Winwap Technologies Oy

SMS Client SDK

SMS Client SDK version: 1.0
Document dated: 30 June 2005, Rev A8

Contents

1. SMS CLIENT SDK OVERVIEW 4
1.1. PREREQUISITES TO USE THE SDK 4
1.2. NOTICE OF CONFIDENTIALITY ..... **ERROR! BOOKMARK NOT DEFINED.**
1.3. DOCUMENT CONTENT 4
1.4. VERSION HISTORY 4
2. TYPES, CONSTANTS AND STRUCTURES 5
2.1. TYPE DEFINITIONS 5
2.2. ERROR CONSTANTS 5
2.3. LOG LEVEL CONSTANTS 6
2.4. MESSAGE TYPE CONSTANTS 6
2.5. STRUCTURES 6
2.5.1. RGD_APIVersionInfo 6
2.5.2. RGD_DeviceInfo 7
2.5.3. RGD_ConnectionInfo 7
2.5.4. RGD_ConnectionsList 8
2.5.5. RGD_Message 8
3. API FUNCTIONS 10
3.1. GENERAL FUNCTIONS 11
3.1.1. Init 11
3.1.2. Fini 12
3.1.3. GetVersion 13
3.2. CONNECTION FUNCTIONS 14
3.2.1. CreateConnection 14
3.2.2. FreeConnection 15
3.2.3. CloseConnections 16
3.2.4. FreeConnections 17
3.2.5. GetConnections 18
3.2.6. Connect 19
3.2.7. Disconnect 20
3.2.8. IsConnected 21
3.3. LOG FUNCTIONS 22
3.3.1. LogOpen 22
3.3.2. LogClose 23
3.3.3. LogGetLevel 24
3.3.4. LogSetLevel 25
3.4. AUTHENTICATION FUNCTIONS 26
3.4.1. SendPIN 26
3.4.2. ChangePIN 27
3.4.3. SendPIN2 28
3.4.4. ChangePIN2 29
3.5. DEVICE INFORMATION FUNCTIONS 30
3.5.1. GetDeviceInfo 30
3.5.2. GetSignalQuality 31
3.5.3. GetBatteryLevel 32
3.5.4. GetDeviceIMEI 33
3.5.5. GetDeviceIMSI 34
3.6. SMS FUNCTIONS 35
3.6.1. GetServiceCentre 35

3.6.2. SetServiceCentre *36*
3.6.3. GetMessageStore *37*
3.6.4. SetMessageStore *38*
3.6.5. SendSMS *39*
3.6.6. GetNewMessages *40*
3.6.7. ClearInBox *41*
3.6.8. GetInBoxCount *42*
3.6.9. GetMessage *43*
3.6.10. DeleteMessage *44*
3.6.11. DeleteMessageByIdx *45*
3.7. OTHER FUNCTIONS 46
3.7.1. SendCmd *46*
4. USING THE SMS CLIENT SDK 47
4.1. DECODING MMS NOTIFICATION INDICATION MESSAGES 47

1. SMS Client SDK Overview

The SMS Client SDK is meant to be an add-on tool for application developers that need to send and retrieve SMS messages from GPRS devices, such as the Siemens MC35i box.

This document is written for the Delphi development environment and as such uses the Delphi definitions for all data types, constants and records.

The samples distributed with the package contain the C/C++ header files.

1.1. Prerequisites to use the SDK

In order to use the SMS Client SDK the developer needs to have a GPRS device attached to the computer via a COM port (or USB port that emulates a COM port). If this is not present the SDK will not work.

