handspring

Using the SMS Library

Table of Contents

List of Acronyms	
GSM	Groupe Spécial Mobile, or, Global System for Mobile Communications
ISP	Internet Service Provider
PPP	Point-to-Point Protocol
SIM	Subscriber Identity Module
SMS	Short Message Service
UDH	User Data Header

1. Overview

This document describes how to send and receive messages using the SMS interfaces in the SMS library. The SMS application in the VisorPhone uses the library defined in this document. Third-party developers can use this library to create their own custom SMS applications.

Technically, the SMS library is part of the GSM library, but it is described in a different document since it is logically a separate unit. The SMS library relies on functionality from the GSM library and uses some similar methods.

This document describes some of the key features of the SMS library. Among those features are:

- sending messages
- receiving messages.
- encoding
- message segmentation
- message database

2. What is SMS?

The point-to-point Short Message Service (SMS) provides a means of sending short messages to and from a GSM phone. SMS is implemented using a Service Center, which acts as a store and forward center for short messages.

Two different point-to-point services are defined in the SMS specification: mobile originated and mobile terminated. Mobile originated messages are transported from a GSM phone to a service center. These messages may be destined for other mobile users, or an email gateway. Mobile terminated messages are transported from the service center to a GSM phone. These messages may have originated from another GSM phone or a variety of other sources (e.g., email or website).

A message sent on the SMS network is limited to 160 characters. If a longer message is desired, then the message must be segmented.

3. Why Use the SMS Library?

The SMS library allows for sending and receiving short messages from the VisorPhone to another SMS-enabled recipient.

These messages can be sent much more quickly than when using a PPP connection with the VisorPhone. This is because SMS does not require an initial connection to an ISP. Connecting to an ISP can take up to 30 seconds, including modem negotiation time and user authentication.

Cost is another consideration for using SMS. A typical U.S. service provider charges 15 cents a minute for data service, but only 10 cents for a short message.

Additionally, SMS messages can be sent directly to the VisorPhone. They do not require a client application to dial in to the network to check for new email; instead, the message is sent to the client application. The client can even receive an SMS message while a voice or data call is in progress.

Finally, the client application can request confirmation that a message has arrived at its destination address. A typical email system does not allow for this functionality.

4. Understanding the SMS Library

The SMS library takes care of the low-level details of communicating with the VisorPhone. All incoming messages are indicated to the application using launch codes. A successful (or unsuccessful) transmission of a message is also indicated using launch codes.

The SMS library handles the following functions for SMS messages.

- Sending
- Receiving
- Encoding

"Encoding" only refers to character encoding. Currently, only the GSM alphabet is supported. The GSM alphabet is different from the Palm OS alphabet. Additionally, encryption and compression of messages is not supported in the current version of the SMS library.

Note: The current version of the SMS library only handles text messages. All messages handled by the library are assumed to be text, and are handled appropriately. If a client application wants to send binary data, it must encode and decode the data properly.

4.1. Incoming Message Events

An application must register itself with the SMS library in order to receive incoming message events.

Each incoming message is indicated to the application by sending a `kMessageInd` event. A segmented message is indicated as a new incoming message only after all of its parts have been received. If a new message was added to the database (first part of a segmented message), the application is sent the `kSegmentInd` event to trigger an update of a list that might contain this new message.

If the user requests a status report for a message, the service center sends one or more status report messages. These messages contain binary information, rather than text. A status report contains the ID of the message the report refers to along with a binary result code. The SMS library converts the status reports into text form. The error code is translated into a meaningful description, and the message recipient is shown. The reference to the original message is replaced by this message's text. The application is notified of the incoming status report with a `kMessageInd` event. The status report becomes a regular incoming message.

If a status report is requested for a message that is sent to multiple recipients, the service center will send a status report for each recipient. The SMS library does not combine these status reports into a single report.

4.2. Outgoing Messages

A message is sent by creating a new outgoing message using `GSMLibNewMessage` and filling in the desired recipient and message text. More than one recipient may be specified if the message is being sent to a group. The message text may be longer than 160 characters; in that case, the text is automatically segmented. Since segmentation is transparent to the user of the SMS library, the library segments and reassembles the messages automatically.

4.3. GSM Alphabet vs. Palm OS Alphabet

A special alphabet is used to encode SMS messages. The text of all incoming and outgoing messages stored in the message database is encoded using the standard Palm OS encoding.

When a message is received, its encoding is changed from the GSM alphabet to the Palm OS alphabet. Any missing characters are replaced by substitution strings. A message is encoded in the GSM alphabet when being sent. Optionally, substitution strings may be converted to their character equivalents. The following table shows the character with its corresponding substitution string.

Table 1. Substitution strings

Δ	\Delta	Π	\Pi
Φ	\Phi	Ψ	\Psi
Γ	\Gamma	Σ	\Sigma
Λ	\Lambda	Θ	\Theta
Ω	\Omega	Ξ	\Xi

Important: The client application should not assume that a message contains 160 characters or less. Even if the actual message is less than 160 characters, the message's text may be longer than 160 characters because characters that are not available on Palm OS are replaced by substitution strings.

