.

`subAddressP`
A pointer to the subaddress.

`otherAddressP`
A pointer to another address.

`destinationAddressP`
: A pointer to the destination address.

`loginP`
: A pointer to the login.

`passwordP`
: A pointer to the password.

`bearerParamsP`
: A pointer to the bearer parameters.

`accessPointP`
: A pointer to the access point name.

`duration1`
: Duration 1 in milliseconds.

`duration2`
: Duration 2 in milliseconds.

`bufferSize`
: The number of bytes requested by the SIM in an Open Channel command.

`transportPort`
: The transport port.

`onDemand`
: If true, then the link is established immediately.

`bearerCode`
: The bearer code. One of the values described in "Card Launch Browser Command Bearer Codes" on page 97.

`otherAddressType`
: The type of the address specified by `otherAddressP`. One of the values described in "Card Open Channel Command Address and Transport Types" on page 99.

`destinationAddressType`
: The type of the address specified by `destinationAddressP`. One of the values described in "Card Open Channel Command Address and Transport Types" on page 99.

`transportType`
: The type of the address specified by `transportPort`. One of the values described in "Card Open Channel Command Address and Transport Types" on page 99.

`addressSize`
: The size of `addressP` in bytes.

`subAddressSize`
: The size of `subAddressP` in bytes.

`otherAddressSize`
: The size of `otherAddressP` in bytes.

`bearerParamsSize`
: The size of `bearerParamsP` in bytes.

`loginSize`
: The size of `loginP` in bytes.

`passwordSize`
: The size of `passwordP` in bytes.

`destinationAddressSize`
: The size of `destinationAddressP` in bytes.

`accessPointSize`
: The size of `accessPointP` in bytes.

`pad1`
: Padding bytes.

`pad2`
: Padding bytes.

Comments Used by `TelCatGetCmdParameters()` and `TelCatSetCmdResponse()` depending on the `cmdId` field of the `TelCatCmdParamsType` or `TelCatCmdResponseType` structure.

TelCatPlayToneType Struct

Purpose Specifies the parameters that the Card Application Toolkit's Play Tone command uses.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCatPlayToneType {
   uint32_t sndDuration;
   uint8_t sndCode;
   uint8_t pad1;
   uint16_t pad2;
} TelCatPlayToneType
```

Fields

`sndDuration`
The sound duration in milliseconds. Values range from 100 to 15300000. Set to 0 for the default duration.

`sndCode`
One of the values described in "Card Play Tone Command Sound Codes" on page 99.

`pad1`
Padding bytes.

`pad2`
Padding bytes.

Comments Used by `TelCatGetCmdParameters()` and `TelCatSetCmdResponse()` depending on the `cmdId` field of the `TelCatCmdParamsType` or `TelCatCmdResponseType` structure.

TelCatRefreshType Struct

Purpose Specifies the refresh mode that the Card Application Toolkit's Refresh command uses.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCatRefreshType {
   uint16_t *filePathP;
   uint8_t fileIdCount;
   uint8_t opCode;
   uint16_t pad;
} TelCatRefreshType
```

Fields

`filePathP`
A pointer to the concatenated absolute paths of the modified Elementary File, or `NULL` if no file is specified.

`fileIdCount`
The number of file identifiers in `filePathP`.

`opCode`
The operation code. One of the "Card Refresh Command Opcodes" on page 100.

`pad`
Padding bytes.

Comments

Used by `TelCatGetCmdParameters()` and `TelCatSetCmdResponse()` depending on the `cmdId` field of the `TelCatCmdParamsType` or `TelCatCmdResponseType` structure.

TelCatSendShortMessageType Struct

Purpose

Specifies the parameters that the Card Application Toolkit's Send Short Message command uses.

Declared In

`TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCatSendShortMessageType {
   char *addressP;
   uint8_t *TPDUP;
   uint8_t TPDUSize;
   uint8_t addressSize;
   Boolean packingRequired;
   uint8_t pad;
} TelCatSendShortMessageType
```

Fields

`addressP`
A pointer to an optional RP_Destination_Address.

`TPDUP`
A pointer to an SMS transport protocol data unit (TPDU).

`TPDUSize`
The size of `TPDUP` in bytes.

`addressSize`
The size of `addressP` in bytes.

`packingRequired`
: If `true`, then packing is required.

`pad`
: Padding bytes.

Comments Used by `TelCatGetCmdParameters()` and `TelCatSetCmdResponse()` depending on the `cmdId` field of the `TelCatCmdParamsType` or `TelCatCmdResponseType` structure.

TelCatSetUpCallType Struct

Purpose Specifies the call setup that the Card Application Toolkit's Set Up Call command uses.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCatSetUpCallType {
   uint8_t *bearerCapP;
   char *numberP;
   char *userConfTextP;
   char *callEstaTextP;
   size_t userConfTextSize;
   size_t numberSize;
   size_t callEstaTextSize;
   uint8_t userConfIconId;
   Boolean userConfExplicitIcon;
   Boolean autoRedial;
   uint8_t bearerCapSize;
   uint8_t condition;
   uint8_t callEstaIconId;
   Boolean callEstaExplicitIcon;
   uint8_t pad;
} TelCatSetUpCallType
```

Fields

`bearerCapP`
: A pointer to the bearer capability configuration parameters defined by GSM 04.08 5.3.0 section 10.5.4.5.

`numberP`
: A pointer to the number to dial.

`userConfTextP`
: A pointer to the user confirmation text to display. Set to `NULL` if no text is provided.

`callEstaTextP`
: A pointer to the call establishment text to display. Set to `NULL` if no text is provided.

`userConfTextSize`
: The size of `userConfTextP` in bytes.

`numberSize`
: The size of `numberP` in bytes.

`callEstaTextSize`
: The size of `callEstaTextP` in bytes.

`userConfIconId`
: The user confirmation icon ID. Set to 0 if there is no icon.

`userConfExplicitIcon`
: If `true`, the user confirmation icon is explicit.

`autoRedial`
: If `true`, automatic redial is requested by the card.

`bearerCapSize`
: The size of `bearerCapP` in bytes.

`condition`
: The call set up conditions. One of the values described in "Card Set Up Call Command Call Conditions" on page 101.

`callEstaIconId`
: The call establishment icon ID. Set to 0 if there is no icon.

`callEstaExplicitIcon`
: If `true`, the call establishment icon is explicit.

`pad`
: Padding bytes.

Comments Used by `TelCatGetCmdParameters()` and `TelCatSetCmdResponse()` depending on the `cmdId` field of the `TelCatCmdParamsType` or `TelCatCmdResponseType` structure.

TelCatSetUpEventListType Struct

Purpose Specifies the list of events to monitor that the Card Application Toolkit's Set Up Event List command uses.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCatSetUpEventListType {
   uint8_t *eventP;
   uint8_t eventCount;
   uint8_t pad1;
   uint16_t pad2;
} TelCatSetUpEventListType
```

Fields

`eventP`
A pointer to the list of events to be monitored. The values are described in "Card Set Up Event List Command Events" on page 101.

`eventCount`
The number of events in `eventP`. Set this field to 0 to stop monitoring.

`pad1`
Padding bytes.

`pad2`
Padding bytes.

Comments Used by `TelCatGetCmdParameters()` and `TelCatSetCmdResponse()` depending on the `cmdId` field of the `TelCatCmdParamsType` or `TelCatCmdResponseType` structure.

TelCfgCallForwardingType Struct

Purpose Holds information related to call forwarding.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCfgCallForwardingType {
   TelNumberType number;
   TelNumberType subAddr;
   uint8_t reason;
   uint8_t mode;
   uint8_t classType;
   uint8_t time;
   uint8_t status;
   uint8_t padding[3];
} TelCfgCallForwardingType,
*TelCfgCallForwardingPtr
```

Fields

`number`
: A `TelNumberType` structure that holds the forwarding number.

`subAddr`
: A `TelNumberType` structure that holds the forwarding subaddress.

`reason`
: One of the constants described in "Forwarding Reasons" on page 103.

`mode`
: One of the constants described in "Forwarding Modes" on page 103.

`classType`
: Sum of one or more constants described in "“Connection Types” on page 102Forwarding Classes" on page 102.

`time`
: If the value for the `reason` field is `kTelCfgForwardingReasonNoReply`, this specifies the time to wait (in seconds) before forwarding the call. The default is 20 seconds.

`status`
: The value 0 means inactive and 1 means active.

`padding`
: Padding bytes.

Comments Used by the `TelCfgGetCallForwarding()` and `TelCfgSetCallForwarding()` functions.

TelCfgLevelRangeType Struct

Purpose Holds the minimum and maximum volume levels.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCfgLevelRangeType {
   uint8_t min;
   uint8_t max;
   uint8_t padding[2];
} TelCfgLevelRangeType, *TelCfgLevelRangePtr
```

Fields

`min`
Minimum volume range.

`max`
Maximum volume range.

`padding`
Padding bytes.

Comments Used by the `TelCfgGetLoudspeakerVolumeLevelRange()` and `TelCfgGetRingerSoundLevelRange()` functions.

TelCfgPhoneNumberType Struct

Purpose Holds the phone numbers assigned to the mobile equipment (phone).

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelCfgPhoneNumberType {
   TelNumberType voice;
   TelNumberType fax;
   TelNumberType data;
} TelCfgPhoneNumberType, *TelCfgPhoneNumberPtr
```

Fields

`voice`
A `TelNumberType` structure that holds a voice number.

`fax`
A `TelNumberType` structure that holds a fax number.

`data`
A `TelNumberType` structure that holds a data number.

Comments Used by the `TelCfgGetPhoneNumber()` and `TelCfgSetPhoneNumber()` functions.

TelDtcConnectionInfoType Struct

Purpose Holds information for GSM circuit-switched or GPRS data connections.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelDtcConnectionInfoType {
   uint8_t type;
   uint8_t padding[3];
   union {
      TelDtcCsdConnectionType gsmCsd;
      TelDtcGprsConnectionType gprs;
   } connection;
} TelDtcConnectionInfoType,
*TelDtcConnectionInfoPtr
```

Fields

`type`
: One of the constants defined in "Connection Types" on page 102.

`padding`
: Padding bytes.

`connection`
: Connection information, which is one of the following structures: `TelDtcCsdConnectionType` or `TelDtcGprsConnectionType`.

Comments This structure is used by the Telephony Connection Manager plug-in.

TelDtcCsdConnectionType Struct

Purpose Holds information about a circuit-switched data (CSD) call.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelDtcCsdConnectionType {
   uint8_t speed;
   uint8_t service;
   uint8_t connection;
   uint8_t padding;
   TelNumberType dialNumber;
} TelDtcCsdConnectionType, *TelDtcCsdConnectionPtr
```

Fields

`speed`
: One of the values described in "GSM CSD Bearer Service Speeds" on page 113.

`service`
: One of the values described in "GSM CSD Bearer Service Name" on page 113.

`connection`
: One of the values described in "GSM CSD Bearer Service Connection Element" on page 112.

`padding`
: Padding bytes.

`dialNumber`
: A `TelNumberType` structure that describes a phone number.

Comments A substructure of the `TelDtcConnectionInfoType` structure.

TelDtcGprsConnectionType Struct

Purpose Holds information about a GPRS data call.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelDtcGprsConnectionPtr {
   TelGprsContextType context;
   TelGprsQosType qosMinimum;
   TelGprsQosType qosRequested;
} TelDtcGprsConnectionType,
*TelDtcGprsConnectionPtr
```

Fields

`context`
: A `TelGprsContextType` structure that defines the PDP context for a GPRS data call.

`qosMinimum`
: A `TelGprsQosType` structure that defines the minimum quality of service parameters for a GPRS data call.

`qosRequested`
: A `TelGprsQosType` structure that defines the requested quality of service parameters for a GPRS data call.

Comments A substructure of the `TelDtcConnectionInfoType` structure.

TelEventType Struct

Purpose Holds information about a telephony event.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelEventType {
   eventsEnum eType;
   int16_t screenX;
   int16_t screenY;
   Boolean penDown;
   uint8_t tapCount;
   uint16_t padding;
   MemPtr paramP;
   uint16_t functionId;
   uint16_t transId;
   status_t returnCode;
} TelEventType, *TelEventPtr
```

Fields

`eType`
: Type of the event; always set to `kTelTelephonyEvent`.

`screenX`
: Window-relative position of the pen in pixels (number of pixels from the left bound of the window).

  This field is not filled in for telephony events.

`screenY`
: Window-relative position of the pen in pixels (number of pixels from the top left of the window).

  This field is not filled in for telephony events.

`penDown`
: `true` if the pen was down at the time of the event, otherwise `false`.

  This field is not filled in for telephony events.

`tapCount`
: The number of taps received at this location.

  This field is not filled in for telephony events.

`padding`
: Padding bytes, for alignment purposes.

`paramP`
Pointer to the parameter block passed into the asynchronous function call that generated this event.

`functionId`
: One of the `TelMessages` constants, that identifies the asynchronous function whose completion generated this event.

`transId`
: The transaction ID of the operation.

`returnCode`
: The return code of the asynchronously called function. The value of this field is `errNone` upon success or an error code upon failure.

Comments The `TelEvtGetEvent()` and `TelEvtGetTelephonyEvent()` functions both return a `TelEventType` structure to provide information about a telephony-related event.

TelGprsContextType Struct

Purpose Holds information about a GPRS PDP context.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelGprsContextType {
   uint8_t contextID;
   uint8_t pdpType;
   uint8_t dataCompression;
   uint8_t headerCompression;
   char *accessPointNameP;
   size_t accessPointNameSize;
   char *pdpAddressP;
   size_t pdpAddressSize;
   char *OSPIHHostP;
   size_t OSPIHHostSize;
   uint16_t OSPIHPort;
   uint8_t OSPIHProtocol;
   uint8_t padding;
} TelGprsContextType, *TelGprsContextPtr
```

Fields

`contextID`
: A PDP context ID.

`pdpType`
: The PDP type. One of the values described in "GPRS Packet Data Protocols" on page 108.

`dataCompression`
: Data compression settings. `kTelGprsDataCompressionSetOn` or `kTelGprsDataCompressionSetOff` as described in "GPRS Compression Settings" on page 104.

`headerCompression`
: Header compression settings. `kTelGprsHdrCompressionSetOn` or `kTelGprsHdrCompressionSetOff` as described in "GPRS Compression Settings" on page 104.

`accessPointNameP`
: A pointer to a buffer that holds the access point name. If `accessPointNameSize` == 0, then the default APN is requested from the network.

`accessPointNameSize`
: Size of the `accessPointNameP` buffer.

`pdpAddressP`
: A pointer to a buffer that holds the PDP address. If `pdpAddressSize` == 0, then the address is requested from the network.

`pdpAddressSize`
: Size of the `pdpAddressP` buffer.

`OSPIHHostP`
: A pointer to a buffer that holds the OSPIH name. Required only if OSPIH is chosen for the `pdpType` field.

`OSPIHHostSize`
: Size of the `OSPIHHostP` buffer.

`OSPIHPort`
: The TCP or UDP port on Internet Host. One of the values described in "GPRS OSPIH Protocol Settings" on page 108. Required only if OSPIH is chosen for the `pdpType` field.

`OSPIHProtocol`
: The protocol used over IP, TCP or UDP. One of the values described in "GPRS OSPIH Protocol Settings" on page 108. Required only if OSPIH is chosen for the `pdpType` field.

`padding`
Padding bytes.

Comments Used by `TelGprsGetContext()` and `TelGprsSetContext()`.

TelGprsDataCounterType Struct

Purpose Holds the count of data uploaded and downloaded between the Palm Powered™ device and the GPRS network for a given PDP context.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelGprsDataCounterType {
   uint8_t contextID;
   uint8_t padding[3];
   uint32_t ulBytes;
   uint32_t dlBytes;
   uint32_t ulPackets;
   uint32_t dlPackets;
} TelGprsDataCounterType, *TelGprsDataCounterPtr
```

Fields

`contextID`
The context ID.

`padding`
Padding bytes.

`ulBytes`
The number of bytes uploaded to the network.

`dlBytes`
The number of bytes downloaded from the network.

`ulPackets`
The number of packets (NPDUs) uploaded to the network.

`dlPackets`
The number of packets (NPDUs) downloaded from the network.

Comments Used by `TelGprsGetDataCounter()`.

TelGprsDefinedCidsType Struct

Purpose List of defined GPRS PDP context IDs.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelGprsDefinedCidsType {
   size_t cidCount;
   uint8_t *cidsP;
} TelGprsDefinedCidsType, *TelGprsDefinedCidsPtr
```

Fields

`cidCount`
: Number of elements in the array pointed to by the `cidsP` field. On input, specifies the size of the array. Upon return, receives the number of context IDs actually in the `cidsP` array.

`cidsP`
: Upon return, a pointer to an array of context IDs.

Comments Used by `TelGprsGetDefinedCids()`.

TelGprsEventReportingType Struct

Purpose Holds information about the sending of the unsolicited result code `+CGEV:XXX` when certain events occur in the GPRS phone/module or the network.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelGprsEventReportingType {
   uint8_t mode;
   uint8_t buffer;
   uint8_t padding[2];
} TelGprsEventReportingType,
*TelGprsEventReportingPtr
```

Fields

`mode`
: The event reporting mode. One of the mode values described in "GPRS Event Reporting Settings" on page 105.

`buffer`
: An optional value to specify whether to flush or clear buffered unsolicited result codes. One of the buffer values described in "GPRS Event Reporting Settings" on page 105.

`padding`
Padding bytes.

Comments Used by `TelGprsGetEventReporting()` and `TelGprsSetEventReporting()`.

TelGprsNwkRegistrationType Struct

Purpose Holds network registration information about a PDP context.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelGprsNwkRegistrationType {
   uint8_t registrationType;
   uint8_t registrationStatus;
   uint8_t cellSupportingStatus;
   uint8_t padding;
   uint16_t locationAreaCode;
   uint16_t cellId;
} TelGprsNwkRegistrationType,
*TelGprsNwkRegistrationPtr
```

Fields

`registrationType`
The type of registration: network disable, network enable, and cell enable. One of the values described in "GPRS Network Registration Settings" on page 106.

`registrationStatus`
The registration status. One of the values described in "GPRS Network Registration Status" on page 107.

`cellSupportingStatus`
Indicates whether a cell supports GPRS:

`0`
GPRS is not supported.

`1`
GPRS is supported.

`kTelGprsValueUnknown`
Unknown.

`padding`
Padding bytes.

`locationAreaCode`
: Location information.

`cellId`
: Cell ID.

Comments Used by `TelGprsGetNwkRegistration()` and `TelGprsSetNwkRegistration()`.

TelGprsPdpActivationType Struct

Purpose Holds information the activation state of a PDP context.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelGprsPdpActivationType {
   uint8_t contextID;
   uint8_t state;
   uint8_t padding[2];
} TelGprsPdpActivationType,
*TelGprsPdpActivationPtr
```

Fields

`contextID`
: A context ID.

`state`
: The activation state of the PDP context specified in the `contextID` field. One of the values described in "GPRS PDP Activation State" on page 108.

`padding`
: Padding bytes.

Comments Used by `TelGprsGetPdpActivation()` and `TelGprsSetPdpActivation()`.

TelGprsPdpAddressType Struct

Purpose Holds the address of a PDP context specified by its context ID.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelGprsPdpAddressType {
   uint8_t contextID;
   uint8_t padding[3];
   char *pdpAddressP;
   size_t pdpAddressSize;
} TelGprsPdpAddressType, *TelGprsPdpAddressPtr
```

Fields

`contextID`
The context ID.

`padding`
Padding bytes.

`pdpAddressP`
A pointer to a buffer that holds the PDP address.

`pdpAddressSize`
Size of the `pdpAddressP` buffer.

Comments Used by `TelGprsGetPdpAddress()`.

TelGprsQosType Struct

Purpose Holds information about the quality of service.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelGprsQosType {
   uint8_t contextID;
   uint8_t precedence;
   uint8_t delay;
   uint8_t reliability;
   uint8_t peak;
   uint8_t mean;
   uint8_t padding[2];
} TelGprsQosType, *TelGprsQosPtr
```

Fields

`contextID`
The context ID.

`precedence`
One of the precendence values described in "GPRS Quality of Service" on page 109.

`delay`
One of the delay values described in "GPRS Quality of Service" on page 109.

`reliability`
One of the reliability values described in "GPRS Quality of Service" on page 109.

`peak`
One of the peak values described in "GPRS Quality of Service" on page 109.

`mean`
One of the mean values described in "GPRS Quality of Service" on page 109.

`padding`
Padding bytes.

Comments Used by `TelGprsGetQosRequested()` and `TelGprsSetQosRequested()`, `TelGprsGetQosMinimum()` and `TelGprsSetQosMinimum()`, and `TelGprsGetQosCurrent()`.

TelInfCallsDurationType Struct

Purpose Holds call duration information.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelInfCallsDurationType {
   uint32_t lastCall;
   uint32_t receivedCalls;
   uint32_t dialedCalls;
} TelInfCallsDurationType, *TelInfCallsDurationPtr
```

Fields `lastCall`
Number of seconds used for the last call.

`receivedCalls`
Number of seconds used for all received calls since the call duration timer was reset.

`dialedCalls`
: Number of seconds used for all outgoing calls since the call duration timer was reset.

Comments Used by the `TelInfGetCallsDuration()` function.

TelInfCallsListType Struct

Purpose Holds a list of calls.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelInfCallsListType {
   TelInfCallPtr listP;
   size_t count;
   uint8_t type;
   uint8_t padding[3];
} TelInfCallsListType, *TelInfCallsListPtr
```

Fields

`listP`
: Array of `TelInfCallType` structures that hold call information.

`count`
: Number of elements in the `listP` array.

`type`
: One of the constants described in "Call Types" on page 88, which indicates the type of calls returned in the list.

`padding`
: Padding bytes.

Comments Used by the `TelInfGetCallsList()` function to return call information.

TelInfCallType Struct

Purpose Holds information about a call.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelInfCallType {
   char *fullNameP;
   TelNumberType dialNumber;
   size_t fullNameSize;
   struct tm dateTime;
} TelInfCallType, *TelInfCallPtr
```

Fields

`fullNameP`
: Pointer to a string holding the name associated with the number.

`dialNumber`
: A `TelNumberType` structure that holds information about a telephone number.

`fullNameSize`
: Size of the `fullNameP` string, including the null terminator character.

`dateTime`
: The `tm` structure (defined in `..\posix\time.h`) holds the date and time of the call.

Comments Used in the `TelInfCallsListType` structure.

TelInfIdentificationType Struct

Purpose Holds typed phone information.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelInfIdentificationType {
   char *valueP;
   size_t size;
   uint8_t type;
   uint8_t padding[3];
} TelInfIdentificationType,
*TelInfIdentificationPtr
```

Fields `valueP`
A pointer to a string containing the type of phone information indicated by the `type` field.

`size`
Size of the `valueP` string, including the null terminator character.

`type`
One of the constants described in "Information Types" on page 116, which indicates the type of information returned in `valueP`.

`padding`
Padding bytes.

Comments Used by the `TelInfGetIdentification()` and `TelTestPhoneDriver()` functions to return phone information.

TelMuxChanType Struct

Purpose Holds information about a phone MUX channel.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelMuxChanType {
   uint32_t *chanIdP;
   uint8_t type;
   uint8_t pad[3];
} TelMuxChanType, *TelMuxChanPtr
```

Fields `chanIdP`
A pointer to the channel ID.

`type`
The channel type. One of the values described in "Connection Types" on page 102.

`pad`
Padding bytes.

Comments Used by `TelMuxChanAllocate()`.

TelMuxInfoType Struct

Purpose Holds information about a phone MUX.

Declared In `TelephonyLibTypes.h`

Prototype
```
typedef struct _TelMuxInfoType {
   uint32_t type;
   uint32_t creator;
   uint32_t nameSize;
   uint8_t *nameP;
} TelMuxInfoType, *TelMuxInfoPtr
```

Fields

`type`
The database type.

`creator`
The database creator ID.

`nameSize`
The size of `nameP` in bytes.

`nameP`
A pointer to the MUX device's name.

Comments This structure is used by the phone driver.

TelNotificationType Struct

Purpose Holds information for Telephony Manager notifications.

Declared In `TelephonyLibTypes.h`

Prototype
```
typedef struct _TelNotificationType {
   uint32_t data;
   uint32_t data2;
   uint32_t timeStamp;
   uint16_t id;
   uint8_t priority;
   uint8_t padding;
} TelNotificationType, *TelNotificationPtr
```

Fields

`data`
Various notification-specific data.

`data2`
Various notification-specific data.

`timeStamp`
: Time the event occurred, expressed as the number of seconds elapsed since 12:00 A.M. on January 1, 1904.

`id`
: Identifies the type of event that occurred. One of the constants described in "Notification Identifiers" on page 119.

`priority`
: One of the constants described in "Notification Priorities" on page 123.

`padding`
: Padding bytes.

Comments This structure is passed for the value of the `notifyDetailsP` field in the notification parameter block of a `kTelTelephonyNotification` notification.

TelNumberType Struct

Purpose Holds a phone number.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelNumberType {
   char *numberP;
   size_t size;
   uint16_t type;
   uint16_t padding;
} TelNumberType, *TelNumberPtr
```

Fields

`numberP`
: Buffer containing the phone number.

`size`
: Size of the buffer `numberP`.

`type`
: One of the constants described in "Number Types" on page 123.

`padding`
Padding bytes.

See Also `TelCfgCallForwardingType`, `TelCfgPhoneNumberType`, `TelInfCallType`, `TelPhbEntryType`, `TelSmsMessageType`, `TelSpcCallType`, `TelCfgGetSmsCenter()`, `TelCfgGetVoiceMailNumber()`, `TelCfgSetSmsCenter()`, `TelCfgSetVoiceMailNumber()`

TelNwkLocationType Struct

Purpose Holds network location information.

Declared In `TelephonyLibTypes.h`

Prototype
```
typedef struct _TelNwkLocationType {
   char *areaCodeP;
   size_t areaCodeSize;
   char *cellIdP;
   size_t cellIdSize;
} TelNwkLocationType, *TelNwkLocationPtr
```

Fields

`areaCodeP`
Buffer containing the phone area code.

`areaCodeSize`
Size of the buffer `areaCodeP`.

`cellIdP`
Buffer containing a value that identifies the cell area that the phone is in.

`cellIdSize`
Size of the buffer `cellIdP`.

Comments Used by the `TelNwkGetLocation()` function.

TelNwkOperatorsType Struct

Purpose Holds a list of network operators.

Declared In `TelephonyLibTypes.h`

Prototype
```
typedef struct _TelNwkOperatorsType {
   TelNwkOperatorPtr listP;
   size_t count;
} TelNwkOperatorsType, *TelNwkOperatorsPtr
```

Fields

`listP`
: Array of `TelNwkOperatorType` structures that hold network operator information.

`count`
: Number of elements in the `listP` array.

Comments Used by the `TelNwkGetOperators()` function.

TelNwkOperatorType Struct

Purpose Holds information about a network operator.

Declared In `TelephonyLibTypes.h`

Prototype
```
typedef struct _TelNwkOperatorType {
   uint32_t id;
   char *nameP;
   size_t nameSize;
   uint8_t type;
   uint8_t status;
   uint8_t padding[2];
} TelNwkOperatorType, *TelNwkOperatorPtr
```

Fields

`id`
: Network operator identifier.

`nameP`
: Buffer containing the network operator name.

`nameSize`
: Size of the buffer `nameP`.

`type`
: One of the constants described in "Network Operator Types" on page 118.

`status`
: One of the constants described in "Network Operator Status Constants" on page 118.

`padding`
: Padding bytes.

Comments Used by the `TelNwkGetOperator()` function and in the `TelNwkOperatorsType` structure.

TelNwkPreferredOperatorsType Struct

Purpose Holds a list of preferred network operators.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelNwkPreferredOperatorsType {
   TelNwkPreferredOperatorPtr listP;
   size_t count;
} TelNwkPreferredOperatorsType,
*TelNwkPreferredOperatorsPtr
```

Fields

`listP`
: Array of `TelNwkPreferredOperatorType` structures that hold preferred network operator information.

`count`
: Number of elements in the `listP` array.

Comments Used by the `TelNwkGetPreferredOperators()` function.

TelNwkPreferredOperatorType Struct

Purpose Holds information about a preferred network operator.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelNwkPreferredOperatorType {
   uint32_t id;
   char *nameP;
   size_t nameSize;
   uint16_t index;
   uint16_t padding;
} TelNwkPreferredOperatorType,
*TelNwkPreferredOperatorPtr
```

Fields
`id`
: Network operator identifier.

`nameP`
: Buffer containing the network operator name.

`nameSize`
: Size of the buffer `nameP`.

`index`
: Index of this operator in the preferred operators list (`TelNwkPreferredOperatorsType`).

`padding`
: Padding bytes.

Comments Used in the `TelNwkPreferredOperatorsType` structure.

TelNwkRegistrationType Struct

Purpose Holds network registration information.

Declared In `TelephonyLibTypes.h`

Prototype
```
typedef struct _TelNwkRegistrationType {
   uint32_t operatorId;
   uint8_t mode;
   uint8_t padding[3];
} TelNwkRegistrationType, *TelNwkRegistrationPtr
```

Fields
`operatorId`
: ID of the network operator to register.

`mode`
: One of the constants described in "Registration Search Modes" on page 124.