1.2. Document content

TO UNDERSTAND THE CONTENT OF THIS DOCUMENT:

The reader should be familiar with at least one of the following in order to fully understand the information given in this document:

- Basic programming knowledge in Delphi or C/C++

1.3. Version History

Date	Who	Description
22 March 2005	K.Sandell	Initial document draft
27 April 2005	M Krogius	Document checked and verified
01 June 2005	K. Sandell	Document updated and expanded
14 June 2005	K. Sandell	Corrections to the document
27 June 2005	K. Sandell	Added details and updated the function descriptions. Also added structure descriptions for RGD_ConnectionInfo and RGD_Message
28 June 2005	K. Sandell	Added description on how to handle MMS Notification Indication (MMS-NI) messages
30 June 2005	K. Sandell	Added notes to Fini() and Connect() descriptions. Minor spelling errors corrected.

2. Types, Constants and Structures

2.1. Type definitions

Type	Relates to
TGD_Error	Integer, 32 bit signed value
TGD_ConnectionHandle	Integer, 32 bit signed value
TGD_LogLevel	Integer, 32 bit signed value
TGD_MessageStore	Integer, 32 bit signed value
TGD_MessageType	Integer, 32 bit signed value

2.2. Error constants

The following error constants have been defined

Constant	Description
TGD_Err_None	The error is used to indicate SUCCESS
TGD_Err_NotConnected	A connection to device has not been established using the Connect() function call.
TGD_Err_AlreadyConnected	A connection to the device is already active.
TGD_Err_InvalidParameters	The parameters for the function are invalid.
TGD_Err_ConnectFailed	A connection to the device failed.
TGD_Err_DisconnectFailed	Disconnection from the device failed.
TGD_Err_NotEnoughMemory	The function could not allocate enough memory.
TGD_Err_COMPortInvalid	The COM port specified is not acceptable.
TGD_Err_WrongPIN	The PIN code is invalid.
TGD_Err_NoSuchSMSIndex	A SMS message with the indicated index does not exist.
TGD_Err_InvalidHandle	The Connection handle is invalid.
TGD_Err_NotInitialized	The library is not initialized.
TGD_Err_CommandNotSupported	The AT command sent is not supported by the device-
TGD_Err_OtherCommandProcessing	Another Command is already being processed.
TGD_Err_TrialExpired	The trial period has expired.
TGD_Err_Unknown	An unknown error has occurred.

2.3. Log level constants

Constant	Description
TGD_LogLevel_None	No logging
TGD_LogLevel_Error	Only errors are logged
TGD_LogLevel_Warning	Errors and Warnings are logged
TGD_LogLevel_Debug	Everything is logged. Use for Debug purposes only

2.4. Message Type constants

Constant	Description
TGD_MsgType_Text	The message is a TEXT message
TGD_MsgType_Binary	The message is a BINARY message

2.5. Structures

2.5.1. RGD_APIVersionInfo

The version information structure contains the version information about the library.

```
RGD_APIVersionInfo = Record
          MajVersion      : Word;
          MinVersion      : Word;
          Revision        : Word;
          Build           : Word;
          Date            : Cardinal;
          TrialVersion    : Word;
          ExpireDate      : Cardinal;
          ReleaseInfo     : PChar;
          Copyright       : PChar;
End;
```

Field	Type	Description
MajVersion	Word	Major version
MinVersion	Word	Minor version
Revision	Word	Revision / Release
Build	Word	Build number
Date	Cardinal	$YYYYMMDD formatted 32 bit integer
TrialVersion	Word	0=No,1=Yes

ExpireDate	Cardinal	$YYYYMMDD formatted 32 bit integer
ReleaseInfo	PChar	Pointer to the Release Information text
Copyright	PChar	Pointer to the Copyright text

2.5.2. RGD_DeviceInfo

The Device information structure contains information about the device and the SIM card.

```
RGD_DeviceInfo = Record
          Manufacturer    : PChar;
          Model           : PChar;
          Revision        : PChar;
          IMEI            : PChar;
          IMSI            : PChar;
End;
```

Field	Type	Description
Manufacturer	PChar	The Manufacturer of the device
Model	PChar	The Model of the device
Revision	PChar	The Revision of the Model
IMEI	PChar	The IMEI code for the device
IMSI	PChar	The IMSI code for the SIM card