4.4. Message Segmentation

Most mobile phones allow the user to compose messages of only 160 characters or less; however, with the SMS library, the maximum length of a message is 38760 characters. Messages that contain more that 160 characters are segmented.

The SMS library supports three types of segmentation schemes. Two of these methods are text-based, while the third is binary. The binary method is preferred, since it allows for the reassembly of messages, even if they arrive out of order.

4.4.1 Binary Segmentation

The binary segmentation scheme works by adding a UDH to each segment of a segmented message. The UDH contains a reference number to identify the message, the segment's index, and the total number of segments. Because the UDH takes 8 bytes, the number of characters in a message is reduced to 152.

Using the UDH, it is possible to reassemble messages from their segments, even if the segments arrive out-of-order. The reassembly of incoming segmented messages is transparent to the client application. The client application may retrieve a message's text, even if all the segments have not been received. Use `GSMLibGetText` to retrieve the message.

When the first segment of a segmented message is received, the client receives a `kSegmentInd` event after the segment has been stored in the database. When all of the segments have been received, a `kMessageInd` event is sent.

Automatic reassembly of messages works if all parts of the message are received within one week after the first segment is received. After one week, segmentation information is deleted by the library.

4.4.2 Textual Segmentation

The SMS library supports two textual segmentation schemes. One segmentation scheme is used only for sending segmented messages to an email gateway. The other is used for segmenting regular text messages.

The segmentation scheme used to send messages to an email gateway does not allow the reassembly of the message if the messages arrive out of order. This scheme is a recognized GSM standard. It works by inserting "+" signs into the message's text. The length of an "inner" segment is reduced to 158 characters, and the length of the first and last segments is 159 characters. A message with three parts is segmented as follows:

```
First segment+
+Inner segment+
+Last segment
```

Note: For messages sent to an email gateway, the recipient's address will add to the message's length.

This segmentation scheme is used exclusively to send messages to an email gateway. The SMS library does not use this scheme to send text messages to normal subscribers.

The second scheme for segmentation adds header information to every segment. The header is of the form "*i/k*", where *i* is the segment's index and *k* is the total number of segments. The length of the header is not constant, and is dependent upon the values of *i* and *k*. A message with three parts is segmented as follows:

```
1/3 First segment
2/3 Second segment
3/3 Last segment
```

The library does not attempt to reassemble messages when they are sent using this segmentation scheme. The user must reassemble the messages as needed. The reassembly is not automatically done by the library because the header does not indicate clearly to which segmented message a segment belongs.

4.5. Message Database

All incoming and outgoing SMS messages are stored in a message database on the device. An outgoing message stored in the database may be sent using the SMS library. Incoming messages received are also stored in this database. The SMS library only handles messages stored in this database. If an application wants to store the messages in a separate database, it must copy the messages from the SMS database.

This section describes the fields for a single SMS message. The standard Palm OS routines for databases are used to manage these records.

Important: A message's internal structure is private and not simple. Client applications should modify data in an SMS message only by using the functions provided by the SMS library.

A record in the message database has four different parts: the first part has a fixed size, and the size of the other three parts is variable. As a result, a complete record is variable size.

- Header information (fixed size)
- Segmentation information (variable)
- Address information (variable)
- Message text (variable)

1.1.1 First part of a record

The first part is the message header information. This is a fixed size. These fields are used to store flags and determine the size of the variable sized field. Some fields are not used for all messages. For example, field `validity` is used only for outgoing messages and field `segments` is used only for incoming messages.

Table 2. Header information

Field	Type	Description
`owner`	`DWord`	Application owning this message
`type`	`SMSMessageType`	Message type (incoming or outgoing)
`status`	`SMSMessageStatus`	Current status of the message
`date`	`UInt32`	Date of message sending or receipt
`flags`	`UInt32`	Miscellaneous flags
`validity`	`UInt8`	Validity period for message. This tells the service center how long to store the message before deleting it.
`segments`	`UInt8`	Total number of parts of a message
`size`	`UInt16`	Size of the address information in bytes

The message database contains messages stored by different applications. The owner holds the creator of the application that created an SMS message, or is responsible for it. An application's four-byte unique creator is stored along with a message. The library does not assign a default creator to new messages. Client application should use `GSMLibSetOwner` to set the `owner` field.

The `date` field contains the sending date for outgoing messages and the receiving date for incoming messages. The date is in standard Palm OS date format. If the outgoing message has not been sent, this field contains the date of the message's creation. For an incoming message, this field contains the timestamp passed by the service center, rather than the date when the message was received by the VisorPhone.

The `flags` field holds 32 bits of data. All bits not defined in this table are reserved for future use and should be set to 0.