`padding`
: Padding bytes.

Comments Used by the `TelNwkSetRegistration()` function.

TelNwkUssdType Struct

Purpose Holds Unstructured Supplementary Service Data (USSD).

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelNwkUssdType {
   char *bufferP;
   size_t bufferSize;
   uint8_t result;
   uint8_t dataCodingScheme;
   uint8_t padding[2];
} TelNwkUssdType, *TelNwkUssdPtr
```

Fields

`bufferP`
: Buffer containing the USSD data.

`bufferSize`
: Size of the buffer `bufferP`.

`result`
: One of the constants described in "USSD Result Codes" on page 142. This field is used only when receiving USSD messages, not when sending them.

`dataCodingScheme`
: A data coding scheme as defined in chapter 5 in ETSI (European Telecommunications Standards Institute) TS 100 900 V7.2.0 (GSM 03.38 version 7.2.0 Release 1998). You can retrieve this technical specification document at: http://webapp.etsi.org/workprogram/Report_WorkItem.asp?WKI_ID=6821

`padding`
: Padding bytes.

Comments Used by the `TelNwkCheckUssd()`, `TelNwkReceiveUssd()`, and `TelNwkSendUssd()` functions.

TelOemCallType Struct

Purpose Identifies an OEM function type and associated information.

Declared In `TelephonyLibTypes.h`

Prototype
```
typedef struct _TelOemCallType {
   uint32_t oemId;
   void *paramP;
   size_t paramSize;
   uint8_t funcId;
   uint8_t padding[3];
} TelOemCallType, *TelOemCallPtr
```

Fields

`oemId`
  Unique identifier of the OEM extended function set.

`paramP`
  Pointer to a parameter block that is passed to the function identified by `funcId`.

`paramSize`
  Size of the parameter block `paramP`.

`funcId`
  Identifier of the function within the OEM extended function set.

`padding`
  Padding bytes.

Comments Used by the `TelOemCall()` function.

TelPhbEntriesType Struct

Purpose Holds a list of phone book entries.

Declared In `TelephonyLibTypes.h`

Prototype
```
typedef struct _TelPhbEntriesType {
   TelPhbEntryPtr entryP;
   size_t entryCount;
   uint16_t firstIndex;
   uint16_t lastIndex;
} TelPhbEntriesType, *TelPhbEntriesPtr
```

Fields

`entryP`
: Array of `TelPhbEntryType` structures that hold phone book entries.

`entryCount`
: Number of elements in the `entryP` array.

`firstIndex`
: Index of the first entry to return from the current phone book.

`lastIndex`
: Index of the last entry to return from the current phone book.

Comments Used by the `TelPhbGetEntries()` function. On input, the `firstIndex` and `lastIndex` fields specify the range of entries to return from the current phone book.

TelPhbEntryType Struct

Purpose Holds a phone book entry.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelPhbEntryType {
   char *fullNameP;
   size_t fullNameSize;
   TelNumberType dialNumber;
   uint16_t phoneIndex;
   uint16_t padding;
} TelPhbEntryType, *TelPhbEntryPtr
```

Fields

`fullNameP`
: Buffer containing the name of the entry.

`fullNameSize`
: Size of the buffer `fullNameP`.

`dialNumber`
: A `TelNumberType` structure that holds a phone number.

`phoneIndex`
: Index (zero-based) of this entry in the phone book.

`padding`
: Padding bytes.

Comments Used by the `TelPhbAddEntry()` and `TelPhbGetEntry()` functions.

TelPhbPhonebooksType Struct

Purpose Holds a list of phone books.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelPhbPhonebooksType {
   uint16_t *idP;
   size_t count;
} TelPhbPhonebooksType, *TelPhbPhonebooksPtr
```

Fields

`idP`
Array of phone book identifiers, which are the constants described in "Phone Book Identifiers" on page 124.

`count`
Number of elements in the `idP` array.

Comments Used by the `TelPhbGetPhonebooks()` function.

TelPhbPhonebookType Struct

Purpose Holds information about a phone book.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelPhbPhonebookType {
   size_t usedSlot;
   size_t totalSlot;
   size_t fullNameMaxSize;
   size_t dialNumberMaxSize;
   uint16_t id;
   uint16_t firstIndex;
   uint16_t lastIndex;
   uint16_t padding;
} TelPhbPhonebookType, *TelPhbPhonebookPtr
```

Fields

`usedSlot`
Number of phone book slots that are used.

`totalSlot`
Number of total phone book slots.

`fullNameMaxSize`
Maximum size for a full name in this phone book.

`dialNumberMaxSize`
Maximum size for a phone number in this phone book.

`id`
Phone book identifier. One of the constants described in "Phone Book Identifiers" on page 124.

`firstIndex`
First index of the phone book.

`lastIndex`
Last index of the phone book.

`padding`
Padding bytes.

Comments Used by the `TelPhbGetPhonebook()` function.

TelSmsDateTimeType Struct

Purpose Holds a date and time value.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelSmsDateTimeType {
   uint32_t dateTime;
   Boolean absolute;
   uint8_t padding[3];
} TelSmsDateTimeType, *TelSmsDateTimePtr
```

Fields

`dateTime`
Date and time value. If the `absolute` field is `true`, this is expressed as the number of seconds elapsed since 12:00 A.M. on January 1, 1904. If the `absolute` field is `false`, this is expressed as the number of seconds elapsed from the current time.

`absolute`
If `true`, the `dateTime` value is a Palm OS absolute time value, which is the number of seconds since 1/1/1904. If `false`, the `dateTime` value is relative to the current date and time.

`padding`
    Padding bytes.

Comments Used in the `TelSmsDeliverMessageType` structure.

TelSmsDeliverMessageType Struct

Purpose Holds information about a delivered SMS message.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelSmsDeliverMessageType {
   TelSmsDateTimeType timeStamp;
   Boolean otherToReceive;
   Boolean reportDeliveryIndicator;
   uint8_t networkType;
   uint8_t padding;
   union {
      TelSmsGsmDeliverMessageType gsm;
   } networkParams;
} TelSmsDeliverMessageType,
*TelSmsDeliverMessagePtr
```

Fields

`timeStamp`
    A `TelSmsDateTimeType` structure that holds the timestamp of the message.

`otherToReceive`
    `true` if there are more messages waiting to be received from the service center to the mobile device.

`reportDeliveryIndicator`
    If `true`, indicates that the originating user has asked the network to send a delivery report.

`networkType`
    One of the constants described in "Network Operator Types" on page 118. This indicates which field of the `networkParams` union contains the message information. If this value is `kTelNwkTypeGsmGprs`, then the `networkParams` union contains a `TelSmsGsmDeliverMessageType` structure.

`padding`
    Padding byte.

`networkParams`
: Additional information for different message types. Currently only a GSM message type is defined by a `TelSmsGsmDeliverMessageType` structure.

Comments Used in the `TelSmsMessageType` structure.

TelSmsExtensionType Struct

Purpose Holds extension information about a message.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelSmsExtensionType {
   uint8_t type;
   uint8_t padding[3];
   union {
      TelSmsNbsExtensionType nbs;
      TelSmsSpecialIndicationExtensionType ind;
      TelSmsUserExtensionType user;
   } extension;
} TelSmsExtensionType, *TelSmsExtensionPtr
```

Fields

`type`
: One of the constants described in "SMS Extension Types" on page 129.

`padding`
: Padding bytes.

`extension`
: Extension information, which is one of the following structures: `TelSmsNbsExtensionType`, `TelSmsSpecialIndicationExtensionType`, or `TelSmsUserExtensionType`.

Comments Used in the `TelSmsMessageType` structure.

TelSmsGsmDeliverMessageType Struct

Purpose Holds information for delivered GSM messages.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelSmsGsmDeliverMessageType {
   uint8_t protocolId;
   uint8_t messageClass;
   Boolean replyPath;
   uint8_t padding;
} TelSmsGsmDeliverMessageType,
TelSmsGsmDeliverMessagePtr
```

Fields

`protocolId`
: One of the constants described in "SMS Message Transport Protocol Constants" on page 131.

`messageClass`
: One of the constants described in "SMS Message Class Constants" on page 130.

`replyPath`
: A Boolean value that specifies if the reply path procedure is to be used. The reply path procedure causes a reply to the SMS message to be sent through the service center from which the message came, instead of through the service center whose address is stored on the SIM card.

`padding`
: Padding byte.

Comments Used in the `TelSmsDeliverMessageType` structure.

TelSmsGsmSubmitMessageType Struct

Purpose Holds information for submitted GSM messages.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelSmsGsmSubmitMessageType {
   uint8_t protocolId;
   uint8_t messageClass;
   Boolean rejectDuplicateRequest;
   Boolean replyPath;
} TelSmsGsmSubmitMessageType,
*TelSmsGsmSubmitMessagePtr
```

Fields `protocolId`

Gateway information for routing a message to another transport. Some service centers provide a gateway between SMS and other transports such as mail and FAX. Service centers may reject messages with `protocolId` values that are reserved or unsupported. The mobile device does not interpret reserved or unsupported values, but does store them as received. Specify one of the constants described in "SMS Message Transport Protocol Constants" on page 131.

`messageClass`

One of the constants described in "SMS Message Class Constants" on page 130.

`rejectDuplicateRequest`

A Boolean value that specifies if the service center should reject a submit message for a submit message that is still held in the service center when that message has the same identifier and destination address as a previously submitted message from the same originating address. A value of `true` means that duplicate messages are rejected. Note that his feature is not currently supported.

`replyPath`

A Boolean value that specifies if the reply path procedure is to be used. The reply path procedure causes a reply to the SMS message to be sent through the service center from which the message came, instead of through the service center whose address is stored on the SIM card.

Comments Used in the `TelSmsSubmitMessageType` structure.

TelSmsMessagesType Struct

Purpose List of SMS messages.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelSmsMessagesType {
   TelSmsMessagePtr listP;
   size_t count;
} TelSmsMessagesType, *TelSmsMessagesPtr
```

Fields `listP`

Array of `TelSmsMessageType` structures that hold messages.

`count`

Number of elements in the `listP` array.

Comments Used by the `TelSmsReadMessages()` function.

TelSmsMessageType Struct

Purpose Holds an SMS message.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelSmsMessageType {
   uint8_t *dataP;
   uint32_t messageId;
   size_t dataSize;
   TelNumberType address1;
   TelNumberType address2;
   TelSmsMultiPartInfoType multiPartInfo;
   TelSmsExtensionPtr extensionP;
   size_t extensionCount;
   uint16_t apiVersion;
   uint16_t phoneIndex;
   uint8_t dataCodingScheme;
   uint8_t messageType;
   uint8_t status;
   uint8_t padding;
   union {
      TelSmsSubmitMessageType submit;
      TelSmsDeliverMessageType deliver;
      TelSmsReportMessageType report;
   } message;
} TelSmsMessageType, *TelSmsMessagePtr
```

Fields `dataP`

Buffer containing the message data.

`messageId`

Message identifier.

`dataSize`

Size of the buffer `dataP`.

`address1`
: `TelNumberType` structure that holds the destination address for a submitted message; originating address for delivered and report messages.

`address2`
: `TelNumberType` structure that holds the service center for submitted and delivered GSM messages; callback number for submitted and delivered CDMA and TDMA messages.

`multiPartInfo`
: `TelSmsMultiPartInfoType` structure that holds information about a multipart message.

`extensionP`
: A pointer to an array of `TelSmsExtensionType` structures that you have allocated for this message. You must allocate this array before using this structure.

`extensionCount`
: On input, this is the number of extension structures allocated for this message. You only need to allocate one structure to specify multipart message information, so generally this should be set to 1 for a multipart message.

  Upon return, this is the number of extensions in the SMS header. If the SMS header contains more extensions than you have allocated, the available extension structures are filled, and this function generates a `telErrBufferSize` error.

`apiVersion`
: Version of the SMS API associated with this message.

`phoneIndex`
: Upon return, the SMS index (0-based) of the message on the phone.

  This value is used for input only when calling the `TelSmsReadMessage()` function to read one message at a time, or when calling the `TelSmsDeleteMessage()` function to delete a message.

`dataCodingScheme`
: One of the constants described in "SMS Data Encoding Schemes" on page 127.

`messageType`
: One of the constants described in "SMS Message Types" on page 131.

`status`
: One of the constants described in "SMS Message Status Constants" on page 130.

`padding`
: Padding byte.

`message`
: Message information, which is one of the following structures: `TelSmsSubmitMessageType`, `TelSmsDeliverMessageType`, or `TelSmsReportMessageType`.

Comments Used by the `TelSmsReadMessage()` and `TelSmsSendMessage()` functions.

TelSmsMultiPartInfoType Struct

Purpose Holds information about a multipart message.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelSmsMultiPartInfoType {
   uint16_t bytesSent;
   uint16_t current;
   uint16_t count;
   uint16_t id;
} TelSmsMultiPartInfoType,
*TelSmsMultiPartInfoPtr
```

Fields

`bytesSent`
: On input, set this value to 0.

  Upon return, this is the current count of message bytes that have been sent.

`current`
: On input, set this value to 0.

  Upon return, this is the part number of the current message part.

`count`
: On input, set this value to 0.

  Upon return, this is the number of message parts required to send the data.

`id`
: The ID of the current SMS message. This ID is unique and is the same for all parts of the message. This information is required to reassemble a multi-part SMS.

  On input, set this value to 0.

Comments Used in the `TelSmsMessageType` structure.

TelSmsNbsExtensionType Struct

Purpose Holds information about a NBS message.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelSmsNbsExtensionType {
   uint16_t dest;
   uint16_t source;
} TelSmsNbsExtensionType, *TelSmsNbsExtensionPtr
```

Fields

`dest`
: When the structure is used for input, this is the NBS port number used to encode the data.

  Upon return, this is the NBS port number that was used for the data.

`source`
: The NBS source port number that specifies the content type. Often this is the same as the destination port, but not necessarily so.

Comments Used in the `TelSmsExtensionType` structure.

TelSmsReportMessageType Struct

Purpose Holds information about a report message.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelSmsReportMessageType {
   TelSmsDateTimeType timeStamp;
   uint8_t reportType;
   uint8_t report;
   uint8_t padding[2];
} TelSmsReportMessageType,
*TelSmsReportMessagePtr
```

Fields

`timeStamp`
: A `TelSmsDateTimeType` structure that holds the timestamp of the message.

`reportType`
: One of the constants described in "SMS Report Types" on page 132.

`report`
: One of the constants described in "SMS Delivery Status Reports" on page 128. This is the report.

`padding`
: Padding bytes.

Comments Used in the `TelSmsMessageType` structure.

TelSmsSpecialIndicationExtensionType Struct

Purpose Holds information about waiting messages.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct
_TelSmsSpecialIndicationExtensionType {
   uint8_t type;
   Boolean active;
   Boolean msgStore;
   uint8_t msgWaitingCount;
} TelSmsSpecialIndicationExtensionType,
*TelSmsSpecialIndicationExtensionPtr
```

Fields `type`
: One of the constants described in "SMS Special Indication Types" on page 132.

`active`
: `true` if the indication is active; otherwise `false`.

`msgStore`
: true if the message is to be stored; otherwise `false`.

`msgWaitingCount`
: Number of messages of the type specified (if known), otherwise zero.

Comments Used in the `TelSmsExtensionType` structure.

TelSmsStoragesType Struct

Purpose Holds a list of stores available.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelSmsStoragesType {
   uint16_t *idP;
   size_t count;
} TelSmsStoragesType, *TelSmsStoragesPtr
```

Fields `idP`
: Pointer to an array of store identifiers. Each element is one of the constants described in "SMS Storage Locations" on page 133.

`count`
: Number of elements in the `idP` array.

Comments Used by the `TelSmsGetStorages()` function.

TelSmsStorageType Struct

Purpose Holds information about a store.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelSmsStorageType {
   size_t usedSlot;
   size_t totalSlot;
   uint16_t id;
   uint16_t padding;
} TelSmsStorageType, *TelSmsStoragePtr
```

Fields

`usedSlot`
Number of store slots that are used.

`totalSlot`
Number of total store slots.

`id`
One of the constants described in "SMS Storage Locations" on page 133.

`padding`
Padding bytes.

Comments Used by the `TelSmsGetStorage()` function.

TelSmsSubmitMessageType Struct

Purpose Holds information about a submitted message.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelSmsSubmitMessageType {
   TelSmsDateTimeType validityPeriod;
   Boolean networkDeliveryRequest;
   uint8_t networkType;
   uint8_t padding[2];
   union {
      TelSmsGsmSubmitMessageType gsm;
   } networkParams;
} TelSmsSubmitMessageType,
*TelSmsSubmitMessagePtr
```

Fields

`validityPeriod`
: A `TelSmsDateTimeType` structure that specifies the amount of time for which the message is valid.

`networkDeliveryRequest`
: `true` if a message delivery report is requested from the service center.

`networkType`
: One of the constants described in "Network Operator Types" on page 118. This indicates which field of the `networkParams` union contains the message information. If this value is `kTelNwkTypeGsmGprs`, then the `networkParams` union contains a `TelSmsGsmSubmitMessageType` structure.

`padding`
: Padding bytes.

`networkParams`
: Additional information for different message types. Currently only a GSM message type is defined by a `TelSmsGsmSubmitMessageType` structure.

Comments Used in the `TelSmsMessageType` structure.

TelSmsUserExtensionType Struct

Purpose Holds information about a user-defined extended message header.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelSmsUserExtensionType {
   uint8_t *headerP;
   size_t headerSize;
} TelSmsUserExtensionType,
*TelSmsUserExtensionPtr
```

Fields

`headerP`
: On input, this field must be set to `NULL`. Upon return, this is a pointer to the user-defined header content.

`headerSize`
: Size of the buffer `headerP`. On input, this field must be set to 0.

Comments Used in the `TelSmsExtensionType` structure.

TelSpcCallsType Struct

Purpose Holds a list of current calls.

Declared In `TelephonyLibTypes.h`

Prototype
```
typedef struct _TelSpcCallsType {
   TelSpcCallPtr listP;
   size_t count;
} TelSpcCallsType, *TelSpcCallsPtr
```

Fields

`listP`
: Array of `TelSpcCallType` structures that hold call information.

`count`
: Number of elements in the `listP` array.

Comments Used by the `TelSpcGetCalls()` function.

TelSpcCallType Struct

Purpose Holds information about a call.

Declared In `TelephonyLibTypes.h`

Prototype
```
typedef struct _TelSpcCallType {
   char *dialNameP;
   size_t dialNameSize;
   TelNumberType dialNumber;
   Boolean multiparty;
   uint8_t callId;
   uint8_t direction;
   uint8_t status;
   uint8_t mode;
   uint8_t padding[3];
} TelSpcCallType, *TelSpcCallPtr
```

Fields

`dialNameP`
: Buffer containing the name associated with the call.

`dialNameSize`
: Size of the buffer `dialNameP`.

`dialNumber`
: A `TelNumberType` structure that holds a phone number.

`multiparty`
: `true` for a multiparty call; otherwise, `false`.

`callId`
: Call identifier.

`direction`
: One of the constants described in "Call Direction Constants" on page 86.

`status`
: One of the constants described in "Call Statuses" on page 87.

`mode`
: One of the constants described in "Call Modes" on page 86.

`padding`
: Padding bytes.

Comments Used by the `TelSpcAcceptCall()` and `TelSpcGetCall()` functions.

TelSpcToneDurationRangeType Struct

Purpose Holds the tone duration range.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelSpcToneDurationRangeType {
   uint16_t min;
   uint16_t max;
} TelSpcToneDurationRangeType,
*TelSpcToneDurationRangePtr
```

Fields

`min`
: Minimum tone duration in tens of milliseconds.

`max`
: Maximum tone duration in tens of milliseconds.

Comments Used by the `TelSpcGetToneDurationRange()` function.

TelStyAuthenticationType Struct

Purpose Holds authentication information.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelStyAuthenticationType {
   char *passwordP;
   size_t passwordSize;
   char *newPasswordP;
   size_t newPasswordSize;
   uint16_t type;
   uint16_t reserved;
} TelStyAuthenticationType,
*TelStyAuthenticationPtr
```

Fields

`passwordP`
Pointer to a string containing the current password.

`passwordSize`
Size of the buffer `passwordP`.

`newPasswordP`
Pointer to a string containing the new password to set.

`newPasswordSize`
Size of the buffer `newPasswordP`.

`type`
One of the constants described in "Authentication Types" on page 84, which indicates the type of authentication the phone is waiting for.

`reserved`
Reserved for internal use.

Comments Used by the `TelStyEnterAuthentication()` function.

TelStyFacilitiesType Struct

Purpose Holds a list of security facilities.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelStyFacilitiesType {
   uint16_t *idP;
   size_t count;
} TelStyFacilitiesType, *TelStyFacilitiesPtr
```

Fields

`idP`
: Pointer to an array of security facility identifiers, which are constants described in "Security Facility Types" on page 125.

`count`
: Number of elements in the `idP` array.

Comments Used by the `TelStyGetFacilities()` function.

TelStyFacilityPasswordType Struct

Purpose Holds authentication information for changing a password.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelStyFacilityPasswordType {
   char *passwordP;
   size_t passwordSize;
   char *newPasswordP;
   size_t newPasswordSize;
   uint16_t type;
   uint16_t padding;
} TelStyFacilityPasswordType,
*TelStyFacilityPasswordPtr
```

Fields

`passwordP`
: Pointer to a string containing the current password.

`passwordSize`
: Size of the buffer `passwordP`.

`newPasswordP`
: Pointer to a string containing the new password to set.

`newPasswordSize`
: Size of the buffer `newPasswordP`.

`type`
: One of the constants described in "Security Facility Types" on page 125, which indicates the type of facility that the password is for.

`padding`
: Padding bytes.

Comments Used by the `TelStyChangeFacilityPassword()` function.

TelStyFacilityType Struct

Purpose Holds security facility information.

Declared In `TelephonyLibTypes.h`

Prototype