2.5.3. RGD_ConnectionInfo

A structure that describes the connection information.

```
RGD_ConnectionInfo = Record
          COMPort         : Word;
          BPS             : Cardinal;
          Bits            : Byte;
          Flow            : Byte;
          StopBits        : Byte;
          Authenticated   : Boolean;
          OpenedTime      : Cardinal;
          TotalDataIn     : Cardinal;
          TotalDataOut    : Cardinal;
          DeviceInfo      : PGD_DeviceInfo;
End;
```

Field	Type	Description
COMPort	Word	The COM port used

BPS	Cardinal	Bits Per Second. Normal values are 9600 or 19200
Bits	Byte	Number of Data bits. Usually 8.
Flow	Byte	Flow Control. Set this to 0.
StopBits	Byte	Number of Stop Bits.
Authenticated	Boolean	True=The PIN code has been entered.
OpenedTime	Cardinal	Time when the Connection was Opened (Connected).
TotalDataIn	Cardinal	Bytecount of data received trough the COM port
TotalDataOut	Cardinal	Bytecount of data send trough the COM port
DeviceInfo	Pointer	Pointer to the **RGD_DeviceInfo** structure

2.5.4. RGD_ConnectionsList

```
RGD_ConnectionsList = Record
          NumConnections  : Integer;
          FirstConnection : PGD_ConnectionInfo;
End;
```

2.5.5. RGD_Message

A structure that describes a SMS message.

```
RGD_Message = Record
          MsgIdx            : Integer;
          RecvDate          : Cardinal;
          RecvTime          : Cardinal;
          MSISDM            : PChar;
          MsgType           : TGD_MessageType;
          SMSText           : PChar;
          BINData           : Pointer;
          BINDataSize       : Integer;
End;
```

Field	Type	Description
MsgIdx	Integer	Message Index
RecvDate	Cardinal	Date when message was received. YYYYMMDD format
RecvTime	Cardinal	Time when message was recveived. HHMMSS format
MSISDN	PChar	The senders number
MsgType	Integer	The type of the message. See **TGD_MessageType** for details.

SMSText	PChar	The contents of the SMS message as text. This is NIL if the message is of Binary type.
BINData	Pointer	Pointer to the contents of the SMS message if it is of Binary data type.
BINDataSize	Integer	The size of the Binary data.

3. API Functions

This chapter explains all the API functions. The functions are grouped according to a category.

Category	Description
General functions	These functions do not require a active connection handle
Log function	These functions are related to the Logging of data
Authentication functions	These functions manage authentication of the user for the device
Device Info functions	These functions provide information about the device and SIM card
SMS functions	These functions manage SMS messages and SMS-C related things
Other functions	Functions that are not categorized in any of the other categories

3.1. General functions

The general functions are used for initialization and finalization of the library, as well as version information queries.

3.1.1. Init

Description The Init call is used to initialize the SDK library. This must be the 1st function to call for the library

Syntax function Init(): **TGD_Error**;

Parameters None

Result The function returns **TGD_Err_None** if the library was initialized successfully.

See also **FIni**

3.1.2. Fini

Description The Fini call is used to finalize the GPRS SDK library. This must be the last function to call for the library

Syntax function FIni(): **TGD_Error**;

Parameters None

Result The function returns **TGD_Err_None** if the library was finalized successfully.

See also **Init**

Notes It is safe to call this function with existing connected connections. The function automatically handles all cleanup tasks needed to close and free any existing connections.

3.1.3. GetVersion

Description The GetVersion call is used to retrieve the **RGD_APIVersionInfo** structure for the Library.

Syntax function (Out APIVersionInfo: **PGD_APIVersionInfo**) **TGD_Error**;

Parameters Out APIVersionInfo – A pointer to the version information structure

Results The function returns **TGD_Err_None** if the pointer contains a valid value.

See also **RGD_APIVersionInfo**

3.2. Connection functions

The connection functions manipulate the connection to the device.

3.2.1. CreateConnection

Description This function creates a virtual connection towards a device. The connection handle returned is used in all the subsequent functions that need a connection.

Syntax function CreateConnection(COMPort: Word; BaudRate: Integer; Bits, Flow, StopBits: Byte; Out Handle: **TGD_ConnectionHandle**): **TGD_Error**;

Parameters
COMPort – Numeric representation of the COM port. COM1=1, COM2=2, etc-
BaudRate – The wanted BaudRate. 19200 is recommended
Bits – The number of bits in the data. 8 is recommended
Flow – The Flow Control flag. 0=Hardware (recommended)
StopBits – Number of stop bits. 1 is recommended
Out Handle – The variable that will receive the Connection handle

Results
TGD_Err_None - The connection was created successfully
TGD_Err_TrialExpired – The trial period has expired.
TGD_Err_NotInitialized – The **Init()** functions has not been called

See also **FreeConnection**, **FreeConnections**

3.2.2. FreeConnection

Description This function frees a created connection. It also closes the connection if it is open.

Syntax function FreeConnection(Handle: **TGD_ConnectionHandle**): **TGD_Error**;

Parameters Handle – The Handle of the connection to be freed

Results **TGD_Err_None** - The connection was freed successfully

TGD_Err_InvalidHandle – The handle is not a valid connection handle

TGD_Err_NotInitialized – The **Init()** functions has not been called

See also **CreateConnection**, **FreeConnections**

3.2.3. CloseConnections

Description This function Disconnects all Connected connections.
Note: It does not free the connections.

Syntax function CloseConnections():**TGD_Error**;

Parameters None

Results **TGD_Err_None** - All connected connections have been disconnected
TGD_Err_NotInitialized – The **Init()** functions has not been called

See also **Connect**, **Disconnect**

3.2.4. FreeConnections

Description This function disconnects and frees all connections. It can optionally free only those connections that are NOT connected.

Syntax function FreeConnections(OnlyDisconnected: Boolean): **TGD_Error**;

Parameters OnlyDisconnected – If true, only disconnected connections are freed.

Results **TGD_Err_None** - All connections where freed

TGD_Err_NotInitialized – The **Init()** functions has not been called

See also **FreeConnection**, **Disconnect**

3.2.5. GetConnections

Description This function returns a list of all the connections that have been created.

Syntax function GetConnections(Out ConnectionsList: **PGD_ConnectionsList**): **TGD_Error**;

Parameters Out ConnectionsList – A pointer to the first returned Connection Object

Results **TGD_Err_None** - No errors

TGD_Err_NotInitialized – The **Init()** functions has not been called

See also **Connect**, **Disconnect**

3.2.6. Connect

Description This function attempts to establish a connection to the device. If the connection is established the function automatically gathers the Device Information from the device.

Syntax function Connect(Handle: **TGD_ConnectionHandle**): **TGD_Error**;

Parameters Handle – The handle of the connection

Results
TGD_Err_None - The connection connected to the device
TGD_Err_NotInitialized – The **Init()** functions has not been called
TGD_Err_InvalidHandle – The connection handle is invalid
TGD_Err_ConnectFailed – The connection attempt failed

See also **Disconnect**, **FreeConnection**

Note If the COM port exists, but there is no device present (DSR signal) the function returns an error.

3.2.7. Disconnect

Description This function disconnects a connected connection from the device.

Syntax function Disconnect(Handle: **TGD_ConnectionHandle**): **TGD_Error**;

Parameters Handle – The handle of the connection

Results
TGD_Err_None - The connection was disconnected from the device
TGD_Err_NotInitialized – The **Init()** functions has not been called
TGD_Err_InvalidHandle – The connection handle is invalid
TGD_Err_NotConnected – The connection is not connected to the device
TGD_Err_DisconnectFailed – Failed to disconnect from device

See also **Connect**

3.2.8. IsConnected

Description This function checks if the Connection is connected to the device.