```
typedef struct _TelStyFacilityType {
   char *passwordP;
   size_t passwordSize;
   uint16_t type;
   uint8_t status;
   uint8_t classType;
} TelStyFacilityType, *TelStyFacilityPtr
```

Fields

`passwordP`
Pointer to a string containing the current password.

`passwordSize`
Size of the buffer `passwordP`.

`type`
One of the constants described in "Security Facility Types" on page 125.

`status`
One of the constants described in "Security Facility Status Constants" on page 125.

`classType`
Sum of integers representing various classes of information. The following classes are defined:

`1`
Voice (telephony).

`2`
Data (all bearer services).

`4`
Fax.

`8`
Short message service.

`16`
Data circuit, synchronous.

`32`
Data circuit, asynchronous.

`64`
Dedicated packet access.

`128`
Dedicated PAD access.

Comments Used by the `TelStyGetFacility()`, `TelStyLockFacility()`, and `TelStyUnlockFacility()` functions.

Telephony Manager Constants

Alert Sound Modes

Purpose Alert sound modes used in the `TelCfgGetAlertSoundMode()` and `TelCfgSetAlertSoundMode()` functions.

Declared In `TelephonyLib.h`

Constants `#define kTelCfgAlertSoundModeNormal 0`
Alert sound is enabled.

`#define kTelCfgAlertSoundModeSilent 1`
Alert sound is disabled (silent).

Authentication Types

Purpose Authentication types used in the `type` field of the `TelStyAuthenticationType` structure, and in the `TelStyGetAuthenticationStatus()` function.

Declared In `TelephonyLib.h`

Constants `#define kTelStyAuthReady 0`
Phone is not waiting for any password.

`#define kTelStyAuthSimPin 1`
Phone is waiting for the SIM Personal Identification Number (PIN).

`#define kTelStyAuthSimPuk 2`
Phone is waiting for the SIM Personal Unlocking Key (PUK).

`#define kTelStyAuthPhoneToSimPin 3`
Phone is waiting for the phone-to-SIM card password.

`#define kTelStyAuthPhoneToFirstSimPin 4`
Phone is waiting for the phone-to-first-SIM card PIN.

`#define kTelStyAuthPhoneToFirstSimPuk 5`
Phone is waiting for the phone-to-first-SIM card PUK.

`#define kTelStyAuthSimPin2 6`
Phone is waiting for the SIM PIN2.

`#define kTelStyAuthSimPuk2 7`
Phone is waiting for the SIM PUK2.

`#define kTelStyAuthNetworkPin 8`
Phone is waiting for the network personalization PIN.

`#define kTelStyAuthNetworkPuk 9`
Phone is waiting for the network personalization PUK.

`#define kTelStyAuthNetworkSubsetPin 10`
Phone is waiting for the network subset personalization PIN.

`#define kTelStyAuthNetworkSubsetPuk 11`
Phone is waiting for the network subset personalization PUK.

`#define kTelStyAuthProviderPin 12`
Phone is waiting for the service provider personalization PIN.

`#define kTelStyAuthProviderPuk 13`
Phone is waiting for the service provider personalization PUK.

`#define kTelStyAuthCorporatePin 14`
Phone is waiting for the corporate personalization PIN.

`#define kTelStyAuthCorporatePuk 15`
Phone is waiting for the corporate personalization PUK.

`#define kTelStyAuthNoSim 16`
No SIM inserted.

Battery Status Constants

Purpose Battery status constants used in the `TelPowGetBatteryConnectionStatus()` function.

Declared In `TelephonyLib.h`

Constants

`#define kTelPowBatteryPowered 0`
Phone is powered by the battery.

`#define kTelPowBatteryNotPowered 1`
Phone is not powered by the battery, though a battery is connected.

`#define kTelPowNoBattery 2`
Phone has no battery connected.

`#define kTelPowBatteryFault 3`
Power fault detected.

Call Direction Constants

Purpose Call direction types used in the `direction` field of the `TelSpcCallType` structure.

Declared In `TelephonyLib.h`

Constants

`#define kTelSpcDirectionMobileOriginated 0`
Call originated by the mobile phone.

`#define kTelSpcDirectionMobileTerminated 1`
Call terminated (received) by the mobile phone.

Call Modes

Purpose Call states used in the `mode` field of the `TelSpcCallType` structure.

Declared In `TelephonyLib.h`

Constants

`#define kTelSpcModeVoice 0`
Voice mode call.

`#define kTelSpcModeData 1`
Data mode call.

`#define kTelSpcModeFax 2`
Fax mode call.

Call Release Types

Purpose Call release types used in the `TelSpcReleaseCall()` function.

Declared In `TelephonyLib.h`

Constants `#define kTelSpcAllCalls 0xF0`
All calls.

`#define kTelSpcAllActiveCalls 0xF1`
All active calls.

`#define kTelSpcAllHeldCalls 0xF2`
All held calls.

`#define kTelSpcIncomingCall 0xF3`
An incoming call.

`#define kTelSpcDialingCall 0`
A call being dialed.

Call Statuses

Purpose Call statuses used in the `status` field of the `TelSpcCallType` structure.

Declared In `TelephonyLib.h`

Constants `#define kTelSpcStatusActive 0`
Active call.

`#define kTelSpcStatusHeld 1`
Held call.

`#define kTelSpcStatusDialing 2`
Dialing call.

`#define kTelSpcStatusAlerting 3`
Alerting status.

`#define kTelSpcStatusIncoming 4`
Incoming call.

`#define kTelSpcStatusWaiting 5`
Waiting call (an incoming call when there are other active or held calls).

`#define kTelSpcStatusReleased 6`
Released call.

Call Types

Purpose Call types used in the `type` field of the `TelInfCallsListType` structure.

Declared In `TelephonyLib.h`

Constants `#define kTelInfCallTypeMissed 0`
Missed calls.

`#define kTelInfCallTypeReceived 1`
Incoming calls.

`#define kTelInfCallTypeDialed 2`
Outgoing calls.

Caller Id Status

Purpose Caller ID status.

Declared In `TelephonyLib.h`

Constants `#define kTelSpcCallerIdValid 0`
Valid caller ID is available.

Card Additional Miscellaneous Result Codes

Purpose Identify additional result codes for the Card Application Toolkit. When returning some general result codes, additional result codes must also be sent. The `addInfo` field of the `TelCatCmdResponseType` structure can be set to this value.

Declared In `TelephonyLib.h`

Constants `#define kTelCatAddGeNoSpecificCause 0x00`
No specific cause can be given.

Card Additional “Bearer Independent Protocol Error” Result Codes

Purpose Identify additional result codes required when the `kTelCatResBearerIndProtocolError` result code is sent for the Card Application Toolkit. The `addInfo` field of the `TelCatCmdResponseType` structure is set to one of these values.

Declared In `TelephonyLib.h`

Constants

`#define kTelCatAddBiBufSizeUnavailable 0x04`
Requested buffer size not available.

`#define kTelCatAddBiChannelClosed 0x02`
Channel closed.

`#define kTelCatAddBiInvalidChannelId 0x03`
Channel identifier not valid.

`#define kTelCatAddBiNoChannelAvailable 0x01`
No channel available.

`#define kTelCatAddBiSecurityError 0x05`
Security error (unsuccessful authentication).

`#define kTelCatAddBiTransportUnavailable 0x06`
Requested UICC/terminal interface transport level not available.

Card Additional “Interaction with Call Control, Permanent Problem” Result Codes

Purpose Identify additional result codes required when the `kTelCatResSimControlFault` result code is sent for the Card Application Toolkit. The `addInfo` field of the `TelCatCmdResponseType` structure is set to one of these values.

Declared In `TelephonyLib.h`

Constants

`#define kTelCatAddCsActionNotAllowed 0x01`
Action not allowed.

`#define kTelCatAddCsRequestTypeChange 0x02`
The type of request has changed.

Card Additional "Launch Browser" Result Codes

Purpose Identify additional result codes required when the `kTelCatResBrowserGenericError` result code is sent for the Card Application Toolkit. The `addInfo` field of the `TelCatCmdResponseType` structure is set to one of these values.

Declared In `TelephonyLib.h`

Constants

`#define kTelCatAddLbBearerUnavailable 0x01`
Bearer unavailable.

`#define kTelCatAddLbBrowserUnavailable 0x02`
Browser unavailable.

`#define kTelCatAddLbDataReadError 0x03`
Terminal unable to read the provisioning data.

Card Additional "Terminal Unable to Process Command" Result Codes

Purpose Identify additional result codes required when the `kTelCatResMeUnableNow` result code is sent for the Card Application Toolkit. The `addInfo` field of the `TelCatCmdResponseType` structure is set to one of these values.

Declared In `TelephonyLib.h`

Constants

`#define kTelCatAddUnAccessControlBar 0x05`
Access control class bar.

`#define kTelCatAddUnMeBusyOnCall 0x02`
Terminal currently busy on call.

`#define kTelCatAddUnMeBusyOnSendDtmf 0x09`
ME currently busy on SEND DTMF command.

`#define kTelCatAddUnMeBusyOnSuppSvc 0x03`
Reserved for GSM/3G.

`#define kTelCatAddUnMeBusyOnUssd 0x08`
Reserved for GSM/3G.

`#define kTelCatAddUnNoRadioResource 0x06`
Radio resource not granted.

`#define kTelCatAddUnNoService 0x04`
No service.

`#define kTelCatAddUnNotInSpeechCall 0x07`
Not in speech call.

`#define kTelCatAddUnScreenBusy 0x01`
Screen is busy.

Card Browser Termination Cause Codes

Purpose Identify the causes of a card browser termination for the Card Application Toolkit. The `browserTerminationCause` field of the `TelCatEventToCardType` structure is set to these values.

Declared In `TelephonyLib.h`

Constants `#define kTelCatBrowserTerminationError 0x01`
Terminated because of an error.

`#define kTelCatBrowserTerminationUser 0x00`
The user terminated the browser.

Card Call Set Up Actions

Purpose Identify whether the user accepted or rejected the incoming call for the Card Application Toolkit. The *`iAction`* parameter of the `TelCatCallAction()` function is set to these values.

Declared In `TelephonyLib.h`

Constants `#define kTelCatCallAccept 1`
The user accepted the call.

`#define kTelCatCallReject 0`
The user rejected the call.

Card Command IDs

Purpose Identify the command IDs for the Card Application Toolkit. The `cmdId` field of the `TelCatCmdParamsType` structure is set to one of these values.

Declared In `TelephonyLib.h`

Constants

`#define kTelCatCmdCloseChannel 0x41`
Close the channel.

`#define kTelCatCmdDisplayText 0x21`
Display text.

`#define kTelCatCmdGetInkey 0x22`
Get in key.

`#define kTelCatCmdGetInput 0x23`
Get input.

`#define kTelCatCmdLaunchBrowser 0x15`
Launch browser.

`#define kTelCatCmdOpenChannel 0x40`
Open the channel.

`#define kTelCatCmdPlayTone 0x20`
Play tone.

`#define kTelCatCmdReceiveData 0x42`
Receive data.

`#define kTelCatCmdRefresh 0x01`
Refresh.

`#define kTelCatCmdRunATCommand 0x34`
Run AT command.

`#define kTelCatCmdSelectItem 0x24`
Select item.

`#define kTelCatCmdSendData 0x43`
Send data.

`#define kTelCatCmdSendDTMF 0x14`
Send DTMF.

`#define kTelCatCmdSendShortMessage 0x13`
Send short message.

`#define kTelCatCmdSendSS 0x11`
Send SS.

`#define kTelCatCmdSendUSSD 0x12`
Send USSD.

`#define kTelCatCmdSetUpCall 0x10`
Set up call.

`#define kTelCatCmdSetUpEventList 0x05`
Set up event list.

`#define kTelCatCmdSetUpIdleModeText 0x28`
Set up idle mode text.

`#define kTelCatCmdSetUpMenu 0x25`
Set up menu.

`#define kTelCatEndOfProactiveSession 0x81`
A special command ID that indicates the end of a proactive command session.

Card Command Termination Reasons

Purpose Identify the reason for terminating a card command or session for the Card Application Toolkit. The *`iReason`* parameter of the `TelCatTerminate()` function is set to these values.

Declared In `TelephonyLib.h`

Constants `#define kTelCatTerminateEndOfRedialingReached 1`
End of redialing reached.

`#define kTelCatTerminateUserEndsSession 2`
The user ended the session.

`#define kTelCatTerminateUserStoppedRedialing 0`
The user stopped redialing.

Card Elementary File Access Modes

Purpose Identify the elementary file (EF) access modes for the Card Application Toolkit. The `mode` field of the `TelCardFileType` structure is set to one of these values.

Declared In `TelephonyLib.h`

Constants `#define kTelCardModeGetInfo 0`
Get EF information.

`#define kTelCardModeReadFile 1`
Read EF body.

`#define kTelCardModeReadPart 2`
Read EF part.

`#define kTelCardModeReadRec 3`
Read EF record.

Card Elementary File Structures

Purpose Identify the elementary file (EF) structure for the Card Application Toolkit. The `fileStruct` field of the `TelCardFileType` structure is set to one of these values.

Declared In `TelephonyLib.h`

Constants `#define kTelCardFileStructCyclic 0x03`
Cyclic.

`#define kTelCardFileStructLinearFixed 0x01`
Linear fixed.

`#define kTelCardFileStructTransparent 0x00`
Transparent.

Card General Result Codes

Purpose Identify general result codes of commands for the Card Application Toolkit. The `resCode` field of the `TelCatCmdResponseType` structure is set to these values.

Declared In `TelephonyLib.h`

Constants

`#define kTelCatResBackwardMove 0x11`
Backward move in the proactive UICC session requested by the user.

`#define kTelCatResBearerIndProtocolError 0x3A`
Bearer Independent Protocol error.

`#define kTelCatResBeyondMeCapabilities 0x30`
Command beyond terminal's capabilities.

`#define kTelCatResBrowserGenericError 0x26`
Launch browser generic error code.

`#define kTelCatResCallClearedByUser 0x23`
User cleared down call before connection or network release.

`#define kTelCatResCmdDataNotUnderstood 0x32`
Command data not understood by terminal.

`#define kTelCatResCmdTypeNotUnderstood 0x31`
Command type not understood by terminal.

`#define kTelCatResHelpInfoRequest 0x13`
Help information required by the user.

`#define kTelCatResMeUnableNow 0x20`
Terminal currently unable to process command.

`#define kTelCatResMissingValues 0x36`
Error, required values are missing.

`#define kTelCatResMultipleCardError 0x38`
MultipleCard commands error.

`#define kTelCatResNetworkUnableNow 0x21`
Network currently unable to process command.

`#define kTelCatResNoResponseFromUser 0x12`
No response from user.

`#define kTelCatResOkAdditionalEfsRead 0x03`
Refresh performed with additional EFs read.

`#define kTelCatResOkIconNotDisplayed 0x04`
Command performed successfully, but requested icon could not be displayed.

`#define kTelCatResOkLimitedService 0x06`
Command performed successfully, limited service.

`#define kTelCatResOkMissingInfo 0x02`
Command performed, with missing information.

`#define kTelCatResOkModifiedBySim 0x05`
Command performed, but modified by call control by NAA.

`#define kTelCatResOkPartialComprehension 0x01`
Command performed with partial comprehension.

`#define kTelCatResOkWithModification 0x07`
Command performed with modification.

`#define kTelCatResSimControlFault 0x39`
Interaction with call control by NAA, permanent problem.

`#define kTelCatResSimControlInteraction 0x25`
Interaction with call control by NAA, temporary problem.

`#define kTelCatResSmsRpError 0x35`
SMS RPERROR in an SMS send command.

`#define kTelCatResSuccess 0x00`
Command performed successfully.

`#define kTelCatResSuppSvcReturnError 0x34`
Supplemental Services (SS) Return Error in a Setup call or Send SS command.

`#define kTelCatResTimerContradiction 0x24`
Action in contradiction with the current timer state.

`#define kTelCatResTransactionTermination 0x14`
USSD/SS Transaction terminated by user in Setup call, Send SS, or Send USSD command.

`#define kTelCatResUnknownCmdNumber 0x33`
Command number not known by terminal;

`#define kTelCatResUserDismissal 0x22`
User did not accept the proactive command.

`#define kTelCatResUserTermination 0x10`
Proactive UICC session terminated by the user.

`#define kTelCatResUssdReturnError 0x37`
USSD Return error in a Send USSD command.

Card Get Inkey and Get Input Command Response Types

Purpose Identify the expected response types for the Card Application Toolkit's Get Inkey and Get Input commands. The `respType` field of the `TelCatCmdResponseType`, `TelCatGetInkeyType`, and `TelCatGetInputType` structures are set to these values.

Declared In `TelephonyLib.h`

Constants `#define kTelCatRespTypeDigitsGSM 0x02`
Applies to Get Inkey, Get Input.

`#define kTelCatRespTypeDigitsGSMPacked 0x03`
Applies to Get Input.

`#define kTelCatRespTypeDigitsUCS2 0x04`
Applies to Get Inkey, Get Input.

`#define kTelCatRespTypeTextGSM 0x05`
Applies to Get Inkey, Get Input.

`#define kTelCatRespTypeTextGSMPacked 0x06`
Applies to Get Input.

`#define kTelCatRespTypeTextUCS2 0x07`
Applies to Get Inkey, Get Input.

`#define kTelCatRespTypeYesOrNo 0x01`
Applies to Get Inkey.

Card Launch Browser Command Bearer Codes

Purpose Identify the bearer codes for the Card Application Toolkit's Launch Browser command. The `prefBearersP` field of the `TelCatLaunchBrowserType` structure points to a list of these values.

Declared In `TelephonyLib.h`

Constants `#define kTelCatBearerCSD 0x1`
Applies to Launch Browser, open channel.

`#define kTelCatBearerGPRS 0x2`
Applies to Launch Browser, open channel.

`#define kTelCatBearerSMS 0x3`
Applies to Launch Browser.

`#define kTelCatBearerUSSD 0x4`
Applies to Launch Browser.

Card Launch Browser Command Conditions

Purpose Identify the conditions for the Card Application Toolkit's Launch Browser command. The `condition` field of the `TelCatLaunchBrowserType` structure is set to these values.

Declared In `TelephonyLib.h`

Constants `#define kTelCatBrowserCloseExistingLaunchNew 0x03`
Close existing browser and launch a new one.

`#define kTelCatBrowserLaunchIfNotLaunched 0x00`
Launch browser if it is not already launched.

`#define kTelCatBrowserUseExisting 0x02`
Use the existing browser.

Card Menu Selection Event Codes

Purpose Identify the menu selection event codes for the Card Application Toolkit. The `evtCode` field of the `TelCatMenuSelectionType` structure is set to these values.

Declared In `TelephonyLib.h`

Constants `#define kTelCatMenuSelAppLaunch 0x01`
Application launch.

`#define kTelCatMenuSelAppMenuRequest 0x03`
Application menu request.

`#define kTelCatMenuSelHelpInfoRequest 0x02`
Help information request.

Card Open Channel Command Address and Transport Types

Purpose Identify the address and transport types for the Card Application Toolkit's Open Channel command. The `otherAddressType` and `destinationAddressType` fields or the `transportType` field of the `TelCatOpenChanType` structure the are set to these values.

Declared In `TelephonyLib.h`

Constants

`#define kTelCatAddressIPv4 0x21`
An IPv4 address.

`#define kTelCatAddressIPv6 0x97`
An IPv6 address.

`#define kTelCatTransportTCP 0x02`
TCP transport type.

`#define kTelCatTransportUDP 0x01`
UDP transport type.

Card Play Tone Command Sound Codes

Purpose Identify the sound codes for the Card Application Toolkit's Play Tone command. The `sndCode` field of the `TelCatPlayToneType` structure is set to these values.

Declared In `TelephonyLib.h`

Constants

`#define kTelCatSoundError 0x12`
Negative acknowledgment or error tone.

`#define kTelCatSoundGeneralBeep 0x10`
General beep.

`#define kTelCatSoundPositiveAck 0x11`
Positive acknowledgment tone.

`#define kTelCatSoundStdCallDropped 0x05`
(Standard) radio path not available, call dropped.

`#define kTelCatSoundStdCalledPartyBusy 0x02`
(Standard) called party is busy.

`#define kTelCatSoundStdCallWaiting 0x07`
(Standard) call waiting tone.

`#define kTelCatSoundStdCongestion 0x03`
(Standard) congestion.

`#define kTelCatSoundStdDial 0x01`
(Standard) dial tone.

`#define kTelCatSoundStdError 0x06`
(Standard) error or special information.

`#define kTelCatSoundStdRadioPathAck 0x04`
(Standard) radio path acknowledgment.

`#define kTelCatSoundStdRing 0x08`
(Standard) ringing tone.

Card Refresh Command Opcodes

Purpose Identify the opcode values for the Card Application Toolkit's Refresh command. These specify the refresh mode this command uses. The `opCode` field of the `TelCatRefreshType` structure is set to one of these values.

Declared In `TelephonyLib.h`

Constants `#define kTelCatRefreshFileChange 0x01`
File change notification.

`#define kTelCatRefreshHardReset 0x04`
Hard reset.

`#define kTelCatRefreshInitAndFileChange 0x02`
Initialization and file change notification.

`#define kTelCatRefreshInitAndFullFileChange 0x00`
Initialization and full file change notification.

`#define kTelCatRefreshInitialization 0x03`
Initialization.

Card Set Up Call Command Call Conditions

Purpose Identify the conditions for setting up a call for the Card Application Toolkit's Set Up Call command. The `condition` field of the `TelCatSetUpCallType` structure is set to one of these values.

Declared In `TelephonyLib.h`

Constants

`#define kTelCatCallCloseOthers 0x04`
Close other calls.

`#define kTelCatCallCloseOthersRedial 0x05`
Close other calls and redial.

`#define kTelCatCallHoldOthers 0x02`
Hold other calls.

`#define kTelCatCallHoldOthersRedial 0x03`
Hold other calls and redial.

`#define kTelCatCallNotBusy 0x00`
Not busy.

`#define kTelCatCallNotBusyRedial 0x01`
Not busy and redial.

Card Set Up Event List Command Events

Purpose Identify the types of events to be monitored for the Card Application Toolkit's Set Up Event List command. The `eventP` field of the `TelCatSetUpEventListType` structure is set to one of these values.

Declared In `TelephonyLib.h`

Constants

`#define kTelCatEventBrowserTermination 0x08`
Browser termination.

`#define kTelCatEventIdleScreenAvailable 0x05`
Idle screen available.

`#define kTelCatEventLanguageSelection 0x07`
Language selection.

`#define kTelCatEventUserActivity 0x04`
User activity.

Connection Types

Purpose Types of telephony connections. The `type` fields of the `TelMuxChanType` and `TelDtcConnectionInfoType` structures `are` set to one of these values.

Declared In `TelephonyLib.h`

Constants

`#define kTelConnectionTypeBT 4`
Bluetooth.

`#define kTelConnectionTypeCommand 0`
Phone MUX command channel.

`#define kTelConnectionTypeCSD 2`
Circuit-switched data.

`#define kTelConnectionTypeGPRS 3`
GPRS.

`#define kTelConnectionTypeModem 1`
Modem.

`#define kTelConnectionTypeOEM 6`
OEM.

`#define kTelConnectionTypeVC 5`
Not used.

"Connection Types" on page 102Forwarding Classes

Purpose Call forwarding classes used in the `classType` field of the `TelCfgCallForwardingType` structure.

Declared In `TelephonyLib.h`

Constants

`#define kTelCfgForwardingClassVoice 1`
Voice call.

`#define kTelCfgForwardingClassData 2`
Data call.

`#define kTelCfgForwardingClassFax 4`
Fax call.

`#define kTelCfgForwardingClassSms 8`
SMS message.

`#define kTelCfgForwardingClassDataCircuitSync 16`
Synchronous data circuit.

`#define kTelCfgForwardingClassDataCircuitAsync 32`
Asynchronous data circuit.

`#define kTelCfgForwardingClassDedicatedPacketAccess 64`
Dedicated packet access.

`#define kTelCfgForwardingClassDedicatedPADAccess 128`
Dedicated PAD access.

Forwarding Modes

Purpose Call forwarding modes used in the `mode` field of the `TelCfgCallForwardingType` structure.

Declared In `TelephonyLib.h`

Constants `#define kTelCfgForwardingModeDisable 0`
Disable call forwarding.

`#define kTelCfgForwardingModeEnable 1`
Enable call forwarding.

`#define kTelCfgForwardingModeRegistration 3`
Register a call forwarding request on the network.

`#define kTelCfgForwardingModeErasure 4`
Erase a call forwarding request stored on the network.

Forwarding Reasons

Purpose Call forwarding reasons used in the `reason` field of the `TelCfgCallForwardingType` structure.

Declared In `TelephonyLib.h`

Constants `#define kTelCfgForwardingReasonUnconditional 0`
Forward unconditionally.

`#define kTelCfgForwardingReasonMobileBusy 1`
Forward if the mobile phone is busy.

`#define kTelCfgForwardingReasonNoReply 2`
Forward if there is no answer.

`#define kTelCfgForwardingReasonNotReachable 3`
Forward if the number is unreachable.

`#define kTelCfgForwardingReasonAllCallForwarding 4`
All call forwarding reasons.

`#define kTelCfgForwardingReasonAllCondCallForwarding 5`
All conditional call forwarding reasons.

GPRS Attachment State

Purpose Identify whether the mobile terminal is attached to or detached from the GPRS service. The *`iAttach`* parameter of the `TelGprsSetAttach()` function takes one of these values.

Declared In `TelephonyLib.h`

Constants `#define kTelGprsAttached 1`
Attached to the GPRS service.

`#define kTelGprsDetached 0`
Detached from the GPRS service.

GPRS Compression Settings

Purpose Identify whether the header and data for a given PDP context are compressed. The `dataCompression` and `headerCompression` fields of the `TelGprsContextType` structure are set to one of these values.

Declared In `TelephonyLib.h`

Constants `#define kTelGprsDataCompressionSetOff 0`
No data compression.

`#define kTelGprsDataCompressionSetOn 1`
V.42 bis data compression.

`#define kTelGprsHdrCompressionSetOff 0`
No header compression.

`#define kTelGprsHdrCompressionSetOn 1`
: V.42 bis header compression.

GPRS Event Reporting Settings

Purpose Identify GPRS events reported by the mobile equipment (ME) or the GPRS network that can be cause the device to send unsolicited result codes. The `mode` field of the `TelGprsEventReportingType` structure is set to one of these values.

Declared In `TelephonyLib.h`

Constants

`#define kTelGprsEventMeClass 7`
: Mobile Station (MS) Class changed by the mobile equipment (ME).

`#define kTelGprsEventMeDeact 3`
: PDP context activation deactivated by the mobile equipment (ME).

`#define kTelGprsEventMeDetach 5`
: GPRS detached by the mobile equipment (ME).

`#define kTelGprsEventNwClass 6`
: Mobile Station (MS) Class changed by the network.

`#define kTelGprsEventNwDeact 2`
: PDP context activation deactivated by the network.

`#define kTelGprsEventNwDetach 4`
: GPRS detached by the network.

`#define kTelGprsEventNwReact 1`
: PDP context activation reactivated by the network.

`#define kTelGprsEventReject 0`
: PDP context activation rejected.

`#define kTelGprsEventReportingBufferedMode 2`
: Event reporting forwarded if the link is OK or buffered and then forwarded when the link is OK again.

`#define kTelGprsEventReportingClearBuffer 0`
: Mobile equipment (ME) buffer of unsolicited result code is cleared when

`kTelOemGprsEventReportingEnabledMode` or `kTelOemGprsEventReportingBufferedMode` is chosen.

`#define kTelGprsEventReportingDisabledMode 0`
: No event reporting forwarded.

`#define kTelGprsEventReportingEnabledMode 1`
: Event reporting forwarded if the link is OK.

`#define kTelGprsEventReportingFlushBuffer 1`
: Mobile equipment (ME) buffer of unsolicited result code is flushed to the Telephony when `kTelOemGprsEventReportingEnabledMode` or `kTelOemGprsEventReportingBufferedMode` is chosen.

GPRS Layer 2 Protocol

Purpose Identify the layer 2 protocol to use.

Declared In `TelephonyLib.h`

Constants `#define kTelGprsLayer2ProtocolNull 1`
: None. This is used for PDP type OSP:IHOSS.

`#define kTelGprsLayer2ProtocolPPP 0`
: Use PPP for a PDP such as IP.

GPRS Network Registration Settings

Purpose Identify how to present GPRS network registration unsolicited events. The `registrationType` field of the `TelGprsNwkRegistrationType` structure and the *`iRegistrationType`* parameter of `TelGprsSetNwkRegistration()` are set to one of these values.

Declared In `TelephonyLib.h`

Constants `#define kTelGprsNwkRegistrationCellEnable 2`
: Present notifications when the GPRS network or the service cell changes.

`#define kTelGprsNwkRegistrationCellSupportingStatusEnable 3`
Present notifications when any of the following change: the GPRS network registration status, service cell, or GPRS supporting status of service cell.

`#define kTelGprsNwkRegistrationDisable 0`
Diasble notifications when the GPRS network registration status changes.

`#define kTelGprsNwkRegistrationNwkEnable 1`
Present notifications when the GPRS network registration status changes.

GPRS Network Registration Status

Purpose Identify the current GPRS network registration status. The `registrationStatus` field of the `TelGprsNwkRegistrationType` structure is set to one of these values.

Declared In `TelephonyLib.h`

Constants

`#define kTelGprsNwkRegistrationStatusDenied 3`
Registration denied.

`#define kTelGprsNwkRegistrationStatusNotRegistered 0`
Not currently searching for a new operator with which to register.

`#define kTelGprsNwkRegistrationStatusRegistered 1`
Registered on the home GPRS network.

`#define kTelGprsNwkRegistrationStatusRoaming 5`
Registered on a GPRS network while roaming.

`#define kTelGprsNwkRegistrationStatusSearching 2`
Not registered but currently searching for a new operator with which to register.

`#define kTelGprsNwkRegistrationStatusUnknown 4`
Registration status unknown.

GPRS OSPIH Protocol Settings

Purpose Identify the protocol used over IP on OSPIH. The `OSPIHProtocol` field of the `TelGprsContextType` structure is set to one of these values.

Declared In `TelephonyLib.h`

Constants `#define kTelGprsOSPIHProtocolTCP 1`
TCP used over IP on GPRS OSPIH.

`#define kTelGprsOSPIHProtocolUDP 0`
UDP used over IP on GPRS OSPIH.

GPRS Packet Data Protocols

Purpose Identify the GPRS packet data protocol used in a given PDP context. The `pdpType` field of the `TelGprsContextType` structure is set to one of these values.

Declared In `TelephonyLib.h`

Constants `#define kTelGprsPdpIP 0`
Internet Protocol.

`#define kTelGprsPdpOSPIH 2`
Internet Hosted Octet Stream Protocol (IHOSP).

`#define kTelGprsPdpPPP 1`
Point-to-Point Protocol.

`#define kTelGprsValueUnknown 0xFF`
Unknown protocol type value.

GPRS PDP Activation State

Purpose Identify whether a given PDP context is activated. The `state` field of the `TelGprsPdpActivationType` structure is set to one of these values.

Declared In `TelephonyLib.h`

Constants `#define kTelGprsPdpActivated 1`

`#define kTelGprsPdpDeactivated 0`

GPRS Quality of Service

Purpose Identify the quality-of-service level for a PDP context. Several of the fields of the `TelGprsQosType` structure are set to these values.