Syntax function IsConnected(Handle: **TGD_ConnectionHandle**): **TGD_Error**;

Parameters Handle – The handle of the connection

Results
TGD_Err_None - The connection is connected to the device
TGD_Err_NotInitialized – The **Init()** functions has not been called
TGD_Err_InvalidHandle – The connection handle is invalid
TGD_Err_NotConnected – The connection is not connected to the device

See also **Connect**, **Disconnect**

3.3. Log functions

The log functions manipulate the log files for a connection.

3.3.1. LogOpen

Description This function opens a logfile for the connection.

Syntax function LogOpen(Handle: **TGD_ConnectionHandle**; FileName:PChar): **TGD_Error**;

Parameters Handle – The handle of the connection

Filename – The filename of the logfile (including path)

Results **TGD_Err_None** - The log was opened for the connection

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

See also **LogClose**

3.3.2. LogClose

Description This function closes an open logfile for a connection.

Syntax function LogClose(Handle: **TGD_ConnectionHandle**): **TGD_Error**;

Parameters Handle – The handle of the connection

Results **TGD_Err_None** - The log was closed for the connection
TGD_Err_NotInitialized – The **Init()** functions has not been called
TGD_Err_InvalidHandle – The connection handle is invalid

See also **LogOpen**

3.3.3. LogGetLevel

Description This function gets the logging level for an open logfile.

Syntax function LogGetLevel(Handle: **TGD_ConnectionHandle**; Out LogLevel: **TGD_LogLevel**): **TGD_Error**;

Parameters Handle – The handle of the connection

LogLevel – The current loglevel

Results **TGD_Err_None** - No error

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

See also **LogSetLevel**

3.3.4. LogSetLevel

Description This function sets the logging level for an open logfile.

Syntax function LogSetLevel(Handle: **TGD_ConnectionHandle**; LogLevel: **TGD_LogLevel**): **TGD_Error**;

Parameters Handle – The handle of the connection

LogLevel – The wanted logging level

Results **TGD_Err_None** - The log-level was set successfully

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

See also **LogGetLevel**

3.4. Authentication functions

The authentication functions manage the PIN codes of the device.

3.4.1. SendPIN

Description This function is used to authenticate the application towards the Device.

Note: The authentication is needed only once per device power-up.

Syntax function SendPIN(Handle: **TGD_ConnectionHandle**; PIN: PChar): **TGD_Error**;

Parameters Handle – The handle of the connection

PIN – The PIN number

Results **TGD_Err_None** - Authentication successful

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

TGD_Err_NotConnected – The connection is not connected

TGD_Err_WrongPIN – Invalid / Wrong PIN

See also **ChangePIN**

3.4.2. ChangePIN

Description This function allows the user to change the PIN1 for the device.

Syntax function ChangePIN(Handle: **TGD_ConnectionHandle**; OldPIN, NewPIN: PChar): **TGD_Error**;

Parameters
Handle – The handle of the connection
OldPIN – The old PIN number
NewPIN – The new PIN number

Results
TGD_Err_None - No error
TGD_Err_NotInitialized – The **Init()** functions has not been called
TGD_Err_InvalidHandle – The connection handle is invalid
TGD_Err_NotConnected – The connection is not connected
TGD_Err_WrongPIN – Invalid / Wrong PIN

See also **SendPIN**

3.4.3. SendPIN2

Description This function sends the PIN2 to the device.

Syntax function SendPIN2(Handle: **TGD_ConnectionHandle**; PIN2: PChar): **TGD_Error**;

Parameters Handle – The handle of the connection

PIN2 – The PIN2 number

Results **TGD_Err_None** - Authentication successful

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

TGD_Err_NotConnected – The connection is not connected

TGD_Err_WrongPIN – Invalid / Wrong PIN2

See also **ChangePIN2**

3.4.4. ChangePIN2

Description This function changes the PIN2 on the device.

Syntax function ChangePIN2(Handle: **TGD_ConnectionHandle**; OldPIN2, NewPIN2: PChar): **TGD_Error**;

Parameters
Handle – The handle of the connection
OldPIN2 – The old PIN2 number
NewPIN2 – The new PIN2 number

Results
TGD_Err_None - No error
TGD_Err_NotInitialized – The **Init()** functions has not been called
TGD_Err_InvalidHandle – The connection handle is invalid
TGD_Err_NotConnected – The connection is not connected
TGD_Err_WrongPIN – Invalid / Wrong PIN2

See also **SendPIN2**

3.5. Device information functions

The Device information functions retrieve information about the device as well as the SIM card.

3.5.1. GetDeviceInfo

Description This function retrieves the device information into a structure. This command is issued automatically when a connection is connected to a device.

Syntax function GetDeviceInfo(Handle: **TGD_ConnectionHandle**; Out DeviceInfo: **PGD_DeviceInfo**): **TGD_Error**;

Parameters Handle – The handle of the connection

DeviceInfo – A pointer to a Device Information structure

Results **TGD_Err_None** - No error

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

TGD_Err_NotConnected – The connection is not connected

See also **Connect**

3.5.2. GetSignalQuality

Description This function retrieves the current signal quality that the device reports.

By calling this function consecutively a historical log of signal quality can be stored.

Syntax function GetSignalQuality(Handle: **TGD_ConnectionHandle**; Out SignalQuality: Integer): **TGD_Error**;

Parameters Handle – The handle of the connection

SignalQuality – The current signal quality.

Results **TGD_Err_None** - No error

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

TGD_Err_NotConnected – The connection is not connected

See also

Notes The SignalQuality values returned have the following meaning:

0	-113 dBm or less
1	-111 dBm
2...30	-109... -53 dBm
31	-51 dBm or greater
99	not known or not detectable

3.5.3. GetBatteryLevel

Description This function retrieves the current battery charge level for the device.

Note: This function may not be implemented on all Devices.

Syntax function GetBatteryLevel(Handle: **TGD_ConnectionHandle**; Out BatteryLevel: Integer): **TGD_Error**;

Parameters Handle – The handle of the connection

BatteryLevel – The current battery level

Results **TGD_Err_None** - No error

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

TGD_Err_NotConnected – The connection is not connected

See also

Notes The returned values have the following meaning:

0 Feature not implemented, or no battery info available

3.5.4. GetDeviceIMEI

Description

This function retrieves the IMEI code for the device. The IMEI code is the product serial number identification and is always unique to the device.

IMEI stands for International Mobile Equipment Identification.

Syntax

function GetDeviceIMEI(Handle: **TGD_ConnectionHandle**; Out IMEI: PChar): **TGD_Error**;

Parameters

Handle – The handle of the connection

IMEI – A pointer to the IMEI string

Results

TGD_Err_None - No error

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

TGD_Err_NotConnected – The connection is not connected

See also

3.5.5. GetDeviceIMSI

Description

This function retrieves the IMSI code for the SIM card in the device. The IMSI code is always unique for the SIM card, and contains information about the operator as well as the serial number for the SIM card.

IMSI stands for International Mobile Subscriber Identity.

Syntax

function GetDeviceIMSI(Handle: **TGD_ConnectionHandle**; Out IMSI: PChar): **TGD_Error**;

Parameters

Handle – The handle of the connection

IMSI – A pointer to the IMSI string

Results

TGD_Err_None - No error

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

TGD_Err_NotConnected – The connection is not connected

See also

Notes

The IMSI is composed of two major parts, the 1st part is the MCC (Mobile Country Code) + MNC (Mobile Network Code) codes, and the 2nd part is the IMSI itself.

Example IMSI: 24405xxxxxxxxx

In the example above the 244 is the MCC country code (Finland) and the 05 (Elisa) is the MNC code for the operator within the country.