Declared In `TelephonyLib.h`

Constants

`#define kTelGprsQosDelayBestEffort 4`
Best effort.

`#define kTelGprsQosDelayClass1 1`
<2 seconds for a 1024 SDU size.

`#define kTelGprsQosDelayClass2 2`
<15 seconds for a 1024 SDU size.

`#define kTelGprsQosDelayClass3 3`
<75 seconds for a 1024 SDU size.

`#define kTelGprsQosDelayDefault 0`
Default delay.

`#define kTelGprsQosMeanClass1 1`
100 octets/hour (~0.22 bit/s).

`#define kTelGprsQosMeanClass10 10`
100000 octets/hour (~0.22 kbit/s).

`#define kTelGprsQosMeanClass11 11`
200000 octets/hour (~0.44 kbit/s).

`#define kTelGprsQosMeanClass12 12`
500000 octets/hour (~1.11 kbit/s).

`#define kTelGprsQosMeanClass13 13`
1000000 octets/hour (~2.2 kbit/s).

`#define kTelGprsQosMeanClass14 14`
2000000 octets/hour (~4.4 kbit/s).

`#define kTelGprsQosMeanClass15 15`
5000000 octets/hour (~11.1 kbit/s).

`#define kTelGprsQosMeanClass16 16`
10000000 octets/hour (~22 kbit/s).

`#define kTelGprsQosMeanClass17 17`
20000000 octets/hour (~44 kbit/s).

`#define kTelGprsQosMeanClass18 18`
50000000 octets/hour (~111 kbit/s).

`#define kTelGprsQosMeanClass2 2`
: 200 octets/hour (~0.44 bit/s).

`#define kTelGprsQosMeanClass3 3`
: 500 octets/hour (~1.11 bit/s).

`#define kTelGprsQosMeanClass4 4`
: 1000 octets/hour (~2.2 bit/s).

`#define kTelGprsQosMeanClass5 5`
: 2000 octets/hour (~4.4 bit/s).

`#define kTelGprsQosMeanClass6 6`
: 5000 octets/hour (~11.1 bit/s).

`#define kTelGprsQosMeanClass7 7`
: 10000 octets/hour (~22 bit/s).

`#define kTelGprsQosMeanClass8 8`
: 20000 octets/hour (~44 bit/s).

`#define kTelGprsQosMeanClass9 9`
: 50000 octets/hour (~111 bit/s).

`#define kTelGprsQosMeanClassBestEffort 31`
: Best effort.

`#define kTelGprsQosMeanDefault 0`
: Default mean.

`#define kTelGprsQosPeakClass1 1`
: Up to 1000 octets/s (8 kbit/s).

`#define kTelGprsQosPeakClass2 2`
: Up to 2000 octets/s (16 kbit/s).

`#define kTelGprsQosPeakClass3 3`
: Up to 4000 octets/s (32 kbit/s).

`#define kTelGprsQosPeakClass4 4`
: Up to 8000 octets/s (64 kbit/s).

`#define kTelGprsQosPeakClass5 5`
: Up to 16000 octets/s (128 kbit/s).

`#define kTelGprsQosPeakClass6 6`
: Up to 32000 octets/s (256 kbit/s).

`#define kTelGprsQosPeakClass7 7`
: Up to 64000 octets/s (512 kbit/s).

`#define kTelGprsQosPeakClass8 8`
Up to 128000 octets/s (1024 kbit/s).

`#define kTelGprsQosPeakClass9 9`
Up to 256000 octets/s (2048 kbit/s).

`#define kTelGprsQosPeakDefault 0`
Default peak.

`#define kTelGprsQosPrecedenceDefault 0`
Default precedence.

`#define kTelGprsQosPrecedenceHigh 1`
High precedence.

`#define kTelGprsQosPrecedenceLow 3`
Low precedence.

`#define kTelGprsQosPrecedenceNormal 2`
Normal precedence.

`#define kTelGprsQosReliabilityClass1 1`
GTP mode acknowledged, LLC mode acknowledged, LLC data protected, RLC block acknowledged.

`#define kTelGprsQosReliabilityClass2 2`
GTP mode unacknowledged, LLC mode acknowledged, LLC data protected, RLC block acknowledged.

`#define kTelGprsQosReliabilityClass3 3`
GTP mode unacknowledged, LLC mode unacknowledged, LLC data protected, RLC block acknowledged.

`#define kTelGprsQosReliabilityClass4 4`
GTP mode unacknowledged, LLC mode unacknowledged, LLC data protected, RLC block unacknowledged.

`#define kTelGprsQosReliabilityClass5 5`
GTP mode unacknowledged, LLC mode unacknowledged, LLC data unprotected, RLC block unacknowledged.

`#define kTelGprsQosReliabilityDefault 0`
Default reliability.

GPRS SMS Service Preferences

Purpose Identify the preferred service for transferring SMS messages. The `TelGprsGetSmsService()` and `TelGprsSetSmsService()` functions use these values.

Declared In `TelephonyLib.h`

Constants

`#define kTelGprsSmsGprsOnly 0`
Transfer SMS messages over GPRS only.

`#define kTelGprsSmsGprsPreferred 2`
Transfer SMS messages over GPRS, if available; otherwise use GSM.

`#define kTelGprsSmsGsmOnly 1`
Transfer SMS messages over GSM only.

`#define kTelGprsSmsGsmPreferred 3`
Transfer SMS messages over GSM, if available; otherwise use GPRS.

GSM CSD Bearer Service Connection Element

Purpose Identify the GSM bearer service connection element for circuit-switched data (CSD) calls. The `connection` field of the `TelDtcCsdConnectionType` structure is set to one of these values.

Declared In `TelephonyLib.h`

Constants

`#define kTelDtcBearerConnectionBothNonTransparentPreferred 3`
Both, nontransparent preferred.

`#define kTelDtcBearerConnectionBothTransparentPreferred 2`
Both, transparent preferred.

`#define kTelDtcBearerConnectionNonTransparent 1`
Nontransparent.

`#define kTelDtcBearerConnectionTransparent 0`
Transparent.

GSM CSD Bearer Service Name

Purpose Identify the GSM bearer service name for circuit-switched data (CSD) calls. The `service` field of the `TelDtcCsdConnectionType` structure is set to one of these values.

Declared In `TelephonyLib.h`

Constants

`#define kTelDtcBearerDataAsynchronousRDI 4`
Data circuit asynchronous (RDI).

`#define kTelDtcBearerDataAsynchronousUDI 0`
Data circuit asynchronous (UDI or 3.1-kHz modem).

`#define kTelDtcBearerDataSynchronousRDI 5`
Data circuit synchronous (RDI).

`#define kTelDtcBearerDataSynchronousUDI 1`
Data circuit synchronous (UDI or 3.1-kHz modem).

`#define kTelDtcBearerPacketAccessSynchronousRDI 7`
Packet Access (synchronous) (RDI).

`#define kTelDtcBearerPacketAccessSynchronousUDI 3`
Packet Access (synchronous) (UDI).

`#define kTelDtcBearerPADAccessAsynchronousRDI 6`
PAD Access (asynchronous) (RDI).

`#define kTelDtcBearerPADAccessAsynchronousUDI 2`
PAD Access (asynchronous) (UDI).

GSM CSD Bearer Service Speeds

Purpose Identify the GSM bearer service speed settings used for circuit-switched data (CSD) calls. The `speed` field of the `TelDtcCsdConnectionType` structure is set to one of these values.

Declared In `TelephonyLib.h`

Constants

`#define kTelDtcBearerDataRate1200bpsV110 66`
1200 bps (V.110).

`#define kTelDtcBearerDataRate1200bpsV120 34`
1200 bps (V.120).

`#define kTelDtcBearerDataRate1200bpsV22 2`
: 1200 bps (V.22).

`#define kTelDtcBearerDataRate1200_75bpsV23 3`
: 1200/75 bps (V.23).

`#define kTelDtcBearerDataRate14400bpsV110 75`
: 14400 bps (V.110 or X.31 flag stuffing).

`#define kTelDtcBearerDataRate14400bpsV120 43`
: 14400 bps (V.120).

`#define kTelDtcBearerDataRate14400bpsV34 14`
: 14400 bps (V.34).

`#define kTelDtcBearerDataRate19200bpsV110 79`
: 19200 bps (V.110 or X.31 flag stuffing).

`#define kTelDtcBearerDataRate19200bpsV120 47`
: 19200 bps (V.120).

`#define kTelDtcBearerDataRate19200bpsV34 15`
: 19200 bps (V.34).

`#define kTelDtcBearerDataRate2400bpsV110 68`
: 2400 bps (V.110 or X.31 flag stuffing).

`#define kTelDtcBearerDataRate2400bpsV120 36`
: 2400 bps (V.120).

`#define kTelDtcBearerDataRate2400bpsV22bis 4`
: 2400 bps (V.22 bis).

`#define kTelDtcBearerDataRate2400bpsV26ter 5`
: 2400 bps (V.26 ter).

`#define kTelDtcBearerDataRate28800bpsV110 80`
: 28800 bps (V.110 or X.31 flag stuffing).

`#define kTelDtcBearerDataRate28800bpsV120 48`
: 28800 bps (V.120).

`#define kTelDtcBearerDataRate28800bpsV34 16`
: 28800 bps (V.34).

`#define kTelDtcBearerDataRate300bpsV110 65`
: 300 bps (V.110).

`#define kTelDtcBearerDataRate300bpsV21 1`
: 300 bps (V.21).

`#define kTelDtcBearerDataRate38400bpsV110 81`
: 38400 bps (V.110 or X.31 flag stuffing).

`#define kTelDtcBearerDataRate38400bpsV120 49`
: 38400 bps (V.120).

`#define kTelDtcBearerDataRate48000bpsV110 82`
: 48000 bps (V.110 or X.31 flag stuffing).

`#define kTelDtcBearerDataRate48000bpsV120 50`
: 48000 bps (V.120).

`#define kTelDtcBearerDataRate4800bpsV110 70`
: 4800 bps (V.110 or X.31 flag stuffing).

`#define kTelDtcBearerDataRate4800bpsV120 38`
: 4800 bps (V.120).

`#define kTelDtcBearerDataRate4800bpsV32 6`
: 4800 bps (V.32).

`#define kTelDtcBearerDataRate56000bpsTrans 115`
: 56000 bps (bit transparent).

`#define kTelDtcBearerDataRate56000bpsV110 83`
: 56000 bps (V.110 or X.31 flag stuffing).

`#define kTelDtcBearerDataRate56000bpsV120 51`
: 56000 bps (V.120).

`#define kTelDtcBearerDataRate64000bpsTrans 116`
: 64000 bps (bit transparent).

`#define kTelDtcBearerDataRate9600bpsV110 71`
: 9600 bps (V.110 or X.31 flag stuffing).

`#define kTelDtcBearerDataRate9600bpsV120 39`
: 9600 bps (V.120).

`#define kTelDtcBearerDataRate9600bpsV32 7`
: 9600 bps (V.32).

`#define kTelDtcBearerDataRate9600bpsV34 12`
: 9600 bps (V.34).

`#define kTelDtcBearerDataRateAuto 0`
: Enable autobauding, the automatic selection of the speed. This setting is possible in the case of a 3.1-kHz modem and nontransparent service.

Information Types

Purpose Information types used in the `type` field of the `TelInfIdentificationType` structure.

Declared In `TelephonyLib.h`

Constants `#define kTelInfPhoneManufacturer 0`
Phone manufacturer.

`#define kTelInfPhoneModel 1`
Phone model.

`#define kTelInfPhoneRevision 2`
Phone revision.

`#define kTelInfPhoneSerialNumber 3`
Phone serial number.

`#define kTelInfSubscriberIdentity 4`
Subscriber identity.

Line IDs

Purpose IDs for speech and GPRS lines.

Declared In `TelephonyLib.h`

Constants `#define kTelSpcCallingLineId 0xFF`
ID of a calling line. We can't provide a real ID knowing that an error might occur after `TelSpcCallNumber()` returns. So use this one to "close" the line.

`#define kTelSpcGprsLineId 0xFE`
ID of a GPRS line.

Mute Status Constants

Purpose Mute status constants used in the `TelSndGetMuteStatus()` and `TelSndSetMuteStatus()` functions.

Declared In `TelephonyLib.h`

Constants `#define kTelSndMuteStatusOff 0`
Microphone is unmuted.

`#define kTelSndMuteStatusOn 1`
Microphone is muted.

MUX IOCTL Values

Purpose Specify IOCTL and other values related to controlling the phone MUX.

Declared In `TelephonyLib.h`

Constants

`#define IOC_PMUX '4'`
Specifies the IOCTL group for the phone MUX.

`#define kPhoneMuxType 'pmux'`
The database type for a phone mux. Specified in the `TelMuxInfoType` structure's `type` field.

`#define kPMuxChanClose _IO(IOC_PMUX, 3)`
Close the MUX channel specified by the `CLID` parameter.

`#define kPMuxChanOpen _IO(IOC_PMUX, 2)`
Open the MUX channel specified by the `CLID` parameter.

`#define kPMuxDisable _IO(IOC_PMUX, 1)`
Set the MUX mode to disabled (transparent).

`#define kPMuxEnable _IO(IOC_PMUX, 0)`
Set the MUX mode to enabled. If the `mode` parameter is 0, the MUX is in basic mode; if 1, the MUX is in extended mode.

MUX Status

Purpose Values that specify the status of the phone MUX.

Declared In `TelephonyLib.h`

Constants

`#define kTelMuxChanClosed 0`
The specified MUX channel is closed.

`#define kTelMuxChanOpened 1`
The specified MUX channel is open.

`#define kTelMuxChanStatusNotif 1`

`#define kTelMuxModeDisabled 0`
The MUX is disabled.

`#define kTelMuxModeEnabled 1`
The MUX is enabled.

`#define kTelMuxModeStatusNotif 0`

Network Operator Status Constants

Purpose Status values used in the `status` field of the `TelNwkOperatorType` structure.

Declared In `TelephonyLib.h`

Constants `#define kTelNwkOperatorStatusUnknow 0`
Unknown network status.

`#define kTelNwkOperatorStatusAvailable 1`
Network is available.

`#define kTelNwkOperatorStatusCurrent 2`
This network operator is the current operator.

`#define kTelNwkOperatorStatusForbidden 3`
Network is forbidden to be used.

Network Operator Types

Purpose Network types used in the `type` field of the `TelNwkOperatorType` structure.

Declared In `TelephonyLib.h`

Constants `#define kTelNwkTypeCdma 0`
CDMA network.

`#define kTelNwkTypeGsmGprs 1`
GSM GPRS network.

`#define kTelNwkTypeTdma 2`
TDMA network.

`#define kTelNwkTypePdc 3`
PDC network.

`#define kTelNwkTypeCdpd 4`
CDPD network.

Network Status Constants

Purpose Network status constants returned by `TelNwkGetStatus()` and in the `data` field of a `kTelNwkLaunchCmdNetworkStatusChange` notification.

Declared In `TelephonyLib.h`

Constants

`#define kTelNwkStatusNotRegisteredNotSearching 0`
Not registered and not searching.

`#define kTelNwkStatusRegisteredHome 1`
Registered and in the home area.

`#define kTelNwkStatusNotRegisteredSearching 2`
Not registered and searching.

`#define kTelNwkStatusRegistrationDenied 3`
Registration denied.

`#define kTelNwkStatusUnknow 4`
Unknown registration.

`#define kTelNwkStatusRegisteredRoaming 5`
Registered and roaming.

Notification Identifiers

Purpose Identifies the type of telephony notification. These values are used in the `id` field of the `TelNotificationType` structure.

Declared In `TelephonyLib.h`

Constants

`#define kTelCatLaunchCmdEndSession 31`
The running application on the card has terminated.

`#define kTelCatLaunchCmdExecCmd 30`
The card is currently running a CAT command. `data` is the identifier of the command, which is one of the `kTelCatCmd<cmd name>` constants described in "Card Command IDs" on page 92.

`#define kTelCatLaunchCmdNoApps 29`
: There are no CAT applications in the SIM card.

`#define kTelDtcLaunchCmdClosed 35`
: A data call session has stopped.

`#define kTelDtcLaunchCmdStarted 34`
: A data call session has started. `data` is the type of data connection, which is one of the values described in "Connection Types" on page 102. `data2` is additional information that depends on the connection type.

`#define kTelGprsLaunchCmdEventReporting 24`
: An event occurred on the GPRS connection. `data` is the event type. `data2` is 0, and `priority` is `kTelGprsNotificationPriority`.

`#define kTelGprsLaunchCmdNwkRegistration 25`
: The GPRS network location has changed—for example, +CGREG. `data` is the network status, which is one of the values described in "GPRS Network Registration Status" on page 107. The first 16 bits of `data2` is the location and area code and the second 16 bits is the cell ID, and `priority` is `kTelGprsNotificationPriority`.

`#define kTelGprsLaunchCmdSessionBytesExchanged 28`
: The number of data bytes exchanged during the last GPRS session is available. `data` is the number of uplink bytes exchanged, `data2` is the number of downlink bytes exchanged, and `priority` is `kTelGprsNotificationPriority`.

`#define kTelMuxLaunchCmdChanStatus 33`
: Provides the status of a given MUX channel. `data` is the channel ID and `data2` is the status, either `kTelMuxChanClosed` or `kTelMuxChanOpened`.

`#define kTelMuxLaunchCmdModeStatus 32`
: Provides the current MUX mode. `data` is either `kTelMuxModeEnabled` or `kTelMuxModeDisabled`.

`#define kTelNwkLaunchCmdNetworkStatusChange 11`
: Network status has changed. `data` is the new network status, which is one of the "Network Status Constants" on page 119.

`#define kTelNwkLaunchCmdSignalLevelChange 9`
: Network signal level has changed. `data` is the new signal level.

`#define kTelNwkLaunchCmdUssdAnswer 10`
: USSD answer is available. `data` is the result code of the USSD sequence and `data2` is the data size if any are available.

`#define kTelPowLaunchCmdBatteryChargeLevelChange 12`
: Battery charge level has changed. `data` is the new battery charge level.

`#define kTelPowLaunchCmdBatteryConnectionStatusChange 13`
: Battery connection status has changed. `data` is the new battery connection status.

`#define kTelPowLaunchCmdConnectionOff 15`
: Phone connection is off.

`#define kTelPowLaunchCmdConnectionOn 14`
: Phone connection is on. `data` is the authentication status.

`#define kTelPowLaunchCmdPhonebookNotReady 17`
: Phone book storage is not ready.

`#define kTelPowLaunchCmdPhonebookReady 16`
: Phone book storage is ready.

`#define kTelPowLaunchCmdSmsNotReady 19`
: SMS storage is not ready.

`#define kTelPowLaunchCmdSmsReady 18`
: SMS storage is ready.

`#define kTelSmsLaunchCmdIncomingMessage 0`
: Incoming SMS message. `data` is the storage ID and `data2` is the message ID.

`#define kTelSpcLaunchCmdCallAlerting 4`
: Call is alerting. `data` is the call ID and `data2` is a bit mask regrouping mode, direction and multiparty info.

`#define kTelSpcLaunchCmdCallConnect 1`
: Call is connected. `data` is the call ID and `data2` is a bit mask regrouping mode, direction and multiparty info.

`#define kTelSpcLaunchCmdCallDialing 3`
Dialing call. `data` is the call ID and `data2` is a bit mask regrouping mode, direction and multiparty info.

`#define kTelSpcLaunchCmdCallerIdAvailable 8`
Caller ID is available.

`#define kTelSpcLaunchCmdCallHeld 2`
Call is placed on hold. `data` is the call ID and `data2` is a bit mask regrouping mode, direction and multiparty info.

`#define kTelSpcLaunchCmdCallIncoming 5`
Incoming voice call. `data` is the call ID and `data2` is a bit mask regrouping mode, direction and multiparty info.

`#define kTelSpcLaunchCmdCallReleased 7`
Call has been released. `data` is the call ID and `data2` is the call duration.

`#define kTelSpcLaunchCmdCallWaiting 6`
Voice call is waiting (an incoming voice call has arrived while another call is active or on hold). `data` is the call ID and `data2` is a bit mask regrouping mode, direction and multiparty info.

`#define kTelStyLaunchCmdAuthenticated 20`
Authentication successful.

`#define kTelStyLaunchCmdAuthenticationCanceled 21`
Authentication canceled by user.

`#define kTelStyLaunchCmdNoPhoneProfileAvailable 23`
No phone profile is available.

`#define kTelStyLaunchCmdPhoneProfileAvailable 22`
At least one phone profile is available.

Notification Masks

Purpose Masks used to extract data from the `data2` field of the `TelNotificationType` structure for many of the telephony notifications.

Declared In `TelephonyLib.h`

Constants `#define kTelNotificationCallDirectionMask 0x00000010`
Used to extract call direction information.

`#define kTelNotificationCallMultipartyMask 0x00000020`
Used to extract multiparty information.

Notification Priorities

Purpose Notification priorities used in the `priority` field of the `TelNotificationType` structure.

Declared In `TelephonyLib.h`

Constants

`#define kTelCallNotificationPriority 0`
Voice call.

`#define kTelSmsNotificationPriority 1`
SMS message.

`#define kTelCallerNumberNotificationPriority 2`
Caller ID notification.

`#define kTelStkNotificationPriority 3`
Not used.

`#define kTelGprsNotificationPriority 3`
A change in the GPRS network: a GPRS event, network location, or the availability of the number of data bytes exchanged.

`#define kTelOtherNotificationPriority 4`
Other priority.

Number Types

Purpose Phone number types used in the `type` field of the `TelNumberType` structure.

Declared In `TelephonyLib.h`

Constants

`#define kTelNumberTypeInternational 145`
International number.

`#define kTelNumberTypeNational 161`
National number.

`#define kTelNumberTypeUnknown 129`
Unknown number type.

Phone Book Identifiers

Purpose Phone book identifiers used in the `idP` field of the `TelPhbPhonebooksType` structure and in the `id` field of the `TelPhbPhonebookType` structure.

Declared In `TelephonyLib.h`

Constants

`#define kTelPhbMEDialled 0x4443`
Phone dialed numbers phone book.

`#define kTelPhbEmergency 0x454E`
Phone or SIM emergency number list.

`#define kTelPhbSIMFixDialling 0x4644`
SIM fix dialing phone book.

`#define kTelPhbSIMLastDialling 0x4C44`
SIM last-dialed number phone book.

`#define kTelPhbMEMissed 0x4D43`
Phone missed calls list.

`#define kTelPhbME 0x4D45`
Phone phone book.

`#define kTelPhbMEAndSIM 0x4D54`
Combined phone and SIM phone book.

`#define kTelPhbOwnNumbers 0x4F4E`
Phone or SIM own numbers list.

`#define kTelPhbMEReceived 0x5243`
Phone received calls list.

`#define kTelPhbSD 0x5344`
SIM service number.

`#define kTelPhbSIM 0x534D`
SIM phone book.

`#define kTelPhbTA 0x5441`
Terminal adapter phone book.

Registration Search Modes

Purpose Registration search modes used in the `TelNwkGetRegistrationMode()` and

`TelNwkSetRegistration()` functions and the `TelNwkRegistrationType` structure.

Declared In `TelephonyLib.h`

Constants `#define kTelNwkRegistrationAutomatic 0`
Automatic search mode.

`#define kTelNwkRegistrationManual 1`
Manual search mode.

`#define kTelNwkRegistrationManualAutomatic 4`
If manual search mode fails, then automatic search mode is used.

Security Facility Status Constants

Purpose Status constants used in the `status` field of the `TelStyFacilityType` structure.

Declared In `TelephonyLib.h`

Constants `#define kTelStyFacilityStatusNotActive 0`
Facility is not active.

`#define kTelStyFacilityStatusActive 1`
Facility is active.

Security Facility Types

Purpose Security facility types used in the `type` field of the `TelStyFacilityPasswordType` and `TelStyFacilityType` structures.

Declared In `TelephonyLib.h`

Constants `#define kTelStyFacilityTypeAllBar 0x4142`
All barring services.

`#define kTelStyFacilityTypeAllInBar 0x4143`
All incoming barring services.

`#define kTelStyFacilityTypeAllOutBar 0x4147`
All outgoing barring services.

`#define kTelStyFacilityTypeAllIn 0x4149`
Bar all incoming calls.

`#define kTelStyFacilityTypeAllOut 0x414F`
Bar all outgoing calls.

`#define kTelStyFacilityTypeControl 0x4353`
Lock control surface.

`#define kTelStyFacilityTypeSIMFixDial 0x4644`
SIM fixed dialing memory.

`#define kTelStyFacilityTypeInRoaming 0x4952`
Bar incoming calls when roaming outside the home country.

`#define kTelStyFacilityTypePhoneLock 0x4D45`
Phone lock feature.

`#define kTelStyFacilityTypeInNotAny 0x4E41`
Bar incoming calls from numbers not stored in any memory.

`#define kTelStyFacilityTypeInNotME 0x4E4D`
Bar incoming calls from numbers not stored in the phone memory.

`#define kTelStyFacilityTypeInNotSIM 0x4E53`
Bar incoming calls from numbers not stored in SIM memory.

`#define kTelStyFacilityTypeInNotTA 0x4E54`
Bar incoming calls from numbers not stored in TA memory.

`#define kTelStyFacilityTypeOutInt 0x4F49`
Bar outgoing international calls.

`#define kTelStyFacilityTypeOutIntExHome 0x4F58`
Bar outgoing international calls except to home country.

`#define kTelStyFacilityTypeSimPin2 0x5032`
SIM PIN 2.

`#define kTelStyFacilityTypeCorpPerso 0x5043`
Corporate personalization.

`#define kTelStyFacilityTypeFirstSim 0x5046`
First SIM entered.

`#define kTelStyFacilityTypeNetPerso 0x504E`
Network personalization.

`#define kTelStyFacilityTypeSerProPerso 0x5050`
Service provider personalization.

`#define kTelStyFacilityTypePhoneSim 0x5053`
Lock phone to current SIM card and ask for password when a different SIM card is inserted.

`#define kTelStyFacilityTypeNetSubPerso 0x5055`
Network subset personalization.

`#define kTelStyFacilityTypeSim 0x5343`
SIM.

SMS Data Encoding Schemes

Purpose Data encoding schemes used in the `dataCodingScheme` field of `TelSmsMessageType` structure.

Declared In `TelephonyLib.h`

Constants `#define kTelSms8BitsEncoding 0`
8-bit encoding.

`#define kTelSmsBitsASCIIEncoding 1`
ANSI X3.4 encoding.

`#define kTelSmsIA5Encoding 2`
CCITT T.50 encoding.

`#define kTelSmsIS91Encoding 3`
TIA/EIA/IS-91 section 3.7.1 encoding.

`#define kTelSmsUCS2Encoding 4`
UCS2 encoding; used with GSM only.

`#define kTelSmsDefaultGSMEncoding 5`
Default encoding for GSM only.

`#define kTelSmsAutomatic 6`
The Telephony Manager automatically chooses the best encoding.

SMS Delivery Status Reports

Purpose Delivery status report codes used in the `report` field of the `TelSmsReportMessageType` structure.

Declared In `TelephonyLib.h`

Constants

`#define kTelSmsDSRSuccess 0`
Sucess.

`#define kTelSmsDSRMessageReplaced 1`
Message replaced.

`#define kTelSmsDSRMessageForwarded 2`
Message forwarded.

`#define kTelSmsDSRTempCongestion 3`
Temporarily not delivered due to congestion.

`#define kTelSmsDSRTempSMEBusy 4`
Temporarily not delivered due to the mobile phone being busy.

`#define kTelSmsDSRTempServiceRejected 5`
Temporarily not delivered because the service rejected the message.

`#define kTelSmsDSRTempServiceUnavailable 6`
Temporarily not delivered due to the service being unavailable.

`#define kTelSmsDSRTempSMEError 7`
Temporarily not delivered due to an error in the mobile phone.

`#define kTelSmsDSRTempOther 8`
Temporarily not delivered due to some other cause.

`#define kTelSmsDSRPermRPError 9`
Delivery failed due to a reply path error.

`#define kTelSmsDSRPermBadDestination 10`
Delivery failed due to a bad destination address.

`#define kTelSmsDSRPermUnobtainable 11`
Delivery failed due to an error.

`#define kTelSmsDSRPermServiceUnavailable 12`
Delivery failed due to service unavailability.

`#define kTelSmsDSRPermInternetworkError 13`
Delivery failed due to an internetworking error.

`#define kTelSmsDSRPermValidityExpired 14`
Delivery failed due to its validity expiring.

`#define kTelSmsDSRPermDeletedByOrigSME 15`
Delivery failed due to the message being deleted by the originating mobile phone.

`#define kTelSmsDSRPermDeleteByAdm 16`
Delivery failed due to the message being deleted.

`#define kTelSmsDSRPermSMNotExist 17`
Delivery failed.

`#define kTelSmsDSRPermOther 18`
Delivery failed due to some other cause.

SMS Extension Types

Purpose Extension types used in the `type` field of the `TelSmsExtensionType` structure.

Declared In `TelephonyLib.h`

Constants `#define kTelSmsMultiPartExtensionTypeId 0x00`
Multipart short message with 8-bit concatenation.

`#define kTelSmsSpecialIndicationExtensionTypeId 0x01`
Special SMS message indication.

`#define kTelSmsNbsExtensionTypeId 0x04`
NBS message with a short port number value.

`#define kTelSmsNbs2ExtensionTypeId 0x05`
NBS message with a long port number value.

`#define kTelSmsMultiPart2ExtensionTypeId 0x08`
Multipart short message with 16-bit concatenation.

SMS Message Class Constants

Purpose Message class types used in the `messageClass` field of the `TelSmsGsmDeliverMessageType` and `TelSmsGsmSubmitMessageType` structures.

Declared In `TelephonyLib.h`

Constants `#define kTelSmsClass0 0x00`
Class 0.

`#define kTelSmsClass1 0x01`
Default meaning mobile equipment specific.

`#define kTelSmsClass2 0x02`
SIM specific message.

`#define kTelSmsClass3 0x03`
Default meaning terminal equipment specific.

`#define kTelSmsUnknownClass 0xFF`
Class not specified.

SMS Message Status Constants

Purpose Message class types used in the `status` field of the `TelSmsMessageType` structure.

Declared In `TelephonyLib.h`

Constants `#define kTelSmsStatusReceivedUnread 0`
Received and unread message.

`#define kTelSmsStatusReceivedRead 1`
Received and read message.

`#define kTelSmsStatusStoredUnsent 2`
Stored and unsent message.

`#define kTelSmsStatusStoredSent 3`
Stored and sent message.

SMS Message Transport Protocol Constants

Purpose Message transport protocol types used in the `protocolId` field of the `TelSmsGsmDeliverMessageType` and `TelSmsGsmSubmitMessageType` structures.

Declared In `TelephonyLib.h`

Constants

`#define kTelSmsDefaultProtocol 0`
Default message transport protocol.

`#define kTelSmsFaxProtocol 1`
Fax message.

`#define kTelSmsX400Protocol 2`
X.400 message.

`#define kTelSmsPagingProtocol 3`
Paging message.

`#define kTelSmsEmailProtocol 4`
Email message.

`#define kTelSmsErmesProtocol 5`
Ermes message.

`#define kTelSmsVoiceProtocol 6`
Voice message.

SMS Message Types

Purpose Message types used in the `messageType` field of the `TelSmsMessageType` structure.

Declared In `TelephonyLib.h`

Constants

`#define kTelSmsMessageTypeDelivered 0`
Delivered message.

`#define kTelSmsMessageTypeReport 1`
Report message.

`#define kTelSmsMessageTypeSubmitted 2`
Submitted message.

`#define kTelSmsMessageTypeManualAck 3`
Manual acknowledgement message.

`#define kTelSmsMessageAllTypes 4`
All messages.

SMS Report Types

Purpose Report types used in the `reportType` field of the `TelSmsReportMessageType` structure.

Declared In `TelephonyLib.h`

Constants `#define kTelSmsStatusReportDeliveryType 0`
Status report or delivery acknowledgement.

`#define kTelSmsManualAckDeliveryType 1`
Manual delivery acknowledgement.

SMS Special Indication Types

Purpose Special indication types used in the `type` field of the `TelSmsSpecialIndicationExtensionType` structure.

Declared In `TelephonyLib.h`

Constants `#define kTelSmsSpecialIndicationTypeVM 0x00`
Voicemail message waiting.

`#define kTelSmsSpecialIndicationTypeFax 0x01`
Fax message waiting.

`#define kTelSmsSpecialIndicationTypeEmail 0x02`
Email message waiting.

`#define kTelSmsSpecialIndicationTypeOther 0x03`
Other message waiting.

SMS Storage Locations

Purpose Storage locations used in the `idP` array of the `TelSmsStoragesType` structure, and in the `id` field of the `TelSmsStorageType` structure.

Declared In `TelephonyLib.h`

Constants `#define kTelSmsStoragePhone 0x4D45`
Telephone storage.

`#define kTelSmsStorageAdaptor 0x5341`
Telephone adapter storage.

`#define kTelSmsStorageSIM 0x534D`
SIM storage.

Telephony Initialization Values

Purpose Values used to initialize parameters.

Declared In `TelephonyLib.h`

Constants `#define kTelInvalidAppId -1`
Use this constant to initialize the *`telDescP`* parameter to the `TelOpen()` and `TelOpenPhoneProfile()` functions. The Telephony Manager never assigns this value.

`#define kTelInvalidTransId 0`
Use this constant to initialize the *`ioTransIdP`* parameter to all functions that can be called asynchronously. The Telephony Manager never assigns this value for an asynchronous transaction ID.

Telephony Manager Error Codes

Purpose Error codes returned by the various Telephony Manager functions.

Declared In `TelephonyLib.h`

Constants `#define telErrAlreadyAuthenticating (telErrorClass | 0x29)`
Driver is already authenticating, wait for the notification `kTelStyLaunchCmdAuthenticated`.