For a full list of MCC and MVC codes please visit **http://www.numberingplans.com**.

The most common numbering plan that is used for the IMSI codes is ITU-T Recommendation E.212.

3.6. SMS functions

The SMS functions allow the user to send and retrieve SMS messages from the device.

3.6.1. GetServiceCentre

Description This function retrieves the stored SMS-C number in the device.

Syntax function GetServiceCentre(Handle: **TGD_ConnectionHandle**; Out CentreNumber: PChar): **TGD_Error**;

Parameters Handle – The handle of the connection

CentreNumber – A pointer to the SMS-C Number string

Results **TGD_Err_None** - No error

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

TGD_Err_NotConnected – The connection is not connected

See also **SetServiceCentre**

Notes To successfully send and receive SMS messages the SMS-C number must be set.

3.6.2. SetServiceCentre

Description This function sets the SMS-C number for the device.

Syntax function SetServiceCentre(Handle: TGD_ConnectionHandle; CentreNumber: PChar): TGD_Error;

Parameters Handle – The handle of the connection

CentreNumber – A pointer to the SMS-C Number string

Results **TGD_Err_None** - No error

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

TGD_Err_NotConnected – The connection is not connected

See also **GetServiceCentre**

3.6.3. GetMessageStore

Description This function returns the Message Store currently active in for the Device.

Syntax function GetMessageStore(Handle: TGD_ConnectionHandle; Out MSGStore: PChar): TGD_Error;

Parameters Handle – The handle of the connection

MSGStore – A pointer to the MSGStore name string

Results **TGD_Err_None** - No error

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

TGD_Err_NotConnected – The connection is not connected

See also **SetMessageStore**

Notes The returned values for the function is one of the following:

SM	SIM Card message store
ME	Device message store (default)

3.6.4. SetMessageStore

Description

Syntax function SetMessageStore(Handle: **TGD_ConnectionHandle**; MSGStore: PChar): **TGD_Error**;

Parameters Handle – The handle of the connection

MSGStore – A pointer to the MSGStore name string

Results **TGD_Err_None** - No error

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

TGD_Err_NotConnected – The connection is not connected

See also **GetMessageStore**

Notes The values that can be passed for the function is one of the following:

SM	SIM Card message store
ME	Device message store (default)

3.6.5. SendSMS

Description This function sends a Text SMS message to another MSISDN number.

Syntax function SendSMS(Handle: **TGD_ConnectionHandle**; MSISDN, TextMsg: PChar; Flash: Boolean): **TGD_Error**;

Parameters

Handle – The handle of the connection

MSISDN – A pointer to the MSDISDN (number) that will receive the SMS

TextMsg – A pointer to the string to send as a SMS. Max 160 characters

Flash –TRUE if the SMS is displayed as a FLASH on the destination device.

Results

TGD_Err_None - No error

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

TGD_Err_NotConnected – The connection is not connected

TGD_Err_InvalidParameters – One of the parameters are invalid

See also

3.6.6. GetNewMessages

Description This function checks the device for any unread received SMS messages, and retrieves them into the connection InBox.

Syntax function GetNewMessages(Handle: **TGD_ConnectionHandle**; Out Count: Cardinal): **TGD_Error**;

Parameters Handle – The handle of the connection

Count – The number of messages retrieved and stored in the InBox

Results **TGD_Err_None** - No error

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

TGD_Err_NotConnected – The connection is not connected

See also **ClearInBox**, **GetInBoxCount**

Note This function deletes the processed messages from the device.

3.6.7. ClearInBox

Description This function clears the connections InBox.

Syntax function ClearInBox(Handle: **TGD_ConnectionHandle**): **TGD_Error**;

Parameters Handle – The handle of the connection

Results **TGD_Err_None** - No error
TGD_Err_NotInitialized – The **Init()** functions has not been called
TGD_Err_InvalidHandle – The connection handle is invalid

See also **GetInBoxCount**

3.6.8. GetInBoxCount

Description This function retrieves the number of messages in the Connection InBox. If no new messages are available the function returns 0 in Count.

Syntax function GetInBoxCount(Handle: **TGD_ConnectionHandle**; Out Count: Cardinal): **TGD_Error**;

Parameters Handle – The handle of the connection

Out Count – The number of messages in the InBox

Results **TGD_Err_None** - No error

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

See also **ClearInBox**

3.6.9. GetMessage

Description This function retrieves a message from the Connection InBox, based on the position of the message in the list.

Syntax function GetMessage(Handle: **TGD_ConnectionHandle**; Position: Integer; Out MsgRec: **PGD_Message**): **TGD_Error**;

Parameters

Handle – The handle of the connection

Position – The position of the messages to retrieve. 1st position is 0.

Out MsgRec – Will contain a pointer to the Message Structure.

Results

TGD_Err_None - No error

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

See also **DeleteMessage**, **DeleteMessageByIdx**

3.6.10. DeleteMessage

Description This function deletes a message from the Connection InBox based on the position of the message in the list.

Syntax function DeleteMessage(Handle: **TGD_ConnectionHandle**; Position: Integer): **TGD_Error**;

Parameters Handle – The handle of the connection

Position – The position of the messages to delete. 1st position is 0.

Results **TGD_Err_None** - No error

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

See also **DeleteMessageByIdx**

3.6.11. DeleteMessageByIdx

Description This function deletes a message from the Connection InBox based on the Index Number of the message. The index number is auto-generated by the SDK as new messages are placed in the InBox.

Syntax function DeleteMessageByIdx(Handle: **TGD_ConnectionHandle**; Idx: Integer): **TGD_Error**;

Parameters Handle – The handle of the connection

Idx – The Index of the messages to delete.

Results **TGD_Err_None** - No error

TGD_Err_NotInitialized – The **Init()** functions has not been called

TGD_Err_InvalidHandle – The connection handle is invalid

See also **DeleteMessage**

3.7. Other functions

The other functions are used for generic Device access and uncategorized functions.

3.7.1. SendCmd

Description The SendCmd call is used to send any arbitrary AT command to the device. By using this function the developer may implement any type of functionality for a specific device.

Syntax function SendCMD(Handle:TGD_ConnectionHandle; CMD:PChar; Out Response: PChar): **TGD_Error**;

Parameters Handle – The handle to the connection

CMD – A PChar that points to the command to send

OUT Response – A PChar that receives the pointer to the reply for the command

Results The function returns **TGD_Err_None** if the command was sent successfully.

See also

4. Using the SMS Client SDK

In order to successfully use the SDK the application developers need to perform the following tasks to initialize the SDK.

1. Load the library
2. Call **Init()** to initialize the library
3. Create a connection using the **CreateConnection()** function
4. Connect the connection to the device using the **Connect()** function

Once the connection is established the developer can access any of the functions that are wanted.

When an application whishes to close/free the DLL the following tasks should be done:

1. Make sure no ongoing API call is in progress
2. Call the **Disconnect()** function
3. Call the **FreeConnection()** function
4. Call the **Fini()** function
5. Unload the library

4.1. Decoding MMS Notification Indication messages

In order to decode the MMS Notification Indication (MMS-NI) messages the following tasks need to be performed:

1. Initialize the SMS Client SDK using Init()
2. Load and Initialize the MMS Stack SDK
3. Connect to the device using Connect()
4. Call GetNewMessages()
5. For each message received do
    a. Call GetMessage()
    b. Call wsp_push_message_init()
    c. Call wsp_push_message_decode()
    d. Call wsp_push_message_release()
    e. Call mms_message_decode()

At this point the MMS-NI message should be decoded and the download URL should be accessible. For instructions on how to download the actual MMS message please refer to the MMS Stack SDK documentation.