```
#define telErrAlreadyConnected (telErrorClass |
  0x52)
```
A connection has already been made with the specified connection profile.

```
#define telErrBatteryLevelTooLow (telErrorClass |
  0x3A)
```
The device battery level is too low to allow opening the phone connection.

```
#define telErrBufferSize (telErrorClass | 0x07)
```
Buffer used to retrieve data is too small.

```
#define telErrCodingScheme (telErrorClass | 0x1C)
```
Specified short message coding scheme is invalid.

```
#define telErrCommandFailed (telErrorClass | 0x0B)
```
Phone couldn't perform the associated command; check the phone driver.

```
#define telErrCommunicationPortAlreadyUsed
  (telErrorClass | 0x2A)
```
Communication port is in use by another application.

```
#define telErrCorporatePINRequired (telErrorClass
  | 0x38)
```
Phone is waiting for the corporate personalization password to be given.

```
#define telErrCorporatePUKRequired (telErrorClass
  | 0x39)
```
Phone is waiting for the corporate personalization unblocking password to be given.

```
#define telErrDriverNotFound (telErrorClass |
  0x1F)
```
Phone driver specified in the phone profile was not found.

```
#define telErrEntryNotFound (telErrorClass | 0x14)
```
Entry not found.

```
#define telErrFeatureNotSupported (telErrorClass |
  0x08)
```
Feature is not supported by the phone or the network.

```
#define telErrGprsIllegalME (telErrorClass | 0x3C)
```
A GPRS attach operation failed because of illegal mobile equipment (ME).

`#define telErrGprsIllegalMS (telErrorClass | 0x3B)`
A GPRS attach operation failed because of an illegal mobile station (MS).

`#define telErrGprsInvalidMobileClass (telErrorClass | 0x46)`
The mobile class is detected as invalid during a GPRS connection.

`#define telErrGprsLocationAreaNotAllowed (telErrorClass | 0x3F)`
A GPRS data connection is not allowed at the current location.

`#define telErrGprsOperatorResourceInsufficient (telErrorClass | 0x47)`
Operator resources are insufficient to establish a GPRS data connection.

`#define telErrGprsPdpActivationRejectedGGSN (telErrorClass | 0x49)`
The PDP activation was rejected by the GGSN.

`#define telErrGprsPdpActivationRejectedUnspecified (telErrorClass | 0x4A)`
The PDP activation was rejected by the operator.

`#define telErrGprsPDPAuthenticationFailure (telErrorClass | 0x45)`
The authentication step failed during a GPRS data connection.

`#define telErrGprsPdpDeactivationNetworkFailure (telErrorClass | 0x4C)`
The operator deactivated the GPRS data connection.

`#define telErrGprsPdpDeactivationRegular (telErrorClass | 0x4B)`
The operator deactivated the GPRS data connection.

`#define telErrGprsPLMNNotAllowed (telErrorClass | 0x3E)`
Access to the Public Land Mobile Network (PLMN) is not allowed.

```
#define
  telErrGprsRequestedServiceOptionNotSubscribed
  (telErrorClass | 0x42)
```
The requested service option is not allowed because the user is not subscribed.

```
#define
  telErrGprsRoamingNotAllowedInThisLocationArea
  (telErrorClass | 0x40)
```
GPRS roaming is not allowed at the current location.

```
#define telErrGprsServiceOptionNotSupported
  (telErrorClass | 0x41)
```
The requested service option is not supported.

```
#define
  telErrGprsServiceOptionTemporarilyOutOfOrder
  (telErrorClass | 0x43)
```
The requested service option is temporarily down.

```
#define telErrGprsServicesNotAllowed
  (telErrorClass | 0x3D)
```
GPRS services are not allowed.

```
#define telErrGprsUnknowOrMissingAPN
  (telErrorClass | 0x48)
```
An unknown or missing APN was used to establish a GPRS data connection.

```
#define telErrGprsUnspecifiedError (telErrorClass
  | 0x44)
```
The default value of a Telephony Manager GPRS function error.

```
#define telErrInvalidDial (telErrorClass | 0x17)
```
Invalid character in the dial string.

```
#define telErrInvalidIndex (telErrorClass | 0x13)
```
Invalid index when accessing a store.

```
#define telErrInvalidParameter (telErrorClass |
  0x1A)
```
One of the function parameters is invalid.

```
#define telErrInvalidString (telErrorClass | 0x16)
```
Invalid character in text string.

`#define telErrLimitedCompatibility (telErrorClass | 0x25)`
Current driver is only partially compatible with the connected phone.

`#define telErrMemAllocation (telErrorClass | 0x02)`
Memory allocation error.

`#define telErrMuxBusy (telErrorClass | 0x51)`
The phone MUX is busy.

`#define telErrMuxChanNotAvailable (telErrorClass | 0x50)`
A phone MUX channel is not available.

`#define telErrMuxChanTypeNotSupported (telErrorClass | 0x4F)`
The phone driver does not support the specified phone MUX channel type.

`#define telErrMuxNotSupported (telErrorClass | 0x4E)`
The phone MUX is not supported.

`#define telErrNetworkNotAllowed (telErrorClass | 0x27)`
Network access not allowed, except for emergency calls only.

`#define telErrNetworkPINRequired (telErrorClass | 0x32)`
Phone is waiting for the network personalization password to be given.

`#define telErrNetworkPUKRequired (telErrorClass | 0x33)`
Phone is waiting for the network personalization unblocking password to be given.

`#define telErrNetworkSubsetPINRequired (telErrorClass | 0x34)`
Phone is waiting for the network subset personalization password to be given.

`#define telErrNetworkSubsetPUKRequired (telErrorClass | 0x35)`
Phone is waiting for the network subset personalization unblocking password to be given.

```
#define telErrNetworkTimeOut (telErrorClass |
  0x19)
```
Network didn't reply within the allowed time period.

```
#define telErrNoNetwork (telErrorClass | 0x18)
```
No network available.

```
#define telErrNoSIMInserted (telErrorClass | 0x0D)
```
No SIM inserted.

```
#define telErrOperationNotAllowed (telErrorClass |
  0x28)
```
Operation not allowed.

```
#define telErrPassword (telErrorClass | 0x11)
```
Incorrect password.

```
#define telErrPhoneComm (telErrorClass | 0x09)
```
No communication link with the phone.

```
#define telErrPhoneMemAllocation (telErrorClass |
  0x12)
```
Phone memory is full.

```
#define telErrPhoneMemFailure (telErrorClass |
  0x15)
```
Phone encountered a memory error.

```
#define telErrPhoneNumber (telErrorClass | 0x1D)
```
Specified short message SMSC or destination phone number is invalid.

```
#define telErrPhoneReply (telErrorClass | 0x0A)
```
Phone reply syntax is incorrect; check the phone driver.

```
#define telErrPhoneToFirstSIMPINRequired
  (telErrorClass | 0x2E)
```
Phone is waiting for the phone-to-first SIM card password to be given.

```
#define telErrPhoneToFirstSIMPUKRequired
  (telErrorClass | 0x2F)
```
Phone is waiting for the phone-to-first SIM card unblocking password to be given.

`#define telErrPhoneToSIMPINRequired (telErrorClass | 0x2D)`
Phone is waiting for the phone-to-SIM card password to be given.

`#define telErrProfileConflict (telErrorClass | 0x26)`
Current profile conflicts with the requested profile.

`#define telErrProviderPINRequired (telErrorClass | 0x36)`
Phone is waiting for the service provider personalization password to be given.

`#define telErrProviderPUKRequired (telErrorClass | 0x37)`
Phone is waiting for the service provider personalization unblocking password to be given.

`#define telErrResultBusyResource (telErrorClass | 0x05)`
Resource is busy.

`#define telErrResultTimeOut (telErrorClass | 0x03)`
Timeout was reached.

`#define telErrResultUserCancel (telErrorClass | 0x04)`
User cancelled the action.

`#define telErrSecurity (telErrorClass | 0x06)`
Phone access has not been granted.

`#define telErrSettings (telErrorClass | 0x23)`
Invalid telephony settings. Phone panel preferences don’t exist or the Telephony profile is not correctly configured.

`#define telErrSIMBusy (telErrorClass | 0x0F)`
SIM couldn't reply.

`#define telErrSIMFailure (telErrorClass | 0x0E)`
SIM is not working properly.

`#define telErrSIMPIN2Required (telErrorClass | 0x30)`
Phone is waiting for the SIM PIN2 to be given.

`#define telErrSIMPINRequired (telErrorClass | 0x2B)`
: Phone is waiting for the SIM PIN to be given.

`#define telErrSIMPUK2Required (telErrorClass | 0x31)`
: Phone is waiting for the SIM PUK2 to be given.

`#define telErrSIMPUKRequired (telErrorClass | 0x2C)`
: Phone is waiting for the SIM PUK to be given.

`#define telErrSIMWrong (telErrorClass | 0x10)`
: Phone is not accepting the SIM.

`#define telErrSpcCallError (telErrorClass | 0x21)`
: Call has encountered an error.

`#define telErrSpcLineIsBusy (telErrorClass | 0x0C)`
: Phone line is busy.

`#define telErrSpcLineIsReleased (telErrorClass | 0x20)`
: Call has been released.

`#define telErrUnavailableValue (telErrorClass | 0x24)`
: The requested value cannot be retrieved at this time.

`#define telErrUnknown (telErrorClass | 0x01)`
: Unknown Telephony Manager internal error.

`#define telErrValidityPeriod (telErrorClass | 0x1B)`
: Specified short message validity period is invalid.

`#define telErrValueStale (telErrorClass | 0x1E)`
: Information couldn't be retrieved; a copy of the last retrieved value was returned.

`#define telErrVersion (telErrorClass | 0x22)`
: Shared library version doesn't match the application version.

TelMessages Enum

Purpose Identifies a function.

Declared In `TelephonyLibTypes.h`

Constants These function name constants have the following format:

`kTel`*`functionName`*`Message`

where *`functionName`* is replaced by a function name. Examples include:

`kTelCancelMessage`
The `TelCancel()` function.

`kTelTestPhoneDriverMessage`
The `TelTestPhoneDriver()` function.

`kTelCncOpenMessage`
The `TelCncOpen()` function.

For a complete list, see the `TelephonyLibTypes.h` file.

Comments These values are used for the *`iFunctionId`* parameter of the `TelIsFunctionSupported()` function.

TelServices Enum

Purpose Identifies a service (group of related functions).

Declared In `TelephonyLibTypes.h`

Constants

`kTelCncServiceId`
Connection service.

`kTelNwkServiceId`
Network service.

`kTelStyServiceId`
Security service.

`kTelPowServiceId`
Power service.

`kTelCfgServiceId`
Configuration service.

`kTelSmsServiceId`
SMS service.

`kTelEmcServiceId`
: Emergency call service.

`kTelSpcServiceId`
: Speech call service.

`kTelPhbServiceId`
: Phone Book service.

`kTelSndServiceId`
: Sound service.

`kTelInfServiceId`
: Information service.

`kTelOemServiceId`
: OEM service.

`kTelGprsServiceId`
: GPRS service.

`kTelCatServiceId`
: CAT service.

`kTelMuxServiceId`
: MUX service.

`kTelLastServiceId = kTelMuxServiceId`
: The last value of this enum.

Comments These values are used for the *`iServiceId`* parameter of the `TelIsServiceAvailable()` function.

USSD Result Codes

Purpose Result codes used in the `result` field of the `TelNwkUssdType` structure.

Declared In `TelephonyLib.h`

Constants `#define kTelNwkUssdNoFurtherUserActionRequired 0`
: No further user action required.

`#define kTelNwkUssdFurtherUserActionRequired 1`
: Further user action required.

`#define kTelNwkUssdTerminatedByNetwork 2`
: USSD terminated by network.

`#define kTelNwkUssdOtherClientResponded 3`
Other local client has responsed.

`#define kTelNwkUssdOperationNotSupported 4`
Operation not supported.

`#define kTelNwkUssdNetworkTimeOut 5`
Network timeout.

Version Constants

Purpose Version of the Telephony Manager and SMS API.

Declared In `TelephonyLib.h`

Constants

```
#define kTelMgrVersion
  sysMakeROMVersion(kTelMgrVersionMajor,
  kTelMgrVersionMinor, kTelMgrVersionFix,
  kTelMgrStage, kTelMgrVersionBuild)
```
The Telephony Manager version information.

`#define kTelSmsAPIVersion 0x0001`
Version of the SMS API.

Vibrator Modes

Purpose Phone vibrator alert modes used in the `TelCfgGetVibratorMode()` and `TelCfgSetVibratorMode()` functions.

Declared In `TelephonyLib.h`

Constants

`#define kTelCfgVibratorModeDisable 0`
Vibrator is disabled.

`#define kTelCfgVibratorModeEnable 1`
Vibrator is enabled.

Telephony Manager Events

kTelTelephonyEvent

Purpose Sent when an asynchronously called Telephony Manager function completes.

Declared In `TelephonyLib.h`

Prototype `#define kTelTelephonyEvent telAsyncReplyEvent`

Comments The `TelEvtGetEvent()` and `TelEvtGetTelephonyEvent()` functions both return a `TelEventType` structure to provide information about a telephony-related event.

You call the `TelEvtGetEvent()` function to retrieve telephony and other events.

You call the `TelEvtGetTelephonyEvent()` function to retrieve only telephony events. This function does not consume non-telephony events.

See Also "Telephony Events" on page 10 and Chapter 3, "Events and the Event Loop," in *Exploring Palm OS: Programming Basics*.

Telephony Manager Notifications

kTelTelephonyNotification

Purpose Broadcast by the Telephony Manager when various telephony events occur. Applications interested in such events can register to receive this notification.

Declared In `TelephonyLib.h`

Prototype `#define kTelTelephonyNotification 'tmgr'`

Parameters The `notifyDetailsP` field of the notification parameter block points to a `TelNotificationType` structure.

See Also "Notification Identifiers" on page 119, and Chapter 11, "Notification Manager," in *Exploring Palm OS: Programming Basics*.

Telephony Manager Functions and Macros

TelCancel Function

Purpose Cancels an asynchronous function call.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelCancel (int32_t telDesc,
   uint16_t iCanceledTransId,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iCanceledTransId`*
The transaction ID associated with the function that you are canceling.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error, such as `telErrCommandFailed`, is returned if the function call could not be cancelled. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` function must have been called.

This function cancels a pending asynchronous function call. You can cancel any asynchronous call except for an asynchronous call to the `TelCancel()` function.

The function call that is cancelled returns the `telErrUserCancel` error code.

You can check if this function is supported by using the macro `TelIsCancelSupported(`*`telDesc`*`)`.

TelCardGetFile Function

Purpose Retrieves the content and the properties of a specific file within the card's file system given the path and filename.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelCardGetFile (int32_t iTelDesc,
   TelCardFileType *ioFileP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`iTelDesc`*
The telephony file descriptor.

↔ *`ioFileP`*
A pointer to a `TelCardFileType` structure.

On input, the `pathP` field specifies the path of the file on the card, the `bufSize` field specifies the size of the `bufP` buffer, the `partOffset` and `partSize` fields specify the offset and size of the part of the file to retrieve, the `mode` field specifies the type of file access requested, and the `recId` field specifies the record to be read.

Upon return, the remaining fields receive the requested content from the file and information about the file.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCardGetFileSupported (`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the structure referenced by *`ioFileP`* remains in memory until the asynchronous call completes.

TelCatCallAction Function

Purpose Inform the card whether the user accepted or rejected to set up the call.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelCatCallAction (int32_t iTelDesc,
   uint8_t iAction, uint16_t *ioTransIdP)
```

Parameters

→ *iTelDesc*
The telephony file descriptor.

→ *iAction*
One of the values described in "Card Call Set Up Actions" on page 91.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCatCallActionSupported (`*`telDesc`*`)`.

TelCatGetCmdParameters Function

Purpose Retrieve the parameters of the currently running proactive command.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelCatGetCmdParameters
   (int32_t iTelDesc,
   TelCatCmdParamsType *ioParamsP,
   uint16_t *ioTransIdP)
```

Parameters

→ *iTelDesc*
The telephony file descriptor.

↔ *`ioParamsP`*

A pointer to a `TelCatCmdParamsType` structure.

On input, the `cmdParamP` field specifies a structure associated with the command and the `cmdParamSize` field specifies the size of the `cmdParamP` buffer.

Upon return, the remaining fields receive the parameters and other information about the currently running command.

↔ *`ioTransIdP`*

If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCatGetCmdParametersSupported (`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the structure referenced by *`ioParamsP`* remains in memory until the asynchronous call completes.

Most proactive commands use an extended parameter block to define more properties than the one described in `TelCatCmdParamsType`. A CAT type is related to each proactive command that needs extended parameters. The caller must allocate a block in the application, set `cmdParamP` to point on this block, and set `cmdParamSize` to the size of this block. The block must be large enough to handle the extended structure as well as all items that the structure can reference—for example, strings and sub-structures. A good size for this block is 1024 bytes: APDUs have a maximum of 256 bytes and the decoded information should not be larger than four times the encoded information.

TelCatGetConfig Function

Purpose Retrieve the current configuration parameters from the card.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelCatGetConfig (int32_t iTelDesc,
    TelCatConfigType *ioCfgP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`iTelDesc`*
The telephony file descriptor.

↔ *`ioCfgP`*
A pointer to a `TelCatConfigType` structure.

On input, the `profileSize` field specifies the size of the `profileP` buffer.

Upon return, the remaining fields receive the current configuration parameters from the card.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCatGetConfigSupported (`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the structure referenced by *`ioCfgP`* remains in memory until the asynchronous call completes.

See Also `TelCatSetConfig()`

TelCatMenuSelection Function

Purpose Notify the card to launch an application or to provide its help information if there is any.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelCatMenuSelection (int32_t iTelDesc,
   TelCatMenuSelectionType *iSelectionP,
   uint16_t *ioTransIdP)
```

Parameters

→ *iTelDesc*
The telephony file descriptor.

→ *iSelectionP*
A pointer to a `TelCatMenuSelectionType` structure.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCatMenuSelectionSupported (`*`telDesc`*`)`.

TelCatNotifyCardOfEvent Function

Purpose Notify the card of an event that has occurred in Palm OS.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelCatNotifyCardOfEvent
   (int32_t iTelDesc,
   TelCatEventToCardType *iEventP,
   uint16_t *ioTransIdP)
```

Parameters

→ *iTelDesc*
The telephony file descriptor.

→ *iEventP*
A pointer to a `TelCatEventToCardType` structure.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCatNotifyCardOfEventSupported (`*`telDesc`*`)`.

TelCatSetCmdResponse Function

Purpose Send a specific response for the currently running proactive command.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelCatSetCmdResponse (int32_t iTelDesc,
    TelCatCmdResponseType *iResponseP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`iTelDesc`*
The telephony file descriptor.

→ *`iResponseP`*
A pointer to a `TelCatCmdResponseType` structure.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCatSetCmdResponseSupported (`*`telDesc`*`)`.

TelCatSetConfig Function

Purpose Informs the card about the Palm OS supported Card Application Toolkit features as well as the language setting.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelCatSetConfig (int32_t iTelDesc,
   TelCatConfigType *iCfgP, uint16_t *ioTransIdP)
```

Parameters

→ *`iTelDesc`*
The telephony file descriptor.

→ *`iCfgP`*
A pointer to a `TelCatConfigType` structure.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCatSetConfigSupported (`*`telDesc`*`)`.

See Also `TelCatGetConfig()`

TelCatTerminate Function

Purpose Notify the card to terminate the current command/session for the given reason.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelCatTerminate (int32_t iTelDesc,
   uint8_t iReason, uint16_t *ioTransIdP)
```

Parameters

→ *`iTelDesc`*
The telephony file descriptor.

→ *iReason*
One of the values described in "Card Browser Termination Cause Codes" on page 91.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCatTerminateSupported (`*`telDesc`*`)`.

TelCfgGetAlertSoundMode Function

Purpose Gets the current alert sound mode of the phone.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelCfgGetAlertSoundMode
   (int32_t telDesc, uint8_t *oAlertSoundModeP,
   uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
The telephony file descriptor.

← *oAlertSoundModeP*
Pointer to the alert sound mode. One of the constants described in "Alert Sound Modes" on page 84.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgGetAlertSoundModeSupported(`*`telDesc`*`)`.

GSM AT Command AT+CALM? (GSM 07.07)

See Also `TelCfgSetAlertSoundMode()`

TelCfgGetCallForwarding Function

Purpose Gets the call forwarding number and conditions.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelCfgGetCallForwarding
   (int32_t telDesc,
   TelCfgCallForwardingPtr ioCallForwardingP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`ioCallForwardingP`*
Pointer to a `TelCfgCallForwardingType` structure that contains the forwarding number and conditions.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgGetCallForwardingSupported(`*`telDesc`*`)`.

GSM AT Command AT+CCFC=x (GSM 07.07)

See Also `TelCfgSetCallForwarding()`

TelCfgGetCallIdRestrictionStatus Function

Purpose Gets the call identifier restriction status.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelCfgGetCallIdRestrictionStatus
    (int32_t telDesc,
    uint8_t *oCallIdRestrictionP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oCallIdRestrictionP`*
Pointer to a call identifier restriction value.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgGetCallIdRestrictionStatusSupported(`*`telDesc`*`)`.

GSM AT Command AT+CLIR? (GSM 07.07)

See Also `TelCfgSetCallIdRestrictionStatus()`

TelCfgGetLoudspeakerVolumeLevel Function

Purpose Retrieves the loudspeaker volume level of the phone.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelCfgGetLoudspeakerVolumeLevel
   (int32_t telDesc,
   uint8_t *oLoudspeakerVolumeLevelP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oLoudspeakerVolumeLevelP`*
A pointer to the loudspeaker volume level.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgGetLoudspeakerVolumeLevelSupported(`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the structure referenced by *`oLoudspeakerVolumeLevelP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CLVL? (GSM 07.07)

See Also `TelCfgGetLoudspeakerVolumeLevelRange()`, `TelCfgSetLoudspeakerVolumeLevel()`

TelCfgGetLoudspeakerVolumeLevelRange Function

Purpose Gets the loudspeaker volume level range.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelCfgGetLoudspeakerVolumeLevelRange
    (int32_t telDesc,
    TelCfgLevelRangePtr oLoudspeakerVolumeLevelRangeP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oLoudspeakerVolumeLevelRangeP`*
A pointer to a `TelCfgLevelRangeType` structure. Upon return, this structure contains the minimum level and the maximum level of the phone loudspeaker volume.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgGetLoudspeakerVolumeLevelRangeSupported(`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the structure referenced by *`oLoudspeakerVolumeLevelRangeP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CLVL=? (GSM 07.07)

See Also `TelCfgGetLoudspeakerVolumeLevel()`, `TelCfgSetLoudspeakerVolumeLevel()`

TelCfgGetPhoneNumber Function

Purpose Gets the connected telephone numbers (voice, fax, and data).

Declared In `TelephonyLib.h`

Prototype
```
status_t TelCfgGetPhoneNumber (int32_t telDesc,
   TelCfgPhoneNumberPtr ioPhoneNumberP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioPhoneNumberP`*
A pointer to a `TelCfgPhoneNumberType` structure.

On input, the `voice.voiceNumberSize` field specifies the allocated size of the `voice.voiceNumberP` buffer. The `fax.faxNumberSize` field specifies the allocated size of the `fax.faxNumberP` buffer. The `data.dataNumberSize` field specifies the allocated size of the `data.dataNumberP` buffer.

Upon return, the `voice.voiceNumberP` buffer contains the voice phone number, and the `voice.voiceNumberSize` field specifies the size of the voice phone number. The `fax.faxNumberP` buffer contains the fax phone number, and the `fax.faxNumberSize` field specifies the size of the fax phone number. The `data.dataNumberP` buffer contains the data phone number, and the `data.dataNumberSize` field specifies the size of the data phone number

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgGetPhoneNumberSupported(`*`telDesc`*`)`.

If the `voice.voiceNumberP` buffer is too small to contain the complete voice phone number, the voice phone number is truncated (and ends with the null terminated character) and this function returns the `telErrBufferSize` error. The `voice.voiceNumberSize` field will contain the size needed to retrieve the complete voice phone number.

If the `fax.faxNumberP` is too small to contain the complete fax phone number, the fax phone number is truncated (and ends with the null terminated character) and this function returns the `telErrBufferSize` error. The `fax.faxNumberSize` field will contain the size needed to retrieve the complete fax phone number.

If the `data.dataNumberP` is too small to contain the complete data phone number, the data phone number is truncated (and ends with the null terminated character) and this function returns the `telErrBufferSize` error. The `data.dataNumberSize` field will contain the size needed to retrieve the complete data phone number.

When using this function asynchronously, you must ensure that the structure referenced by *`ioPhoneNumberP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CPBR or AT+CNUM (GSM 07.07)

See Also `TelCfgSetPhoneNumber()`

TelCfgGetRingerSoundLevel Function

Purpose Gets the current ringer sound level of the phone.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelCfgGetRingerSoundLevel
    (int32_t telDesc, uint8_t *oRingerSoundLevelP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oRingerSoundLevelP`*
A pointer to the ringer sound level of the phone.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgGetRingerSoundLevelSupported(`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the structure referenced by *`oRingerSoundLevelP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CRSL? (GSM 07.07)

See Also `TelCfgGetRingerSoundLevelRange()`, `TelCfgSetRingerSoundLevel()`

TelCfgGetRingerSoundLevelRange Function

Purpose Gets the ringer sound level range of the phone.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelCfgGetRingerSoundLevelRange
   (int32_t telDesc,
   TelCfgLevelRangePtr oRingerSoundLevelRangeP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oRingerSoundLevelRangeP`*
A pointer to a `TelCfgLevelRangeType` structure.

Upon return, this structure contains the minimum and maximum level of the phone ringer volume.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgGetRingerSoundLevelRangeSupported(`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the structure referenced by *`oRingerSoundLevelRangeP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CRSL=? (GSM 07.07)

See Also `TelCfgGetRingerSoundLevel()`, `TelCfgSetRingerSoundLevel()`

TelCfgGetSmsCenter Function

Purpose Gets the SMS Service Center telephone number.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelCfgGetSmsCenter (int32_t telDesc,
    TelNumberPtr ioSmsCenterP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioSmsCenterP`*
A pointer to a `TelNumberType` structure.

On input, the `size` field of this structure specifies the allocated size of the `numberP` buffer.

Upon return, the `numberP` buffer contains the dial number string, and the `size` field specifies the size of the dial number string.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgGetSmsCenterSupported(`*`telDesc`*`)`.

If the `numberP` buffer is too small to contain the complete dial number, the dial number is truncated (and ends with the null terminated character) and this function returns the `telErrBufferSize` error. The `size` field will contain the size needed to retrieve the complete dial number.

When using this function asynchronously, you must ensure that the structure referenced by *`ioSmsCenterP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CSCA? (GSM 07.07)

See Also `TelCfgSetSmsCenter()`

TelCfgGetVibratorMode Function

Purpose Gets the current vibrator alert mode of the phone.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelCfgGetVibratorMode (int32_t telDesc,
   uint8_t *oVibratorModeP, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oVibratorModeP`*
A pointer to the current status of the phone vibrator alert feature. One of the constants described in "Vibrator Modes" on page 143.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgGetVibratorModeSupported(`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the structure referenced by *`oVibratorModeP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CVIB? (GSM 07.07)

See Also `TelCfgSetVibratorMode()`

TelCfgGetVoiceMailNumber Function

Purpose Gets the voice mail number.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelCfgGetVoiceMailNumber
    (int32_t telDesc,
    TelNumberPtr ioVoiceMailNumberP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioVoiceMailNumberP`*
A pointer to a `TelNumberType` structure.

On input, the `size` field of this structure specifies the allocated size of the `numberP` buffer.

Upon return, the `numberP` buffer contains the voice mail number string, and the `size` field specifies the size of the voice mail number string.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgGetVoiceMailNumberSupported(`*`telDesc`*`)`.

If the `numberP` buffer is too small to contain the complete voice mail number, the voice mail number is truncated (and ends with the null terminated character) and this function returns the `telErrBufferSize` error. The `size` field will contain the size needed to retrieve the complete voice mail number.

When using this function asynchronously, you must ensure that the structure referenced by *`ioVoiceMailNumberP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CSVM? (GSM 07.07)

See Also `TelCfgSetVoiceMailNumber()`

TelCfgSetAlertSoundMode Function

Purpose Sets the alert sound mode of the phone.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelCfgSetAlertSoundMode
   (int32_t telDesc, uint8_t iAlertSoundMode,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iAlertSoundMode`*
Alert sound mode. One of the constants described in "Alert Sound Modes" on page 84.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgSetAlertSoundModeSupported(`*`telDesc`*`)`.

GSM AT Command AT+CALM=x (GSM 07.07)

See Also `TelCfgGetAlertSoundMode()`

TelCfgSetCallForwarding Function

Purpose Sets the call forwarding number and conditions.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelCfgSetCallForwarding
    (int32_t telDesc,
    TelCfgCallForwardingPtr iCallForwardingP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iCallForwardingP`*
Pointer to a `TelCfgCallForwardingType` structure.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgSetCallForwardingSupported(`*`telDesc`*`)`.

GSM AT Command AT+CCFC=x (GSM 07.07)

See Also `TelCfgGetCallForwarding()`

TelCfgSetCallIdRestrictionStatus Function

Purpose Sets the call identifier restriction status.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelCfgSetCallIdRestrictionStatus
   (int32_t telDesc, uint8_t iCallIdRestriction,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iCallIdRestriction`*
Call identifier restriction.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgSetCallIdRestrictionStatusSupported(`*`telDesc`*`)`.

GSM AT Command AT+CLIR=x (GSM 07.07)

See Also `TelCfgGetCallIdRestrictionStatus()`

TelCfgSetLoudspeakerVolumeLevel Function

Purpose Sets the loudspeaker volume level of the phone.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelCfgSetLoudspeakerVolumeLevel
   (int32_t telDesc,
   uint8_t iLoudspeakerVolumeLevel,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iLoudspeakerVolumeLevel`*
The loudspeaker volume level to set.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgSetLoudspeakerVolumeLevelSupported(`*`telDesc`*`)`.

GSM AT Command AT+CLVL=X (GSM 07.07)

See Also `TelCfgGetLoudspeakerVolumeLevel()`, `TelCfgGetLoudspeakerVolumeLevelRange()`

TelCfgSetPhoneNumber Function

Purpose Sets the connected telephone number (voice, fax, and data).

Declared In `TelephonyLib.h`

Prototype
```
status_t TelCfgSetPhoneNumber (int32_t telDesc,
   TelCfgPhoneNumberPtr iPhoneNumberP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iPhoneNumberP`*
A pointer to a `TelCfgPhoneNumberType` structure. This structure contains the voice, fax or data phone number to set.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgSetPhoneNumberSupported(`*`telDesc`*`)`.

If a field of the `TelCfgPhoneNumberType` structure is `NULL`, then its value is not stored in the phone. To clear a value, specify an empty string.

GSM AT Command AT+CPBW (GSM 07.07)

See Also `TelCfgGetPhoneNumber()`

TelCfgSetRingerSoundLevel Function

Purpose Sets the ringer sound level of the phone.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelCfgSetRingerSoundLevel
    (int32_t telDesc, uint8_t iRingerSoundLevel,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iRingerSoundLevel`*
The ringer sound level to set.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgSetRingerSoundLevelSupported(`*`telDesc`*`)`.

GSM AT Command AT+CRSL=x (GSM 07.07)

See Also `TelCfgGetPhoneNumber()`

TelCfgSetSmsCenter Function

Purpose Sets the SMS Service Center telephone number.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelCfgSetSmsCenter (int32_t telDesc,
    TelNumberPtr iSmsCenterP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iSmsCenterP`*
A pointer to a `TelNumberType` structure.

The `dialNumberP` value must point to a null terminated telephone number string for the SMS Service Center.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgSetSmsCenterSupported(`*`telDesc`*`)`.

GSM AT Command AT+CSCA=x (GSM 07.07)

See Also `TelCfgGetSmsCenter()`

TelCfgSetVibratorMode Function

Purpose Sets the vibrator alert mode of the phone to on or off.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelCfgSetVibratorMode (int32_t telDesc,
   uint8_t iVibratorMode, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iVibratorMode`*
Vibrator alert mode to set. One of the constants described in "Vibrator Modes" on page 143.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgSetVibratorModeSupported(`*`telDesc`*`)`.

GSM AT Command AT+CVIB=x (GSM 07.07)

See Also `TelCfgGetVibratorMode()`

TelCfgSetVoiceMailNumber Function

Purpose Sets the voice mail number.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelCfgSetVoiceMailNumber
    (int32_t telDesc,
    TelNumberPtr iVoiceMailNumberP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iVoiceMailNumberP`*
A pointer to a `TelNumberType` structure.

The `dialNumberP` value must point to a null terminated string containing the voice mail number.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsCfgSetVoiceMailNumberSupported(`*`telDesc`*`)`.

GSM AT Command AT+CSVM=x (GSM 07.07)

See Also `TelCfgGetVoiceMailNumber()`

TelClose Function

Purpose Closes the Telephony library, cleans up the memory used, and deactivates the Telephony Server, if the application is the last one to use the Telephony Manager.

Declared In `TelephonyLib.h`

Prototype `status_t TelClose (int32_t` *`telDesc`*`)`

Parameters → *`telDesc`*
The telephony file descriptor.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error.

Comments The `TelOpen()` function must have been called.

Call this function when you are done with the Telephony Manager. This function is always synchronous.

If no other application is using the Telephony Manager, this function stops the Telephony Server and releases any resources used by the Telephony Manager.

See Also `TelCncClose()`

TelCncClose Function

Purpose Closes the connection to the phone.

Declared In `TelephonyLib.h`

Prototype `status_t TelCncClose (int32_t` *`telDesc`*`)`

Parameters → *`telDesc`*
The telephony file descriptor.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

This function is always synchronous.

See Also `TelClose()`, `TelCncOpen()`

TelCncGetStatus Function

Purpose Retrieves the status of the connection to the phone.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelCncGetStatus (int32_t telDesc,
   uint8_t *oStatusP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oStatusP`*
A pointer to the connection's status. The value is 0 if the connection is closed, and 1 if the connection is opened.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error.

Comments The `TelOpen()` function must have been called.

This function is always synchronous.

See Also `TelCncClose()`, `TelCncOpen()`

TelCncOpen Function

Purpose Opens the connection to the phone, using the transport layer provided in the telephony profile (current or specified).

Declared In `TelephonyLib.h`

Prototype

```
status_t TelCncOpen (int32_t telDesc,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` function must have been called.

The connection to the transport is synchronous but just after the connection, if successful, the init string is sent synchronously or asynchronously.

See Also `TelCncClose()`

TelEmcDial Function

Purpose Calls the emergency service.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelEmcDial (int32_t telDesc,
   TelSpcCallPtr oCallP, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`oCallP`*
Pointer to a `TelSpcCallType` structure that contains information about the call.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsEmcDialSupported(`*`telDesc`*`)`.

GSM AT Command ATDxxx; (GSM 07.07)

See Also `TelIsEmcServiceAvailable()`

TelEvtGetEvent Function

Purpose Gets both telephony and standard Palm OS events.

Declared In `TelephonyLib.h`

Prototype

```
void TelEvtGetEvent (int32_t telDesc,
   EventPtr oEventP, int32_t iTimeOut)
```

Parameters

→ *telDesc*
The telephony file descriptor.

← *oEventP*
Pointer to a `TelEventType` or `EventType` structure holding the retrieved event.

→ *iTimeOut*
Timeout value.

Returns Nothing.

Comments The `TelOpen()` function must have been called.

This function must be called by every application that uses the Telephony Manager, instead of `EvtGetEvent()`.

See Also `TelEvtGetTelephonyEvent()`

TelEvtGetTelephonyEvent Function

Purpose Gets only telephony events.

Declared In `TelephonyLib.h`

Prototype

```
void TelEvtGetTelephonyEvent (int32_t telDesc,
   EventPtr oEventP, int32_t iTimeOut)
```

Parameters

→ *telDesc*
The telephony file descriptor.

← *`oEventP`*
Pointer to a `TelEventType` structure holding the retrieved event.

→ *`iTimeOut`*
Timeout value.

Returns Nothing.

Comments The `TelOpen()` function must have been called.

Use this function instead of the function `TelEvtGetEvent()` when you want to process only telephony events.

TelGprsGetAttach Function

Purpose Retrieves the attachment state (attached or detached) of the GPRS service.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelGprsGetAttach (int32_t telDesc,
   uint8_t *oAttach, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oAttach`*
A pointer to the attachment state. One of the constants described in "GPRS Attachment State" on page 104.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsGetAttachSupported (`*`telDesc`*`)`.

GSM AT Command AT+CGATT=<state> (GSM 07.07)

See Also `TelGprsSetAttach()`

TelGprsGetAvailableContextId Function

Purpose Retrieves an available PDP context ID (CID).

Declared In `TelephonyLib.h`

Prototype

```
status_t TelGprsGetAvailableContextId
    (int32_t telDesc, uint8_t *oContextIdP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oContextIdP`*
A pointer to an available context ID—that is, one that is deactivated.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsGetAvailableContextIdSupported (`*`telDesc`*`)`.

GSM AT Command AT+CGDCONT=? (for CID range) and
AT+CGACT? (for CID activation status)
(GSM 07.07)

See Also `TelGprsGetContext()`, `TelGprsSetContext()`

TelGprsGetContext Function

Purpose Retrieves a PDP context.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelGprsGetContext (int32_t telDesc,
   TelGprsContextPtr ioContextP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*

The telephony file descriptor.

↔ *`ioContextP`*

A pointer to a `TelGprsContextType` structure.

On input, the `contextID` field specifies the context to retrieve, the `accessPointNameSize` field specifies the size of the `accessPointNameP` buffer, the `pdpAddressSize` field specifies the size of the `pdpAddressP` buffer, and the `OSPIHHostSize` field specifies the size of the `OSPIHHostP` buffer.

Upon return, the `pdpType` field contains one of the "GPRS Packet Data Protocols" on page 108. The `accessPointNameP` buffer contains the access point name (if `accessPointNameSize` is not zero), and the `accessPointNameSize` field specifies the size of the access point name. The `pdpAddressP` buffer contains the PDP address (if `pdpAddressSize` is not zero), and the `pdpAddressSize` field specifies the size of the PDP address. The `dataCompression` and `headerCompression` fields are set to values described in "GPRS Compression Settings" on page 104. If the `pdpType` field is set to `kTelGprsPdpOSPIH` and `OSPIHHostSize` is not zero, then the `OSPIHHostP` buffer contains the OSPIH host name, and the `OSPIHHostSize` field specifies the size of the host name. If the `pdpType` field is set to `kTelGprsPdpOSPIH`, then `OSPIHPort` is set to the TCP or UDP port on the Internet Host (see "GPRS OSPIH Protocol Settings" on page 108) and the `OSPIHProtocol` field is set to the protocol used over IP, either TCP or UDP.

↔ *`ioTransIdP`*

If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsGetContextSupported (`*`telDesc`*`)`.

If the `accessPointNameP`, `pdpAddressP`, or `OSPIHHostP` buffer is too small to contain the information to be retrieved, the corresponding string is truncated (and ends with the null terminated character) and this function returns the `telErrBufferSize` error. The `accessPointNameSize`, `pdpAddressSize`, or `OSPIHHostSize` field, respectively, contains the size needed to retrieve the complete string.

When using this function asynchronously, you must ensure that the structure referenced by *`ioContextP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CGDCONT? (GSM 07.07)

See Also `TelGprsSetContext()`

TelGprsGetDataCounter Function

Purpose Retrieves GPRS data counters for a current or a previous session given a PDP context.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelGprsGetDataCounter (int32_t telDesc,
    TelGprsDataCounterPtr oDataCounterP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *oDataCounterP*
A pointer to a `TelGprsDataCounterType` structure.

On input, the `contextID` field specifies the ID of the PDP context to retrieve counters for.

Upon return, the remaining fields receive the number of uploaded and downloaded bytes and packets.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsGetDataCounterSupported (`*telDesc*`)`.

GSM AT Command No standard GSM 07.07 AT command for this feature.

TelGprsGetDefinedCids Function

Purpose Retrieves the list of defined PDP context IDs (CIDs).

Declared In `TelephonyLib.h`

Prototype

```
status_t TelGprsGetDefinedCids (int32_t telDesc,
    TelGprsDefinedCidsPtr ioCidsP,
    uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
The telephony file descriptor.

↔ *ioCidsP*
A pointer to a `TelGprsDefinedCidsType` structure.

On input, if you set the `cidsP` field to `NULL` and `cidCount` to 0, then this function returns only the count of defined context IDs in `cidCount`, and `errNone`. No CID information is returned.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsGetDefinedCidsSupported (`*`telDesc`*`)`.

GSM AT Command AT+CGPADDR=? (GSM 07.05)

See Also `TelGprsSetContext()`, `TelGprsGetContext()`

TelGprsGetEventReporting Function

Purpose Retrieves the selected mode for the sending of the unsolicited result code `+CGEV:XXX` when certain events occur in the GPRS phone/ module or the network.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelGprsGetEventReporting
    (int32_t telDesc,
    TelGprsEventReportingPtr oEvtReportP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oEvtReportP`*
A pointer to a `TelGprsEventReportingType` structure, which receives the event reporting mode and a value that indicates the effect on buffered unsolicited result codes.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsGetEventReportingSupported (telDesc)`.

GSM AT Command AT+CGEREP? (GSM 07.07)

See Also `TelGprsSetEventReporting()`

TelGprsGetNwkRegistration Function

Purpose Retrieves the current GPRS network registration information: mode, status, location area code, and cell ID.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelGprsGetNwkRegistration
    (int32_t telDesc,
    TelGprsNwkRegistrationPtr ioRegistrationP,
    uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
The telephony file descriptor.

↔ *ioRegistrationP*
A pointer to a `TelGprsNwkRegistrationType` structure, which receives network registration information.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsGetNwkRegistrationSupported (`*`telDesc`*`)`.

GSM AT Command AT+CGREG? (GSM 07.07)

See Also `TelGprsSetNwkRegistration()`

TelGprsGetPdpActivation Function

Purpose Retrieves the state (activated or deactivated) of a PDP context.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelGprsGetPdpActivation
   (int32_t telDesc,
   TelGprsPdpActivationPtr ioPdpActivationP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioPdpActivationP`*
A pointer to a `TelGprsPdpActivationType` structure.

On input, the `contextID` field specifies the ID of the context. Upon return, the `state` field receives one of the values defined in "GPRS PDP Activation State" on page 108.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsGetPdpActivationSupported (`*`telDesc`*`)`.

GSM AT Command AT+CGACT? (GSM 07.07)

See Also `TelGprsSetPdpActivation()`

TelGprsGetPdpAddress Function

Purpose Retrieves the PDP address for the specified PDP context ID.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelGprsGetPdpAddress (int32_t telDesc,
   TelGprsPdpAddressPtr ioPdpAddressP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioPdpAddressP`*
A pointer to a `TelGprsPdpAddressType` structure.

On input, the `contextID` field specifies the context address to retrieve and the `pdpAddressSize` field specifies the size of the `pdpAddressP` buffer.

Upon return, the `pdpAddressP` buffer contains the PDP address (if `pdpAddressSize` is not zero), and the `pdpAddressSize` field specifies the size of the address.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsGetPdpAddressSupported (`*`telDesc`*`)`.

If the `pdpAddressP` buffer is too small to contain the information to be retrieved, the corresponding string is truncated (and ends with the null terminated character) and this function returns the `telErrBufferSize` error. The `pdpAddressSize` field contains the size needed to retrieve the complete string.

GSM AT Command AT+CGPADDR=<cid> (GSM 07.07)

TelGprsGetQosCurrent Function

Purpose Retrieves the current quality of service for an activated PDP context.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelGprsGetQosCurrent (int32_t telDesc,
    TelGprsQosPtr ioQosCurrentP,
    uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
The telephony file descriptor.

↔ *ioQosCurrentP*
A pointer to a `TelGprsQosType` structure.

On input, the `contextID` field specifies the PDP context ID.

Upon return, the remaining fields receive the current quality of service parameters.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsGetQosCurrentSupported (`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the structure referenced by *ioQosCurrentP* remains in memory until the asynchronous call completes.

GSM AT Command No standard GSM 07.07 AT command for this feature.

TelGprsGetQosMinimum Function

Purpose Retrieves the minimum acceptable quality of service for a PDP context.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelGprsGetQosMinimum (int32_t telDesc,
   TelGprsQosPtr ioQosMinimumP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioQosMinimumP`*
A pointer to a `TelGprsQosType` structure.

On input, the `contextID` field specifies the PDP context ID.

Upon return, the remaining fields receive the minimum acceptable quality of service parameters.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsGetQosMinimumSupported (`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the structure referenced by *`ioQosMinimumP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CGQMIN? (GSM 07.07)

See Also `TelGprsSetQosMinimum()`

TelGprsGetQosRequested Function

Purpose Retrieves the quality of service requested for a PDP context.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelGprsGetQosRequested (int32_t telDesc,
   TelGprsQosPtr ioQosRequestedP,
   uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
The telephony file descriptor.

↔ *ioQosRequestedP*
A pointer to a `TelGprsQosType` structure.

On input, the `contextID` field specifies the PDP context ID.

Upon return, the remaining fields receive the quality of service parameters.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsGetQosRequestedSupported (`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the structure referenced by *`ioQosRequestedP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CGQREQ? (GSM 07.07)

See Also `TelGprsSetQosRequested()`

TelGprsGetSmsService Function

Purpose Retrieves the selected service type used for SMS messages.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelGprsGetSmsService (int32_t telDesc,
   uint8_t *oSMSService, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oSMSService`*
One of the values described in "GPRS SMS Service Preferences" on page 112.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsGetSmsServiceSupported (`*`telDesc`*`)`.

GSM AT Command AT+CGSMS? (GSM 07.07)

See Also `TelGprsSetSmsService()`

TelGprsSetAttach Function

Purpose Attaches or detaches the mobile terminal to or from the GPRS service.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelGprsSetAttach (int32_t telDesc,
   uint8_t iAttach, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *iAttach*
Attach or detach. One of the constants described in "GPRS Attachment State" on page 104.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsSetAttachSupported (`*`telDesc`*`)`.

GSM AT Command AT+CGATT=<attach> (GSM 07.07)

See Also `TelGprsGetAttach()`

TelGprsSetContext Function

Purpose Sets the parameters of a PDP context.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelGprsSetContext (int32_t telDesc,
   TelGprsContextPtr iContextP,
   uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
The telephony file descriptor.

→ *iContextP*
A pointer to a `TelGprsContextType` structure, which specifies the packet data protocol (PDP) context ID, information about the PDP, APN, data and header compression settings, and other information.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsSetContextSupported (`*`telDesc`*`)`.

GSM AT Command AT+CGDCONT = [<cid>[,<PDP_type>[,<APN>[,<d_comp>[,<h_comp>[,<pdp1>[,...[,[,pdpN]]]]]]]]]
(GSM 07.07)

See Also `TelGprsGetContext()`

TelGprsSetEventReporting Function

Purpose Enables or disables the sending of the unsolicited result code `+CGEV:XXX` when certain events occur in the GPRS phone/module or the network.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelGprsSetEventReporting
    (int32_t telDesc,
    TelGprsEventReportingPtr iEvtReportP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iEvtReportP`*
A pointer to a `TelGprsEventReportingType` structure, which specifies the event reporting mode and a value that specifies the effect on buffered unsolicited result codes.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsSetEventReportingSupported (`*`telDesc`*`)`.

GSM AT Command AT+CGEREP=[<mode>[, <buffer>]] (GSM 07.07)

See Also `TelGprsGetEventReporting()`

TelGprsSetNwkRegistration Function

Purpose Controls the presentation of an unsolicited result code when there is a change in network registration or a change of the network cell.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelGprsSetNwkRegistration
    (int32_t telDesc, uint8_t iRegistrationType,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iRegistrationType`*
One of the values described in "GPRS Network Registration Settings" on page 106.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsSetNwkRegistrationSupported (`*`telDesc`*`)`.

GSM AT Command AT+CGREG=<n> (GSM 07.07)

See Also `TelGprsGetNwkRegistration()`

TelGprsSetPdpActivation Function

Purpose Activates or deactivates a PDP context.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelGprsSetPdpActivation
   (int32_t telDesc,
   TelGprsPdpActivationPtr iPdpActivationP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iPdpActivationP`*
A pointer to a `TelGprsPdpActivationType` structure, which specifies the ID of the context and whether to activate or deactivate it.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsSetPdpActivationSupported (`*`telDesc`*`)`.

GSM AT Command AT+CGACT=[<state> [,<cid> [,<cid> [,...]]]] (GSM 07.07)

See Also `TelGprsGetPdpActivation()`

TelGprsSetQosMinimum Function

Purpose Sets the minimum acceptable quality of service at the PDP context activation.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelGprsSetQosMinimum (int32_t telDesc,
    TelGprsQosPtr iQosMinimumP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iQosMinimumP`*
A pointer to a `TelGprsQosType` structure, which specifies the PDP context ID and the minimum acceptable quality of service parameters.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsSetQosMinimumSupported (`*`telDesc`*`)`.

GSM AT Command AT+CGQMIN=[<cid> [,<precedence>[,<delay>
[,<reliability>[,<peak>[,<mean>]]]]]]
(GSM 07.07)

See Also `TelGprsGetQosMinimum()`

TelGprsSetQosRequested Function

Purpose Sets the quality of service requested at the PDP context activation.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelGprsSetQosRequested (int32_t telDesc,
   TelGprsQosPtr iQosRequestedP,
   uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
The telephony file descriptor.

→ *iQosRequestedP*
A pointer to a `TelGprsQosType` structure, which specifies the PDP context ID and the quality of service parameters.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsSetQosRequestedSupported (telDesc)`.

GSM AT Command AT+CGQREQ=[<cid> [,<precedence>[,<delay>
[,<reliability>[,<peak>[,<mean>]]]]]]
(GSM 07.07)

See Also `TelGprsGetQosRequested()`

TelGprsSetSmsService Function

Purpose Selects the service type for SMS messages.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelGprsSetSmsService (int32_t telDesc,
   uint8_t iSMSService, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iSMSService`*
One of the values described in "GPRS SMS Service Preferences" on page 112.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsGprsSetSmsServiceSupported (`*`telDesc`*`)`.

GSM AT Command AT+CGSMS=<service> (GSM 07.07)

See Also `TelGprsGetSmsService()`

TelInfGetCallsDuration Function

Purpose Gets information about the last call duration, the total calls received duration, and the total calls dialed duration.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelInfGetCallsDuration (int32_t telDesc,
   TelInfCallsDurationPtr ioCallsDurationP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`ioCallsDurationP`*
Pointer to a `TelInfCallsDurationType` structure.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsInfGetCallsDurationSupported(`*`telDesc`*`)`.

See Also `TelInfResetCallsDuration()`

TelInfGetCallsList Function

Purpose Gets a list of the specified type of calls (missed, retrieved, or dialed), or the count of calls.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelInfGetCallsList (int32_t telDesc,
   TelInfCallsListPtr ioCallsListP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *ioCallsListP*
: Pointer to a TelInfCallsListType structure. On input, specify the type of calls to receive in the `type` field.

: On input, if you set the `listP` field to `NULL` and `count` to 0, then this function returns only the count of calls in `count`, and `errNone`. No other call information is returned.

↔ *ioTransIdP*
: If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsInfGetCallsListSupported(`*telDesc*`)`.

GSM AT Command AT+CPBS="XX" (GSM 07.07)

See Also `TelInfResetCallsList()`

TelInfGetIdentification Function

Purpose Gets phone identification information including manufacturer, model, revision, serial number or the international mobile subscriber identity.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelInfGetIdentification
    (int32_t telDesc,
    TelInfIdentificationPtr ioParamP,
    uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
: The telephony file descriptor.

↔ `ioParamP`
Pointer to a `TelInfIdentificationType` structure.

On input, the `type` field must be a valid type (one of the Information Types constants.

↔ `ioTransIdP`
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsInfGetIdentificationSupported(`*`telDesc`*`)`.

GSM AT Command AT+CGMM; AT+CGMI; AT+CGMR (GSM 07.07)

See Also `TelInfResetCallsList()`

TelInfResetCallsDuration Function

Purpose Resets all call duration timers.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelInfResetCallsDuration
   (int32_t telDesc, uint16_t *ioTransIdP)
```

Parameters → `telDesc`
The telephony file descriptor.

↔ `ioTransIdP`
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsInfResetCallsDurationSupported(`*`telDesc`*`)`.

See Also `TelInfGetCallsDuration()`

TelInfResetCallsList Function

Purpose Empty the calls list.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelInfResetCallsList (int32_t telDesc,
   uint8_t iCallTypeP, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iCallTypeP`*
Type of calls list to reset. Specify one of the Call Types constants.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsInfResetCallsListSupported(`*`telDesc`*`)`.

GSM AT Command AT+CPBS="XXC" (GSM 07.07)

See Also `TelInfGetCallsList()`

TelIsCatServiceAvailable Macro

Purpose Checks if the Card Application Toolkit (CAT) service set (group of related functions) is supported by the phone, driver, and network.

Declared In `TelephonyLib.h`

Prototype `#define TelIsCatServiceAvailable (telDesc)`

Parameters → *`telDesc`*
The telephony file descriptor.

Returns Returns `errNone` if the service set is supported, or returns `telErrFeatureNotSupported` if it is not supported.

Comments Calling this macro is the same as calling the function `TelIsServiceAvailable()` and passing `kTelCatServiceId` for the *`iServiceId`* parameter.

See Also `TelIsFunctionSupported()`

TelIsCfgServiceAvailable Macro

Purpose Checks if the Configuration service set (group of related functions) is supported by the phone, driver, and network.

Declared In `TelephonyLib.h`

Prototype `#define TelIsCfgServiceAvailable(telDesc)`

Parameters → *`telDesc`*
The telephony file descriptor.

Returns Returns `errNone` if the service set is supported, or returns `telErrFeatureNotSupported` if it is not supported.

Comments Calling this macro is the same as calling the function `TelIsServiceAvailable()` and passing `kTelCfgServiceId` for the *`iServiceId`* parameter.

See Also `TelIsFunctionSupported()`

TelIsCncServiceAvailable Macro

Purpose Checks if the Connection service set (group of related functions) is supported by the phone, driver, and network.

Declared In `TelephonyLib.h`

Prototype `#define TelIsCncServiceAvailable (`*`telDesc`*`)`

Parameters → *`telDesc`*
The telephony file descriptor.

Returns Returns `errNone` if the service set is supported, or returns `telErrFeatureNotSupported` if it is not supported.

Comments Calling this macro is the same as calling the function `TelIsServiceAvailable()` and passing `kTelCncServiceId` for the *`iServiceId`* parameter.

See Also `TelIsFunctionSupported()`

TelIsEmcServiceAvailable Macro

Purpose Checks if the Emergency Call service set (group of related functions) is supported by the phone, driver, and network.

Declared In `TelephonyLib.h`

Prototype `#define TelIsEmcServiceAvailable(`*`telDesc`*`)`

Parameters → *`telDesc`*
The telephony file descriptor.

Returns Returns `errNone` if the service set is supported, or returns `telErrFeatureNotSupported` if it is not supported.

Comments Calling this macro is the same as calling the function `TelIsServiceAvailable()` and passing `kTelEmcServiceId` for the *`iServiceId`* parameter.

See Also `TelIsFunctionSupported()`

TelIsFunctionSupported Function

Purpose Checks if a function is supported by the phone, driver, and network.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelIsFunctionSupported (int32_t telDesc,
   uint16_t iFunctionId)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iFunctionId`*
Identifier of the function to check. Specify one of the `TelMessages` constants.

Returns Returns `errNone` if the function is supported, or returns `telErrFeatureNotSupported` if it is not supported.

Comments The `TelephonyLib.h` header file also defines a series of macros that call this function, passing in the appropriate function identifier. These macros have the form `TelIs`*`FunctionName`*`Supported(`*`telDesc`*`)`.

See Also `TelIsServiceAvailable()`

TelIsGprsServiceAvailable Macro

Purpose Checks if the GPRS service set (group of related functions) is supported by the phone, driver, and network.

Declared In `TelephonyLib.h`

Prototype
```
#define TelIsGprsServiceAvailable (telDesc)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

Returns Returns `errNone` if the service set is supported, or returns `telErrFeatureNotSupported` if it is not supported.

Comments Calling this macro is the same as calling the function `TelIsServiceAvailable()` and passing `kTelGPRSServiceId` for the *`iServiceId`* parameter.

See Also `TelIsFunctionSupported()`

TelIsInfServiceAvailable Macro

Purpose Checks if the Information service set (group of related functions) is supported by the phone, driver, and network.

Declared In `TelephonyLib.h`

Prototype `#define TelIsInfServiceAvailable(`*`telDesc`*`)`

Parameters → *`telDesc`*
The telephony file descriptor.

Returns Returns `errNone` if the service set is supported, or returns `telErrFeatureNotSupported` if it is not supported.

Comments Calling this macro is the same as calling the function `TelIsServiceAvailable()` and passing `kTelInfServiceId` for the *`iServiceId`* parameter.

See Also `TelIsFunctionSupported()`

TelIsMuxServiceAvailable Macro

Purpose Checks if the MUX service set (group of related functions) is supported by the phone, driver, and network.

Declared In `TelephonyLib.h`

Prototype `#define TelIsMuxServiceAvailable (`*`telDesc`*`)`

Parameters → *`telDesc`*
The telephony file descriptor.

Returns Returns `errNone` if the service set is supported, or returns `telErrFeatureNotSupported` if it is not supported.

Comments Calling this macro is the same as calling the function `TelIsServiceAvailable()` and passing `kTelMuxServiceId` for the *`iServiceId`* parameter.

See Also `TelIsFunctionSupported()`

TelIsNwkServiceAvailable Macro

Purpose Checks if the Network service set (group of related functions) is supported by the phone, driver, and network.

Declared In `TelephonyLib.h`

Prototype `#define TelIsNwkServiceAvailable(`*`telDesc`*`)`

Parameters → *`telDesc`*
The telephony file descriptor.

Returns Returns `errNone` if the service set is supported, or returns `telErrFeatureNotSupported` if it is not supported.

Comments Calling this macro is the same as calling the function `TelIsServiceAvailable()` and passing `kTelNwkServiceId` for the *`iServiceId`* parameter.

See Also `TelIsFunctionSupported()`

TelIsOemServiceAvailable Macro

Purpose Checks if the OEM service set (group of related functions) is supported by the phone, driver, and network.

Declared In `TelephonyLib.h`

Prototype `#define TelIsOemServiceAvailable(`*`telDesc`*`)`

Parameters → *`telDesc`*
The telephony file descriptor.

Returns Returns `errNone` if the service set is supported, or returns `telErrFeatureNotSupported` if it is not supported.

Comments Calling this macro is the same as calling the function `TelIsServiceAvailable()` and passing `kTelOemServiceId` for the *`iServiceId`* parameter.

See Also `TelIsFunctionSupported()`

TelIsPhbServiceAvailable Macro

Purpose Checks if the Phone book service set (group of related functions) is supported by the phone, driver, and network.

Declared In `TelephonyLib.h`

Prototype `#define TelIsPhbServiceAvailable(`*`telDesc`*`)`

Parameters → *`telDesc`*
The telephony file descriptor.

Returns Returns `errNone` if the service set is supported, or returns `telErrFeatureNotSupported` if it is not supported.

Comments Calling this macro is the same as calling the function `TelIsServiceAvailable()` and passing `kTelPhbServiceId` for the *`iServiceId`* parameter.

See Also `TelIsFunctionSupported()`

TelIsPowServiceAvailable Macro

Purpose Checks if the Power service set (group of related functions) is supported by the phone, driver, and network.

Declared In `TelephonyLib.h`

Prototype `#define TelIsPowServiceAvailable(`*`telDesc`*`)`

Parameters → *`telDesc`*
The telephony file descriptor.

Returns Returns `errNone` if the service set is supported, or returns `telErrFeatureNotSupported` if it is not supported.

Comments Calling this macro is the same as calling the function `TelIsServiceAvailable()` and passing `kTelPowServiceId` for the *`iServiceId`* parameter.

See Also `TelIsFunctionSupported()`

TelIsServiceAvailable Function

Purpose Checks if a service set (group of related functions) is supported by the phone, driver, and network.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelIsServiceAvailable (int32_t telDesc,
   uint16_t iServiceId)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iServiceId`*
Identifier of the service set to check. Specify one of the `TelServices` constants.

Returns Returns `errNone` if the service set is supported, or returns `telErrFeatureNotSupported` if it is not supported.

Comments The `TelephonyLib.h` header file also defines a series of macros that call this function, passing in the appropriate service set identifier. These macros have the form `TelIs`*`ServiceName`*`ServiceAvailable(`*`telDesc`*`)`.

See Also `TelIsFunctionSupported()`

TelIsSmsServiceAvailable Macro

Purpose Checks if the SMS service set (group of related functions) is supported by the phone, driver, and network.

Declared In `TelephonyLib.h`

Prototype `#define TelIsSmsServiceAvailable(`*`telDesc`*`)`

Parameters

→ *`telDesc`*
The telephony file descriptor.

Returns Returns `errNone` if the service set is supported, or returns `telErrFeatureNotSupported` if it is not supported.

Comments Calling this macro is the same as calling the function `TelIsServiceAvailable()` and passing `kTelSmsServiceId` for the *`iServiceId`* parameter.

See Also `TelIsFunctionSupported()`

TelIsSndServiceAvailable Macro

Purpose Checks if the Sound service set (group of related functions) is supported by the phone, driver, and network.

Declared In `TelephonyLib.h`

Prototype `#define TelIsSndServiceAvailable(`*`telDesc`*`)`

Parameters → *`telDesc`*
The telephony file descriptor.

Returns Returns `errNone` if the service set is supported, or returns `telErrFeatureNotSupported` if it is not supported.

Comments Calling this macro is the same as calling the function `TelIsServiceAvailable()` and passing `kTelSndServiceId` for the *`iServiceId`* parameter.

See Also `TelIsFunctionSupported()`

TelIsSpcServiceAvailable Macro

Purpose Checks if the Speech service set (group of related functions) is supported by the phone, driver, and network.

Declared In `TelephonyLib.h`

Prototype `#define TelIsSpcServiceAvailable(`*`telDesc`*`)`

Parameters → *`telDesc`*
The telephony file descriptor.

Returns Returns `errNone` if the service set is supported, or returns `telErrFeatureNotSupported` if it is not supported.

Comments Calling this macro is the same as calling the function `TelIsServiceAvailable()` and passing `kTelSpcServiceId` for the *`iServiceId`* parameter.

See Also `TelIsFunctionSupported()`

TelIsStyServiceAvailable Macro

Purpose Checks if the Security service set (group of related functions) is supported by the phone, driver, and network.

Declared In `TelephonyLib.h`

Prototype `#define TelIsStyServiceAvailable(`*`telDesc`*`)`

Parameters → *`telDesc`*
The telephony file descriptor.

Returns Returns `errNone` if the service set is supported, or returns `telErrFeatureNotSupported` if it is not supported.

Comments Calling this macro is the same as calling the function `TelIsServiceAvailable()` and passing `kTelStyServiceId` for the *`iServiceId`* parameter.

See Also `TelIsFunctionSupported()`

TelMuxChanAllocate Function

Purpose Allocates and opens a phone MUX channel.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelMuxChanAllocate (int32_t telDesc,
   TelMuxChanPtr ioChanP, uint16_t *ioTransIdP)
```

Parameters → *`telDesc`*
The telephony file descriptor.

↔ *`ioChanP`*
A pointer to a `TelMuxChanType` structure. On input, the `type` field specifes the channel type. Upon return, the `chanIdP` field receives the MUX channel ID.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsMuxChanAllocateSupported (`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the structure referenced by *`ioChanP`* remains in memory until the asynchronous call completes.

See Also `TelMuxChanFree()`

TelMuxChanFree Function

Purpose Closes and frees a phone MUX channel.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelMuxChanFree (int32_t telDesc,
   TelMuxChanPtr iChanP, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iChanP`*
A pointer to a `TelMuxChanType` structure. On input, the `chanIdP` specifies the MUX channel ID to free.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsMuxChanFreeSupported (`*`telDesc`*`)`.

See Also `TelMuxChanAllocate()`

TelMuxChanSetId Function

Purpose Selects the current MUX channel.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelMuxChanSetId (int32_t telDesc,
   uint32_t iChanId, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iChanId`*
A pointer to a `TelMuxChanType` structure. On input, the `chanIdP` specifies the MUX channel ID to select.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsMuxChanSetIdSupported (`*`telDesc`*`)`.

TelMuxEnable Function

Purpose Enable or disable the phone MUX.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelMuxEnable (int32_t telDesc,
   uint8_t iStatus, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iStatus`*
Either the `kTelMuxModeDisabled` or `kTelMuxModeEnabled` value described in "MUX Status" on page 117.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsMuxEnableSupported (`*`telDesc`*`)`.

TelNwkAddPreferredOperator Function

Purpose Adds an operator in the list of preferred operators.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelNwkAddPreferredOperator
   (int32_t telDesc,
   TelNwkPreferredOperatorPtr iPreferedOperatorP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iPreferedOperatorP`*
Pointer to a `TelNwkPreferredOperatorType` structure, which contains the operator identifier to add to the preferred operator list and the index in the list where the new operator is to be added. If the `index` field in this structure is set to the value 0xFFFF, then the preferred operator is added at the first free location in the preferred operator list.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsNwkAddPreferredOperatorSupported(`*`telDesc`*`)`.

GSM AT Command AT+CPOL=x (GSM 07.07)

See Also `TelNwkDeletePreferredOperator()`, `TelNwkGetPreferredOperators()`

TelNwkCancelUssd Function

Purpose Cancels an Unstructured Supplementary Service Data (USSD) session.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelNwkCancelUssd (int32_t telDesc,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsNwkCancelUssdSupported(`*`telDesc`*`)`.

GSM AT Command AT+CUSD=2

See Also `TelNwkReceiveUssd()`, `TelNwkSendUssd()`

TelNwkCheckUssd Function

Purpose Checks if a given string is compliant to the USSD requirement.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelNwkCheckUssd (int32_t telDesc,
   TelNwkUssdPtr iUssdP, uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
The telephony file descriptor.

→ *iUssdP*
Pointer to a `TelNwkUssdType` structure.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsNwkCheckUssdSupported(`*`telDesc`*`)`.

See Also `TelNwkReceiveUssd()`, `TelNwkSendUssd()`

TelNwkDeletePreferredOperator Function

Purpose Deletes a preferred operator from the list of preferred operators.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelNwkDeletePreferredOperator
   (int32_t telDesc, uint16_t iIndex,
   uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
The telephony file descriptor.

→ *iIndex*
Index of the preferred operator to delete from the list.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsNwkDeletePreferredOperatorSupported(`*`telDesc`*`)`.

GSM AT Command AT+CPOL=x (GSM 07.07)

See Also `TelNwkAddPreferredOperator()`, `TelNwkGetPreferredOperators()`

TelNwkGetLocation Function

Purpose Gets information about the location of the phone.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelNwkGetLocation (int32_t telDesc,
    TelNwkLocationPtr ioLocationP,
    uint16_t *ioTransIdP)
```

Parameters
→ *telDesc*
The telephony file descriptor.

↔ *ioLocationP*
Pointer to a `TelNwkLocationType` structure.

On input, the `areaCodeSize` field of this structure specifies the allocated size of the `areaCodeP` buffer, and the `cellIdSize` field of this structure specifies the allocated size of the `cellIdP` buffer.

Upon return, the `areaCodeP` buffer contains the area code string and the `areaCodeSize` field specifies the size of the string. And the `cellIdP` buffer contains the cell ID string and the `cellIdSize` field specifies the size of the string.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsNwkGetLocationSupported(`*`telDesc`*`)`.

If the `locationP` buffer is too small to contain the complete location information, the location information is truncated (and ends with the null terminated character) and the function returns the `telErrBufferSize` error. The `locationSize` field will contain the size needed to retrieve the complete location information.

When using this function asynchronously, you must ensure that the structure referenced by *`ioLocationP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CREG? (GSM 07.07)

TelNwkGetOperator Function

Purpose Gets information about the current operator.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelNwkGetOperator (int32_t telDesc,
    TelNwkOperatorPtr ioOperatorP,
    uint16_t *ioTransIdP)
```

Parameters → *`telDesc`*
The telephony file descriptor.

↔ *`ioOperatorP`*
Pointer to a `TelNwkOperatorType` structure that stores information about the operator.

On input, the `nameSize` field of this structure specifies the allocated size of the `nameP` buffer.

Upon return, the `nameP` buffer contains the name string, and the `nameSize` field specifies the size of the name string.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsNwkGetOperatorSupported(`*`telDesc`*`)`.

If the `nameP` field buffer is too small to contain the complete operator name, the operator name is truncated (and ends with the null terminated character) and this function returns the `telErrBufferSize` error. The `nameSize` field will contain the size needed to retrieve the complete network name.

When using this function asynchronously, you must ensure that the structure referenced by *`ioOperatorP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+COPS? (GSM 07.07)

See Also `TelNwkGetOperators()`, `TelNwkGetPreferredOperators()`, `TelNwkSetOperator()`

TelNwkGetOperators Function

Purpose Gets information about, or the count of, available operators.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelNwkGetOperators (int32_t telDesc,
   TelNwkOperatorsPtr ioOperatorP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioOperatorP`*
Pointer to a `TelNwkOperatorsType` structure that stores the operators information.

On input, the `count` field of this structure contains the size, in elements, of the `listP` array field, and the `listP` field contains an array of `TelNwkOperatorType` structures.

Upon return, each element of the `listP` array contains information about the available operators, and the `count` field contains the number of elements in the array.

On input, if you set the `listP` field to `NULL` and `count` to 0, then this function returns only the count of operators in `count`, and `errNone`. No operator information is returned.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsNwkGetOperatorsSupported(`*`telDesc`*`)`.

If the `listP` array is too small to store the return data, this function returns the `telErrBufferSize` error. The `count` field will contain the size, in elements, needed to retrieve all of the available operators, and the `listP` field will contain only the elements that could fit in the initial array.

When using this function asynchronously, you must ensure that the structure referenced by *`ioOperatorsP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+COPS=? (GSM 07.07)

See Also `TelNwkGetOperator()`, `TelNwkGetPreferredOperators()`, `TelNwkSetOperator()`

TelNwkGetPreferredOperators Function

Purpose Gets the list of preferred operators, or the count of them.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelNwkGetPreferredOperators
    (int32_t telDesc,
    TelNwkPreferredOperatorsPtr ioPreferedOperator
    sP, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioPreferedOperatorsP`*
Pointer to a `TelNwkPreferredOperatorsType` structure.

On input, the `count` field of this structure contains the size, in elements, of the `listP` array field, and the `listP` field contains an array of `TelNwkPreferredOperatorType` structures.

Upon return, each element of the `listP` array contains the operator identifier of a preferred operator and its index, and the `count` field contains the number of elements in the array.

On input, if you set the `listP` field to `NULL` and `count` to 0, then this function returns only the count of preferred operators in `count`, and `errNone`.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsNwkGetPreferredOperatorsSupported(`*`telDesc`*`)`.

If the `listP` array is too small to store the return data, this function returns the `telErrBufferSize` error. The `count` field will contain the size, in elements, needed to retrieve all of the available operators, and the `listP` field will contain only the elements that could fit in the initial array.

When using this function asynchronously, you must ensure that the structure referenced by *`ioPreferedOperatorsP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CPOL? (GSM 07.07)

See Also `TelNwkAddPreferredOperator()`, `TelNwkDeletePreferredOperator()`, `TelNwkGetOperators()`

TelNwkGetProviderId Function

Purpose Gets the international mobile subscriber identity.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelNwkGetProviderId (int32_t telDesc,
   uint32_t *oProviderIdP, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oProviderIdP`*
Pointer to the international mobile subscriber identity number.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsNwkGetProviderIdSupported(`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the structure referenced by *`oProviderIdP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CIMI (GSM 07.07)

TelNwkGetRegistrationMode Function

Purpose Gets the current network registration mode.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelNwkGetRegistrationMode
    (int32_t telDesc, uint8_t *oRegistrationModeP,
    uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
The telephony file descriptor.

← *oRegistrationModeP*
Pointer to the current registration mode. One of the constants described in “Registration Search Modes” on page 124.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsNwkGetRegistrationModeSupported(`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the value referenced by *`oRegistrationModeP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CPOS? (GSM 07.07)

See Also `TelNwkSetRegistration()`

TelNwkGetSignalLevel Function

Purpose Gets the selected network carrier signal level.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelNwkGetSignalLevel (int32_t telDesc,
   uint8_t *oSignalLevelP, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oSignalLevelP`*
Pointer to an indicator of the signal level. See Table 4.1 on page 222 for a list of possible values.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsNwkGetSignalLevelSupported(`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the value referenced by `oSignalLevelP` remains in memory until the asynchronous call completes.

Table 4.1 describes the signal level values returned in `oSignalLevelP`. Signal levels are in decibels per milliwatt (dBm).

Table 4.1 Signal levels returned in `oSignalLevelP`

Value returned	Signal level
0	<= -113 dBm
1	-111 dBm
2 to 30	-109 dBm to -53 dBm
31	>= -51 dBm
99	Unknown or undetectable

GSM AT Command AT+CSQ (GSM 07.07)

TelNwkGetStatus Function

Purpose Gets the status of the current network.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelNwkGetStatus (int32_t telDesc,
   uint8_t *oStatusP, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oStatusP`*
Pointer to the status of the current network. One of the constants described in "Network Status Constants" on page 119.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsNwkGetStatusSupported(`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the value referenced by *`oStatusP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CREG? (GSM 07.07)

TelNwkGetType Function

Purpose Gets the type of the current network.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelNwkGetType (int32_t telDesc,
   uint8_t *oTypeP, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oTypeP`*
Pointer to the type of the current network. One of the constants described in "Network Operator Types" on page 118.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsNwkGetTypeSupported(`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the value referenced by *`oTypeP`* remains in memory until the asynchronous call completes.

TelNwkReceiveUssd Function

Purpose Receives a USSD answer from the network.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelNwkReceiveUssd (int32_t telDesc,
   TelNwkUssdPtr ioUssdP, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioUssdP`*
Pointer to a `TelNwkUssdType` structure containing a buffer allocated to hold the message.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsNwkReceiveUssdSupported(`*`telDesc`*`)`.

See Also `TelNwkCancelUssd()`, `TelNwkSendUssd()`

TelNwkSendUssd Function

Purpose Sends a USSD string to the network.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelNwkSendUssd (int32_t telDesc,
   TelNwkUssdPtr iUssdP, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iUssdP`*
Pointer to a `TelNwkUssdType` structure.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsNwkSendUssdSupported(`*`telDesc`*`)`.

See Also `TelNwkCancelUssd()`, `TelNwkCheckUssd()`, `TelNwkReceiveUssd()`

TelNwkSetOperator Function

Purpose Selects an operator to use from among the set of available operators.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelNwkSetOperator (int32_t telDesc,
   uint32_t iOperatorId, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iOperatorId`*
The operator to select.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsNwkSetOperatorSupported(`*`telDesc`*`)`.

GSM AT Command AT+COPS=x (GSM 07.07)

See Also `TelNwkAddPreferredOperator()`, `TelNwkGetOperators()`, `TelNwkGetPreferredOperators()`

TelNwkSetRegistration Function

Purpose Sets the network registration mode and network operator, if needed.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelNwkSetRegistration (int32_t telDesc,
    TelNwkRegistrationType *iRegistrationP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iRegistrationP`*
Pointer to a `TelNwkRegistrationType` structure.

The `mode` field sets the registration mode.

If the mode is `kTelNwkRegistrationManual` or `kTelNwkRegistrationManualAutomatic`, the `operatorId` field must be set to the operator the user wants to set.

↔ *ioTransIdP*
: If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsNwkSetRegistrationSupported(`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the structure referenced by *`iRegistrationP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CPOS=x,y (GSM 07.07)

See Also `TelNwkGetRegistrationMode()`

TelOemCall Function

Purpose Makes a call to an OEM function.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelOemCall (int32_t telDesc,
   TelOemCallPtr ioOemCallP,
   uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
: The telephony file descriptor.

↔ *ioOemCallP*
: Pointer to a `TelOemCallType` structure that identifies the OEM function to call.

↔ *ioTransIdP*
: If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments Call this function to send a request to an OEM function. The calling function and the OEM function are responsible for coordinating the parameter block that is passed in the `TelOemCallType` structure.

The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsOemCallSupported(`*`telDesc`*`)`.

TelOpen Function

Purpose Opens the Telephony library using the first phone connection profile, initializes telephony services, and activates the Telephony Server.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelOpen (uint32_t iVersnum,
   int32_t *telDescP)
```

Parameters

→ *`iVersnum`*
The version number of the Telephony Manager library for which the application is developed. You can specify the current version of the Telephony Manager library by using the `kTelMgrVersion` constant.

← *`telDescP`*
Pointer to a file descriptor for the Telephony Server that you supply as a parameter to any other telephony functions that you call. On input, you can initialize this parameter with the `kTelInvalidAppId` constant.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error.

Comments You must call this function before calling any other Telephony Manager functions.

See Also `TelClose()`, `TelOpenPhoneProfile()`

TelOpenPhoneProfile Function

Purpose Opens the Telephony Library using a specific Connection Manager phone profile, initializes telephony services, and activates the Telephony Server.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelOpenPhoneProfile (uint32_t iVersnum,
   int32_t *telDescP, uint32_t iProfileId)
```

Parameters

→ *`iVersnum`*
The version number of the Telephony Manager library for which the application is developed. You can specify the current version of the Telephony Manager library by using the `kTelMgrVersion` constant.

← *`telDescP`*
Pointer to a file descriptor for the Telephony Server that you supply as a parameter to any other telephony functions that you call. On input, you can initialize this parameter with the `kTelInvalidAppId` constant.

→ *`iProfileId`*
Pointer to the Connection Manager profile to use.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error.

Comments You must call this function before calling any other Telephony Manager functions.

See Also `TelClose()`, `TelOpen()`

TelPhbAddEntry Function

Purpose Adds an entry to the current phone book.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelPhbAddEntry (int32_t telDesc,
   TelPhbEntryPtr iEntryP, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iEntryP`*
Pointer to a `TelPhbEntryType` structure.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsPhbAddEntrySupported(`*`telDesc`*`)`.

If an entry already exists at the index `iEntryP->phoneIndex`, the old entry is overwritten with the new one.

GSM AT Command AT+CPBW=index1, number, numberType, name (GSM 07.07)

See Also `TelPhbDeleteEntry()`, `TelPhbGetPhonebooks()`

TelPhbDeleteEntry Function

Purpose Deletes an entry in the current phone book.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelPhbDeleteEntry (int32_t telDesc,
   uint16_t iEntryIndex, uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
The telephony file descriptor.

→ *iEntryIndex*
Index of the entry to delete.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsPhbDeleteEntrySupported(`*`telDesc`*`)`.

GSM AT Command AT+CPBW=index1 (GSM 07.07)

See Also `TelPhbAddEntry()`, `TelPhbGetPhonebooks()`

TelPhbGetEntries Function

Purpose Gets entries from the current phone book between two indexes, and the count of entries retrieved.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelPhbGetEntries (int32_t telDesc,
   TelPhbEntriesPtr ioEntriesP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioEntriesP`*
Pointer to a `TelPhbEntriesType` structure. The `firstIndex` and `lastIndex` fields specify the range of phone book entries to return.

If the `entryP` field is `NULL`, this function returns in the `entryCount` field the count of entries found between the indexes specified by the `firstIndex` and `lastIndex` fields; and no actual entries are returned.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsPhbGetEntriesSupported(`*`telDesc`*`)`.

GSM AT Command AT+CPBR=index1, index2 (GSM 07.07)

See Also `TelPhbGetEntry()`, `TelPhbGetPhonebooks()`

TelPhbGetEntry Function

Purpose Gets an entry from the current phone book.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelPhbGetEntry (int32_t telDesc,
   TelPhbEntryPtr ioEntryP, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioEntryP`*
Pointer to a `TelPhbEntryType` structure. On input, the `phoneIndex` field specifies the entry to return.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsPhbGetEntrySupported(`*`telDesc`*`)`.

GSM AT Command AT+CPBR=index1 (GSM 07.07)

See Also `TelPhbAddEntry()`, `TelPhbGetEntries()`, `TelPhbGetPhonebooks()`

TelPhbGetPhonebook Function

Purpose Gets information about the current phone book.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelPhbGetPhonebook (int32_t telDesc,
   TelPhbPhonebookPtr oPhonebookP,
   uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
The telephony file descriptor.

← *oPhonebookP*
Pointer to a `TelPhbPhonebookType` structure that returns information about the current phone book such as its identifier, first index, last index, maximum name size, maximum dial number size, total entry slots, and used entry slots.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsPhbGetPhonebookSupported(`*telDesc*`)`.

GSM AT Command AT+CPBS? and AT+CPBR=? (GSM 07.07)

See Also `TelPhbGetPhonebooks()`, `TelPhbSetPhonebook()`

TelPhbGetPhonebooks Function

Purpose Gets the list of available phone books, or the count of them.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelPhbGetPhonebooks (int32_t telDesc,
   TelPhbPhonebooksPtr ioPhonebooksP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioPhonebooksP`*
Pointer to a `TelPhbPhonebooksType` structure.

On input, the `count` field must be the count of elements in the `idP` buffer. If you set the `idP` field to `NULL`, this function returns in the `count` field the number of available phone books.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsPhbGetPhonebooksSupported(`*`telDesc`*`)`.

GSM AT Command AT+CPBS=? (GSM 07.07)

See Also `TelPhbGetPhonebook()`, `TelPhbSetPhonebook()`

TelPhbSetPhonebook Function

Purpose Sets the current phone book.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelPhbSetPhonebook (int32_t telDesc,
   TelPhbPhonebookPtr ioPhonebookP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioPhonebookP`*
Identifier of the phone book to set as the current one. One of the constants described in "Phone Book Identifiers" on page 124.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsPhbSetPhonebookSupported(`*`telDesc`*`)`.

GSM AT Command AT+CPBS=x (GSM 07.07)

See Also `TelPhbGetPhonebook()`, `TelPhbSetPhonebook()`

TelPowGetBatteryChargeLevel Function

Purpose Gets the current level of the phone battery, as a percentage value.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelPowGetBatteryChargeLevel
    (int32_t telDesc,
    uint8_t *oBatteryChargeLevelP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oBatteryChargeLevelP`*
Pointer to a value that indicates the battery percentage level as an integer value between 0 and 100.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsPowGetBatteryChargeLevelSupported(`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the value referenced by *`oBatteryLevelP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CBC (GSM 07.07)

See Also `TelPowGetBatteryConnectionStatus()`

TelPowGetBatteryConnectionStatus Function

Purpose Gets the status of the phone battery.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelPowGetBatteryConnectionStatus
    (int32_t telDesc,
    uint8_t *oBatteryConnectionStatusP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oBatteryConnectionStatusP`*
Pointer to the battery status value. One of the constants described in "Battery Status Constants" on page 86.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsPowGetBatteryConnectionStatusSupported(`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the value referenced by *`oBatteryConnectionStatusP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CBC (GSM 07.07)

See Also `TelPowGetBatteryChargeLevel()`

TelPowSetPhoneFunctionality Function

Purpose Set the level of functionality of the phone.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelPowSetPhoneFunctionality
   (int32_t telDesc, uint8_t iPhoneFunctionality,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iPhoneFunctionality`*
Specify one of the following values:

`0`
Minimum functionality.

`1`
Full functionality.

`2`
Disable transmit RF circuits only.

`3`
Disable receive RF circuits only.

`4`
Disable both transmit and receive RF circuits.

`5-127`
Reserved for other manufacturer-defined states between minimum and full functionality.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsPowSetPhoneFunctionalitySupported(`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the value referenced by *`oBatteryConnectionStatusP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CFUN=x (GSM 07.07)

See Also `TelPowGetBatteryChargeLevel()`

TelSmsDeleteMessage Function

Purpose Deletes an SMS message from the current store.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelSmsDeleteMessage (int32_t telDesc,
   uint16_t iMessageIndex, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iMessageIndex`*
Index of the message to delete from current store.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSmsDeleteMessageSupported(`*`telDesc`*`)`.

GSM AT Command AT+CMGD=x (GSM 07.05)

See Also `TelSmsGetStorages()`, `TelSmsSetStorage()`

TelSmsGetDataMaxSize Function

Purpose Gets the maximum size of data for a single SMS message.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelSmsGetDataMaxSize (int32_t telDesc,
   size_t *oDataMaxSizeP, uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
The telephony file descriptor.

← *oDataMaxSizeP*
Pointer to the maximum size of the data.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` function must have been called.

You can check if this function is supported by using the macro `TelIsSmsGetDataMaxSizeSupported(`*telDesc*`)`.

TelSmsGetStorage Function

Purpose Gets information about an SMS store.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelSmsGetStorage (int32_t telDesc,
   TelSmsStoragePtr oStorageP,
   uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
The telephony file descriptor.

← *oStorageP*
Pointer to a `TelSmsStorageType` structure.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSmsGetStorageSupported(`*`telDesc`*`)`.

GSM AT Command AT+CPMS? (GSM 07.05)

See Also `TelSmsGetStorages()`, `TelSmsSetStorage()`

TelSmsGetStorages Function

Purpose Gets the list of SMS stores available on the phone, or the count of them.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelSmsGetStorages (int32_t telDesc,
   TelSmsStoragesPtr ioStoragesP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioStoragesP`*
Pointer to a `TelSmsStoragesType` structure.

On input, if you set the `idP` field to `NULL` and `count` to 0, then this function returns only the count of SMS stores in `count`, and `errNone`. No store information is returned.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSmsGetStoragesSupported(`*`telDesc`*`)`.

GSM AT Command AT+CPMS=? (GSM 07.05)

See Also `TelSmsGetStorage()`, `TelSmsSetStorage()`

TelSmsGetUniquePartId Function

Purpose Gets a unique part identifier for a multipart SMS message.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelSmsGetUniquePartId (int32_t telDesc,
   uint16_t *oUniquePartIdP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oUniquePartIdP`*
Pointer to a unique part identifier.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSmsGetUniquePartIdSupported(`*`telDesc`*`)`.

See Also `TelSmsGetStorage()`, `TelSmsSetStorage()`

TelSmsReadMessage Function

Purpose Gets an SMS message from the current store.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelSmsReadMessage (int32_t telDesc,
   TelSmsMessagePtr ioMessageP,
   uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
The telephony file descriptor.

↔ *ioMessageP*
Pointer to a `TelSmsMessageType` structure.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSmsReadMessageSupported(`*telDesc*`)`.

GSM AT Command AT+CMGR=xxx (GSM 07.05)

See Also `TelSmsDeleteMessage()`, `TelSmsReadMessages()`

TelSmsReadMessages Function

Purpose Gets a list of SMS messages in the current store, or a count of them.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelSmsReadMessages (int32_t telDesc,
   TelSmsMessagesPtr ioMessagesP,
   uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
The telephony file descriptor.

↔ *ioMessagesP*
Pointer to a `TelSmsMessagesType` structure.

On input, if you set the `listP` field to `NULL`, then this function returns only the count of SMS message in `count`. No other message information is returned.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSmsReadMessagesSupported(`*`telDesc`*`)`.

GSM AT Command AT+CMGL=xxx (GSM 07.05)

See Also `TelSmsDeleteMessage()`, `TelSmsReadMessage()`

TelSmsSendMessage Function

Purpose Sends an SMS message.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelSmsSendMessage (int32_t telDesc,
   TelSmsMessagePtr ioMessageP,
   uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
The telephony file descriptor.

↔ *ioMessageP*
Pointer to a `TelSmsMessageType` structure containing the message to send.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSmsSendMessageSupported(`*`telDesc`*`)`.

GSM AT Command AT+CMGS=xxx (GSM 07.05)

See Also `TelSmsGetUniquePartId()`

TelSmsSetStorage Function

Purpose Sets an SMS store as the current store.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelSmsSetStorage (int32_t telDesc,
    uint16_t iStorageId, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iStorageId`*
Identifier of the SMS store to set as the current one. Specify one of the constants described in “SMS Storage Locations” on page 133.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSmsSetStorageSupported(`*`telDesc`*`)`.

GSM AT Command AT+CPMS=x (GSM 07.05)

See Also `TelSmsGetUniquePartId()`

TelSndGetMuteStatus Function

Purpose Gets the status of the microphone muting feature for voice calls.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelSndGetMuteStatus (int32_t telDesc,
   uint8_t *oMuteStatusP, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oMuteStatusP`*
Pointer to the mute status value. One of the constants described in "Mute Status Constants" on page 116.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSndGetMuteStatusSupported(`*`telDesc`*`)`.

GSM AT Command AT+CMUT? (GSM 07.07)

See Also `TelSndSetMuteStatus()`

TelSndSetMuteStatus Function

Purpose Sets the status of the microphone muting feature for voice calls.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelSndSetMuteStatus (int32_t telDesc,
   uint8_t iMuteStatus, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iMuteStatus`*
The mute status value, which is one of the constants described in "Mute Status Constants" on page 116.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSndSetMuteStatusSupported(`*`telDesc`*`)`.

GSM AT Command AT+CMUT=x (GSM 07.07)

See Also `TelSndSetMuteStatus()`

TelSpcAcceptCall Function

Purpose Accepts an incoming voice call.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelSpcAcceptCall (int32_t telDesc,
   TelSpcCallPtr oCallP, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *oCallP*
Pointer to a `TelSpcCallType` structure that contains information about the call.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSpcAcceptCallSupported(`*`telDesc`*`)`.

GSM AT Command ATA (GSM 07.07)

See Also `TelSpcHoldActiveCalls()`, `TelSpcReleaseCall()`

TelSpcAddHeldCall Function

Purpose Adds a held call to the conversation.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelSpcAddHeldCall (int32_t telDesc,
   uint16_t *ioTransIdP)
```

Parameters

→ *telDesc*
The telephony file descriptor.

↔ *ioTransIdP*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSpcAddHeldCallSupported(`*`telDesc`*`)`.

GSM AT Command AT+CHLD=3 (GSM 07.07)

See Also `TelSpcHoldActiveCalls()`, `TelSpcReleaseCall()`

TelSpcGetCall Function

Purpose Gets information about a specific call.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelSpcGetCall (int32_t telDesc,
   TelSpcCallPtr ioCallP, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*

The telephony file descriptor.

↔ *`ioCallP`*

Pointer to a `TelSpcCallType` structure, which contains the specific call upon return.

On input, the `callId` field of this structure must be set to identify the call to get information about.

↔ *`ioTransIdP`*

If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSpcGetCallSupported(`*`telDesc`*`)`.

GSM AT Command AT+CLCC (GSM 07.07)

See Also `TelSpcGetCalls()`

TelSpcGetCalls Function

Purpose Gets a list of current calls or the count of them.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelSpcGetCalls (int32_t telDesc,
   TelSpcCallsPtr ioCallsP, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioCallsP`*
Pointer to a `TelSpcCallsType` structure, which contains the current calls list upon return.

On input, if you set the `listP` field to `NULL` and `count` to 0, then this function returns only the count of current calls in `count`, and `errNone`. No other call information is returned.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSpcGetCallsSupported(`*`telDesc`*`)`.

GSM AT Command AT+CLCC (GSM 07.07)

See Also `TelSpcGetCall()`

TelSpcGetToneDuration Function

Purpose Gets the current setting for the length of tones played by the function `TelSpcPlayTone()`.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelSpcGetToneDuration (int32_t telDesc,
    uint16_t *ioToneDurationP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioToneDurationP`*
Pointer to the length of the tones, in tens of milliseconds (for example, the value 4 means 40 milliseconds).

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSpcGetToneDurationSupported(`*`telDesc`*`)`.

GSM AT Command AT+VTD? (GSM 07.07)

See Also `TelSpcSetToneDuration()`

TelSpcGetToneDurationRange Function

Purpose Gets the minimum and maximum length of tones that can be played by the function `TelSpcPlayTone()`.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelSpcGetToneDurationRange
   (int32_t telDesc,
   TelSpcToneDurationRangePtr ioToneDurationRange
   P, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioToneDurationRangeP`*
Pointer to a `TelSpcToneDurationRangeType` structure.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSpcGetToneDurationRangeSupported(`*`telDesc`*`)`.

GSM AT Command AT+VTD=? (GSM 07.07)

See Also `TelSpcSetToneDuration()`

TelSpcHoldActiveCalls Function

Purpose Places all active calls, if any, on hold and accept another (incoming, waiting, or held) call, if any.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelSpcHoldActiveCalls (int32_t telDesc,
   uint16_t *ioTransIdP)
```

Parameters → *`telDesc`*
The telephony file descriptor.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSpcHoldActiveCallsSupported(`*`telDesc`*`)`.

If a call is on hold and you have an active call, this function swaps them; that is, it puts the active call on hold and makes the held call the active call.

GSM AT Command AT+CHLD=2 (GSM 07.07)

See Also `TelSpcAcceptCall()`, `TelSpcReleaseCall()`

TelSpcInitiateCall Function

Purpose Initiates a voice call.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelSpcInitiateCall (int32_t telDesc,
   TelSpcCallPtr ioCallP, uint16_t *ioTransIdP)
```

Parameters → *`telDesc`*
The telephony file descriptor.

↔ *`ioCallP`*
Pointer to a `TelSpcCallType` structure that must contain the number to dial.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSpcInitiateCallSupported(`*`telDesc`*`)`.

GSM AT Command ATDxxxx; (GSM 07.07)

See Also `TelSpcAcceptCall()`, `TelSpcReleaseCall()`

TelSpcPlayTone Function

Purpose Sends DTMF tones.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelSpcPlayTone (int32_t telDesc,
   char iTone, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iTone`*
A single ASCII character in the set of 0-9, #, *, and A-D to send.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSpcPlayToneSupported(`*`telDesc`*`)`.

The duration of each tone is set by the function `TelSpcSetToneDuration()`.

GSM AT Command AT+VTS=xxxx (GSM 07.07)

See Also `TelSpcGetToneDuration()`, `TelSpcGetToneDurationRange()`

TelSpcPrivateCall Function

Purpose Places all active calls on hold except a specific call.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelSpcPrivateCall (int32_t telDesc,
    uint8_t iCallId, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iCallId`*
Call identifier of the call that you want to continue to be active.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSpcPrivateCallSupported(`*`telDesc`*`)`.

GSM AT Command AT+CHLD=2X (GSM 07.07)

TelSpcReleaseCall Function

Purpose Rejects or releases a specific call or releases all active calls, all held calls, or all calls.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelSpcReleaseCall (int32_t telDesc,
   uint8_t iCallId, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iCallId`*
Identifier of a specific call to reject or release, or a constant that indicates what kind of calls to release, from the group of constants described in "Call Release Types" on page 87.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSpcReleaseCallSupported(`*`telDesc`*`)`.

GSM AT Command ATH, AT+CHLD=1 (GSM 07.07)

See Also `TelSpcAcceptCall()`, `TelSpcHoldActiveCalls()`

TelSpcSetToneDuration Function

Purpose Sets the duration of tones played by the function `TelSpcPlayTone()`.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelSpcSetToneDuration (int32_t telDesc,
   uint16_t iToneDuration, uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iToneDuration`*
Duration of tones, in tens of milliseconds (for example, the value 4 means 40 milliseconds).

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsSpcSetToneDurationSupported(`*`telDesc`*`)`.

GSM AT Command AT+VTD=x (GSM 07.07)

See Also `TelSpcGetToneDuration()`, `TelSpcGetToneDurationRange()`

TelStyChangeFacilityPassword Function

Purpose Changes the password of a facility.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelStyChangeFacilityPassword
    (int32_t telDesc,
    TelStyFacilityPasswordPtr iFacilityPasswordP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iFacilityPasswordP`*
Pointer to a `TelStyFacilityPasswordType` structure containing the new password. Note that this structure must also contain the current password (`passwordP`) and the type of facility (`type`) whose password is being changed.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsStyChangeFacilityPasswordSupported(`*`telDesc`*`)`.

GSM AT Command AT+CPWD=*`facility`*, *`oldPassword`*, *`newPassword`* (GSM 07.07)

See Also `TelStyGetFacilities()`

TelStyEnterAuthentication Function

Purpose Displays a user interface to let the user enter the password that the phone is currently waiting for.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelStyEnterAuthentication
    (int32_t telDesc,
    TelStyAuthenticationPtr iAuthenticationP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iAuthenticationP`*
Pointer to a `TelStyAuthenticationType` structure containing the password. The `type` field must contain the type of authentication that the phone is waiting for.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsStyEnterAuthenticationSupported(`*`telDesc`*`)`.

This function displays its own user interface to ask the user for the password, and in some cases for a new password.

You can use this function only with GSM networks.

GSM AT Command AT+CPIN=x (GSM 07.07)

See Also `TelStyGetAuthenticationStatus()`

TelStyGetAuthenticationStatus Function

Purpose Gets the type of authentication password, if any, that the phone is waiting for before it can be operated.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelStyGetAuthenticationStatus
    (int32_t telDesc, uint8_t *oAuthenticationP,
    uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

← *`oAuthenticationP`*
Pointer to the authentication type needed, which is one of the constants described in "Authentication Types" on page 84.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsStyGetAuthenticationStatusSupported(`*`telDesc`*`)`.

When using this function asynchronously, you must ensure that the value referenced by *`oAuthenticationP`* remains in memory until the asynchronous call completes.

GSM AT Command AT+CPIN? (GSM 07.07)

See Also `TelStyEnterAuthentication()`

TelStyGetFacilities Function

Purpose Gets a list of facility types supported by the phone, or the count of them.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelStyGetFacilities (int32_t telDesc,
   TelStyFacilitiesPtr ioFacilitiesP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioFacilitiesP`*
Pointer to a `TelStyFacilitiesType` structure. Upon return, the `idP` field contains the list of facilities supported.

On input, if you set the `idP` field to `NULL` and `count` to 0, then this function returns only the count of facilities in `count`, and `errNone`. No other facility information is returned.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsStyGetFacilitiesSupported(`*`telDesc`*`)`.

GSM AT Command AT+CLCK=? (GSM 07.07)

See Also `TelStyGetFacility()`

TelStyGetFacility Function

Purpose Gets the status of a facility.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelStyGetFacility (int32_t telDesc,
   TelStyFacilityPtr iFacilityP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`iFacilityP`*
Pointer to a `TelStyFacilityType` structure.

On input, the `type` field must contain one of the constants described in "Security Facility Types" on page 125.

Upon return, the `status` field contains the status of the facility, which is one of the constants described in "Security Facility Status Constants" on page 125.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsStyGetFacilitySupported(`*`telDesc`*`)`.

GSM AT Command AT+CLCK=x (GSM 07.07)

See Also `TelStyGetFacilities()`

TelStyLockFacility Function

Purpose Locks a facility.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelStyLockFacility (int32_t telDesc,
   TelStyFacilityPtr iFacilityP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iFacilityP`*
Pointer to a `TelStyFacilityType` structure. The `type` field must contain one of the constants described in "Security Facility Types" on page 125.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsStyLockFacilitySupported(`*`telDesc`*`)`.

GSM AT Command AT+CLCK=x (GSM 07.07)

See Also `TelStyUnlockFacility()`

TelStyUnlockFacility Function

Purpose Unlocks a facility.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelStyUnlockFacility (int32_t telDesc,
   TelStyFacilityPtr iFacilityP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

→ *`iFacilityP`*
Pointer to a `TelStyFacilityType` structure. The `type` field must contain one of the constants described in "Security Facility Types" on page 125.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` and `TelCncOpen()` functions must have been called.

You can check if this function is supported by using the macro `TelIsStyUnlockFacilitySupported(`*`telDesc`*`)`.

GSM AT Command AT+CLCK=x (GSM 07.07)

See Also `TelStyLockFacility()`

TelTestPhoneDriver Function

Purpose Checks the connection with the phone, and if the phone is supported by the driver.

Declared In `TelephonyLib.h`

Prototype

```
status_t TelTestPhoneDriver (int32_t telDesc,
   TelInfIdentificationPtr ioNameP,
   uint16_t *ioTransIdP)
```

Parameters

→ *`telDesc`*
The telephony file descriptor.

↔ *`ioNameP`*
Pointer to a `TelInfIdentificationType` structure to get the model and the brand of the phone. This parameter is optional, so you can set it to `NULL` if you do not want to get this information.

On input you need to specify only the `size` and `valueP` fields.

↔ *`ioTransIdP`*
If `NULL` on input, the function is executed in synchronous mode. Otherwise the function is executed in asynchronous mode, and the transaction identifier is returned here.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error. In asynchronous mode, the result is returned through a `kTelTelephonyEvent`.

Comments The `TelOpen()` function must have been called.

You can check if this function is supported by using the macro `TelIsTestPhoneDriverSupported(`*`telDesc`*`)`.

TelUiManageError Function

Purpose Manages an error by displaying a dialog with the appropriate text message and the appropriate button.

Declared In `TelephonyLib.h`

Prototype
```
status_t TelUiManageError (status_t iError,
   Boolean *ioRetryP)
```

Parameters

→ *`iError`*
The error code to manage.

↔ *`ioRetryP`*
Boolean to indicate if the user wants to retry or not.

Returns Returns `errNone` if the function was successful, otherwise an appropriate Telephony Manager error.

Part II
SMS Exchange Library

The SMS Exchange Library makes it possible to send Short Message Service (SMS) messages to other devices, and to receive SMS messages sent to the Palm Powered™ device.

5

SMS Exchange Library Reference

This chapter describes the SMS Exchange Library API declared in the header file `SmsLib.h`. It discusses the following topics:

- SMS Exchange Library Data Structures
- SMS Exchange Library Constants

You interact with the SMS Exchange Library using the Exchange Manager APIs described in Chapter 5, "Exchange Manager Reference," of the book *Exploring Palm OS: High-Level Communications*. For further information on using Exchange Manager, see Chapter 4, "Object Exchange," of the book *Exploring Palm OS: High-Level Communications*.

Note that the SMS Exchange Library does not implement the `ExgGet()` function.

SMS Exchange Library Data Structures

SmsParamsType Struct

Purpose Identifies information specific to the SMS Exchange Library.

Declared In `SmsLib.h`

Prototype

```
typedef struct SmsParamsType {
   uint32_t creator;
   char *extension;
   char *mimeTypes;
   uint32_t appCreator;
   TelSmsMessageType message;
   uint16_t requestType;
   uint16_t storageId;
   Boolean leaveOnPhone;
   Boolean forceSlotMode;
   Boolean ignoreDefaultValue;
   uint8_t padding[1];
} SmsParamsType, *SmsParamsPtr
```

Fields

`creator`
Creator ID of the SMS Exchange Library. Always set this to `sysFileCSmsLib`.

`extension`
If the SMS message has an attachment, this field specifies the attachment name. Do not set this field directly; the SMS Exchange Library sets it if necessary. See the `appCreator` field description for details.

`mimeTypes`
If the SMS message has an attachment, this field specifies the MIME type of the attachment. Do not set this field directly; the SMS Exchange Library sets it if necessary. See the `appCreator` field description for details.

`appCreator`
The creator ID of the target application for the attachment to the SMS message. Do not set this field directly; the SMS Exchange Library sets it if necessary.

When the SMS Exchange Library receives a message with an attachment, it unwraps the message and attempts to deliver the attachment directly to an application that is registered to receive it. If no application is registered to receive unwrapped attachments of that type, the SMS Exchange Library sends the entire SMS message, and it sets the `extension`, `mimeTypes`, and `appCreator` fields in this structure. The SMS application can use this information to have the Exchange Manager deliver the attachment to the appropriate application using the Local Exchange Library.

`message`
A `TelSmsMessageType` structure holding the message that was sent. Do not set this field directly; the SMS Exchange Library should set it.

`requestType`
One of the constants described in "SMS Message Types" on page 279. These constants can be ORed together.

`storageId`
: Used internally to retrieve a specific message.

`leaveOnPhone`
: Set this to `true` on input to leave received messages on the phone; set to `false` (default) to delete messages from the phone once they are received. Received messages are stored in the SMS Messenger inbox.

`forceSlotMode`
: Set this to `true` on input to force the parsing method to slot mode; set to `false` to use block mode (default). In slot mode, SMS messages are read one at a time and in block mode, they are read all in one block.

`ignoreDefaultValue`
: Set this to `true` on input to ignore the default values for validity period and SMS delivery request that are saved in the preferences; instead use the parameters in the structure. Set to `false` to use the values in the preferences.

`padding`
: Padding byte.

Comments The `socketRef` field of the `ExgSocketType` structure is set to this structure when you send or receive data using the SMS Exchange Library. You need to create this structure and assign it to the `socketRef` field only if you have an SMS message to send and want to use non-default values for some of the fields; otherwise, the SMS Exchange Library creates this structure for you and provides default values.

When receiving an SMS message, the application is sub-launched by the Exchange Manager using the `sysAppLaunchCmdExgReceiveData` launch code. A flattened `SMSParamsType` structure is defined in the `ExgSocketType.socketRef` field and its size is in the `ExgSocketType.socketRefSize` field. You must unflatten the `SMSParamsType` structure before using it. This can be done with the code shown in the Example section.

Example This example shows how to unflatten the `SMSParamsType` structure when receiving an SMS message.

```
// Accept will open a progress dialog and wait for your receive commands
err = ExgAccept(exgSocketP);
if (err >= errNone)
{
  // Get all application specific info, unflatten it first
  if ((err = PrvSmsExgUnflattenSmsParamsType((uint8_t*)exgSocketP->socketRef,
     (size_t) exgSocketP->socketRefSize, &smsParam)) != errNone)
    goto Exit;

  err = exgErrBadParam;

  messageType = kSmsMessageTypeIn;

  switch(smsParam.message.messageType)
  {
    case kTelSmsMessageTypeDelivered:
      if (smsParam.message.message.deliver.networkParams.gsm.messageClass ==
             kTelSmsClass0)
        messageType = kSmsMessageTypeFlash;
      break;

    case kTelSmsMessageTypeReport:
      messageType = kSmsMessageTypeReport;
      break;
  }
  // your code continues here . . .
}

// The unflatten code should be:
/********************************************************************************
 *
 * Function:    PrvSmsExgUnflattenSmsParamsType
 *
 * Description: Unflatten an input buffer into an SmsParamsType structure
 *
 * Parameters:
 * bufferP - input: pointer to a flat buffer.
 * bufferSize - input: the size of the flat buffer.
 * smsParamsP - output: pointer to a filled SmsParamsType structure.
 *
 * Returned:    error if any.
********************************************************************************/
static status_t PrvSmsExgUnflattenSmsParamsType(uint8_t* bufferP, size_t
bufferSize, SmsParamsType* smsParamsP )
{
  uint8_t* offsetP = bufferP;
  int32_t lenRemaining = bufferSize;

// Initialize the structure
  memset(smsParamsP, 0, sizeof(SmsParamsType));

  // Check the tag
  if (lenRemaining < (int32_t)sizeof(uint32_t))
  {
    uint32_t value;
    memmove(&value, offsetP, sizeof(uint32_t));
    if (value != sysFileCSmsLib)
      return exgErrBadParam;
  }
  offsetP += sizeof(uint32_t);
  lenRemaining -= sizeof(uint32_t);

  // Get the structure
  if (lenRemaining >= (int32_t)sizeof(SmsParamsType))
  {
    memmove(smsParamsP, offsetP, sizeof(SmsParamsType));
  }
  offsetP += sizeof(SmsParamsType);
  lenRemaining -= sizeof(SmsParamsType);

  // Get the address 1
  if (smsParamsP->message.address1.size && (lenRemaining >=
      (int32_t)smsParamsP->message.address1.size))
  {
    // Allocate address 1 memory
    if ((smsParamsP->message.address1.numberP =
        (char*)MemPtrNew(smsParamsP->message.address1.size)) == NULL)
      goto cleanup;

    memmove(smsParamsP->message.address1.numberP, offsetP,
      smsParamsP->message.address1.size);
  }
  offsetP += smsParamsP->message.address1.size;
  lenRemaining -= smsParamsP->message.address1.size;

  // Get the address 2
  if (smsParamsP->message.address2.size && (lenRemaining >=
        (int32_t)smsParamsP->message.address2.size))
  {
    // Allocate address 2 memory
    if ((smsParamsP->message.address2.numberP =
        (char*)MemPtrNew(smsParamsP->message.address2.size)) == NULL)
    goto cleanup;
    memmove(smsParamsP->message.address2.numberP, offsetP,
      smsParamsP->message.address2.size);
  }
  offsetP += smsParamsP->message.address2.size;
  lenRemaining -= smsParamsP->message.address2.size;

  // Get the extensions
  if (smsParamsP->message.extensionCount && (lenRemaining >= (int32_t)
      (sizeof(TelSmsExtensionType) * smsParamsP->message.extensionCount)))
  {
    // Allocate extensions memory
    if ((smsParamsP->message.extensionP = (TelSmsExtensionType*)MemPtrNew((size_t)
      (sizeof(TelSmsExtensionType) * smsParamsP->message.extensionCount))) == NULL)
      goto cleanup;

    memmove(smsParamsP->message.extensionP, offsetP,
      sizeof(TelSmsExtensionType) * smsParamsP->message.extensionCount);
  }
  offsetP +=  sizeof(TelSmsExtensionType) * smsParamsP->message.extensionCount;
  lenRemaining -= (int32_t)sizeof(TelSmsExtensionType) *
      smsParamsP->message.extensionCount;

  // Get the extension string
  if ( smsParamsP->extension && lenRemaining)
  {
    // Allocate extension string memory
    if ((smsParamsP->extension = (char*)MemPtrNew((size_t)
        (strlen((char*)offsetP) + 1))) == NULL)
      goto cleanup;

    memmove(smsParamsP->extension, offsetP, strlen((char*)offsetP) + 1);
    lenRemaining -= (int32_t)strlen((char*)offsetP) + 1;
    offsetP +=  strlen((char*)offsetP) + 1;
  }

  // Get the mimeTypes string
  if ( smsParamsP->mimeTypes && lenRemaining)
  {
    // Allocate extension string memory
    if ((smsParamsP->mimeTypes = (char*)MemPtrNew((size_t)
        (strlen((char*)offsetP) + 1))) == NULL)
      goto cleanup;

    memmove(smsParamsP->mimeTypes, offsetP, strlen((char*)offsetP) + 1);
    lenRemaining -= (int32_t)strlen((char*)offsetP) + 1;
    offsetP +=  strlen((char*)offsetP) + 1;
  }
  if (lenRemaining >= 0)
    return errNone;
  return exgErrBadParam;

cleanup:
  // Free only what was really allocated
  if (smsParamsP->message.address1.size && smsParamsP->message.address1.numberP)
  {
    MemPtrFree(smsParamsP->message.address1.numberP);
    if (smsParamsP->message.address2.size && smsParamsP->message.address2.numberP)
    {
      MemPtrFree(smsParamsP->message.address2.numberP);
      if (smsParamsP->message.extensionCount && smsParamsP->message.extensionP)
      {
        MemPtrFree(smsParamsP->message.extensionP);
        if (smsParamsP->extension)
        {
          MemPtrFree(smsParamsP->extension);
          if (smsParamsP->mimeTypes)
          {
            MemPtrFree(smsParamsP->mimeTypes);
          }
        }
      }
    }
  }

  return exgMemError;
}
```

SmsPrefType Struct

Purpose Defines the SMS Exchange Library preferences for sending and receiving SMS messages. Applications can use the `ExgControl()` function to get, set, or display these preferences to the user.

Declared In `SmsLib.h`

Prototype

```
typedef struct SmsPrefType {
   uint32_t validity;
   uint16_t warnOver;
   Boolean leaveOnPhone;
   Boolean report;
   Boolean autoSmsCenter;
   uint8_t padding[3];
   char smsCenterNumberP[kTelPhoneNumberMaxLen];
} SmsPrefType, *SmsPrefPtr
```

Fields

`validity`
: Number of seconds before the message expires. If the message cannot be delivered to the recipient, the service center repeatedly attempts to deliver the message until it expires. The default is one day (86400 seconds).

`warnOver`
: Number of parts a user can send without confirmation. If the user attempts to send a message with more than this number of parts, an alert is displayed, and the user can choose to send the message anyway. The default is 3 parts. (If the user attempts to send a message with more than 3 parts, an alert is displayed.)

`leaveOnPhone`
: If `true`, any incoming messages retrieved from a phone remain on the phone as well. If `false`, the messages are deleted from the phone's inbox.

`report`
: If `true`, the user receives confirmation that an SMS message was delivered.

`autoSmsCenter`
: If `true`, don't use the value stored in the `smscNumberP` field.

`padding`
: Padding bytes.

`smsCenterNumberP`
: A pointer to the message center to be used. If `NULL` or the empty string, the SMS message center set by the phone is used.

SMS Exchange Library Constants

SMS Control Constants

Purpose These constants are passed as the *`operation`* parameter to the `ExgControl()` function. The `ExgControl()` function is a way to communicate directly with the SMS Exchange Library.

Declared In `SmsLib.h`

Constants

```
#define exgLibSmsPrefGetOp (exgLibCtlSpecificOp | 1)

#define exgLibSmsPrefGetDefaultOp
  (exgLibCtlSpecificOp | 2)

#define exgLibSmsPrefSetOp (exgLibCtlSpecificOp | 3)

#define exgLibSmsPrefDisplayOp
  (exgLibCtlSpecificOp | 4)

#define exgLibSmsIncompleteGetCountOp
  (exgLibCtlSpecificOp | 5)

#define exgLibSmsIncompleteDeleteOp
  (exgLibCtlSpecificOp | 6)
```

Comments The following table lists the operation constant, the type of data that should be passed as the *`valueP`* parameter to `ExgControl()`, and what the SMS Exchange Library does in response.

Table 5.1 ExgControl operations for SMS library

Operation	value Data Type	Description
`exgLibSmsPrefGetOp`	`SmsPrefType`	Returns a pointer to the SMS Exchange Libraries preferences in *`valueP`*, and creates the preferences and sets them to the default values if they do not exist.
`exgLibSmsPrefGetDefaultOp`	`SmsPrefType`	Returns the default values for the SMS Exchange Library preferences.

Table 5.1 ExgControl operations for SMS library *(continued)*

Operation	value Data Type	Description
`exgLibSmsPrefSetOp`	`SmsPrefType`	Sets the SMS Exchange Library preferences to the values passed in *`valueP`*.
`exgLibSmsPrefDisplayOp`	One of the "Network Operator Types" on page 118. Note that only `kTelNwkTypeGsmGprs` is supported in Palm OS® Cobalt.	Displays a form that allows the user to set the SMS preferences.
`exgLibSmsIncompleteGetCountOp`	`uint16_t`. Output only.	Gets the number of incomplete messages currently stored in the SMS Exchange Library. The library stores message parts as it receives them. When it has received all of the parts, it reassembles the message and delivers it. This operation tells how many messages are currently under assembly.
`exgLibSmsIncompleteDeleteOp`	`uint16_t`. Input only.	Deletes the incomplete message with the ID passed in *`valueP`*. Pass -1 to delete all incomplete messages.

SMS Extension Types

Purpose Defines Exchange Manager extensions that an application can register for to receive SMS messages of those types.

Declared In `SmsLib.h`

Constants `#define kSmsRegExtensionTypeEMailInd "ewi"`
Email waiting indication.

`#define kSmsRegExtensionTypeFaxInd "fwi"`
Fax waiting indication.

`#define kSmsRegExtensionTypeFlash "fhs"`
Flash SMS message

`#define kSmsRegExtensionTypeMessage "sms"`
SMS message.

`#define kSmsRegExtensionTypeOtherInd "owi"`
Other message waiting indication.

`#define kSmsRegExtensionTypeReport "rps"`
Report message.

`#define kSmsRegExtensionTypeVoiceMailInd "vwi"`
Voice mail waiting indication.

SMS Extension Type Length

Purpose Defines the length of an extension type constant value.

Declared In `SmsLib.h`

Constants `#define kSmsRegExtensionTypeLength 3`
The length of an extension type constant value.

SMS Message Types

Purpose The SMS message type constants identify the type of message being sent. They are used in the `requestType` field of the `SmsParamsType` structure and can be combined with an OR operation.

Declared In `SmsLib.h`

Constants `#define kSmsMessageTypeIn ((uint16_t) 0x0001)`
Classic message.

`#define kSmsMessageTypeReport ((uint16_t) 0x0002)`
Report message.

`#define kSmsMessageTypePart ((uint16_t) 0x0004)`
Internal use only; do not use. For multipart SMS reassembly.

`#define kSmsMessageTypeMultipart ((uint16_t) 0x0008)`
An incomplete multipart message.

`#define kSmsMessageTypeFlash ((uint16_t) 0x0010)`
Flash message.

`#define kSmsMessageTypeIndication ((uint16_t) 0x0020)`
Indication that signals a message is waiting.

`#define kSmsMessageTypeIncoming ((uint16_t) 0xFFFF)`
Internal use only; do not use. For getting an incoming message.

SMS Scheme

Purpose Defines the Exchange Manager URL scheme for the SMS Exchange Library.

Declared In `SmsLib.h`

Constants `#define kSmsScheme "_sms"`
The URL scheme for the SMS Exchange Library.

Index