Table 3. Flag values

Bit	Name	Description
0	`kGreekSymbols`	Set if there was an error while converting the message text (Greek symbols)
1	`kMissingPart`	Set if some part of a message is missing (incoming only)
2	`kAutoDelete`	Set if the message should be deleted after successful transmission
3	`kNotification`	Set if a notification should be sent when the message has been successfully transmitted
4	`kDontEncode`	Set if encoding the message text should be suppressed
5	`kSubstitution`	Set if substitution strings should be replaced by GSM characters
6	`kFailed`	Set if there was an error while sending the message
7	`kStatusReport`	Set if message is a status report (incoming only) or if a status report is requested (outgoing only)
8	`kFreeReply`	Set if reply must be sent using same service center
9	`kInternetEMail`	Set if message is sent to an email gateway
10	`kTextSegments`	Set if textual segmentation methods should be used
11-15		Reserved. Must be set to 0.
16	`kRead`	Set if the message was read (incoming only)
17-31		Reserved for client applications

The `segments` and `size` fields are used to manage the variable-sized parts of the message. A segmented message may consist of up to 255 parts. Each part contains up to 152 characters.

The segments field contains the total number of a message's segments. For each segment, the length is stored using two bytes. Missing parts are assigned the length of 65535 (0xFFFF). The size of the segmentation information is therefore 2 x `segments` bytes. For outgoing messages, this field is set to 0; thus, the segmentation information does not exist.

The `size` field specifies the size of the address information (the list of recipients or the sender) in bytes.

Note: The `size` field is declared as an array in C. The size of segment `i` may be accessed by the expression `size[i]`. The element with index 0 contains the length of the address information.

4.5.3 Second part of a record

The second part is segmentation information. This part is variable in size. This is accessed through the `size` array. Each segment's size is represented with 2 bytes.

4.5.4 Third part of a record

The third part is address information. This part is variable in size. The size of this address information is defined in the `size` field. The address information contains the data in a `GSMAddressList` structure.

4.5.5 Fourth part of a record

The fourth part is the actual message's text. This part is variable in size. The size is calculated by taking the size of the complete message, and subtracting the sizes of the first three parts. All characters in this part are considered to be Palm OS encoded. For outgoing messages, the characters are converted to the GSM alphabet when the message is sent.

Note: The conversion of the text is done just before the message is sent. The result of the conversion is not stored with the message. If sending a message fails, the text is converted again when the message is sent.

5. Library Reference

This section contains a detailed description of all functions supported in the SMS library. Only the functions from the library's public interface are documented.

The SMS library was implemented using C++; however, the header file included with the library can also be used with C. The method used to call library routines makes this possible, since the function's name is translated into an index and then into a jump table.

Some routines provided by the SMS library are used for communicating with the network. These commands are completed asynchronously. Depending on the current network load and the type of interaction with the network, it may take some time before the request is completed.

The client application may want to show a progress dialog box while communication is in process.

5.1. Data Types

The SMS library defines a few data types for handling SMS messages. Some data types defined in the GSM library are also used by the SMS library.

5.2. Event Codes

Some data types (especially enumeration types) defined by the GSM library contain constants that are used exclusively by the SMS library. One example is the enumeration type `GSMEvent` used to indicate to a client application what event is sent by the GSM software.

Table 4. Event Codes

Event Code	Description
`kMessageInd`	This is sent to the registered application if a new message has been received. This event is only sent for complete messages.
`kSegmentInd`	This indicates that a new message has been added to the database. This event is only used for segmented messages to indicate to the application that they should update the list of incoming messages (if the user is able to view partially received messages).
`kMessageStat`	This is sent to the message's owner whenever the status of the message has changed. This event may be used by an application to assign the message to a new category when its status changes from `kPending` to `kSent`.
`kMessageDel`	This is sent to the owner of a message if a message has been deleted from the database. This event may be used by an application to remove a message from a currently shown list and/or update any other data structures.
`kMessageMoved`	This indicates that SMS messages stored on a SIM have been successfully transferred to the message database. This happens if the user inserts a SIM with stored messages into a VisorPhone. The SMS library will ask the user whether these messages should be stored in the message database. If confirmed, the messages are transferred. After the transfer, this event is passed to the client application. The client application may display a dialog after receiving this event to indicate whether all messages were transferred without an error.
`kError`	This event is sent to the client application whenever an error occurs.
`kButton`	This indicates that a hardware button on the VisorPhone has been pressed. The client application can switch to the main application as a result of this event.

Note: If there is an error in the transmission of a message, the status of the message is set to `kSent` and the client application is notified of the error with the `kError` event. Additionally, the `kFailed` bit is set in the message's `flags` field.

All events are passed with a parameter that indicates the affected messages. The parameters are passed using the `SMSParams` structure.

5.3. Message Types

The SMS library distinguishes between different types of messages. The type of message is important because some functions can only be used with a certain message type. For example, the function to forward a message to another address is only defined for incoming messages.

The enumeration type SMSMessageType defines all the types of messages. The current SMS library handles only two types of messages: kMTIncoming and kMTOutgoing. Any other value that an application may pass as the type of a message is reserved by the SMS library for future expansion.

Table 5. Message types

Message Type	Description
kMTIncoming	This is an incoming message.
kMTOutgoing	This is an outgoing message.

Note: The current version of the SMS library does not support cell-broadcast messages.

All messages of the SMS library are text messages encoded in the standard GSM library. The library does not support text messages encoded in a different alphabet or binary message.

5.4. Message Status

When a message is sent or received, its status field is changed by the SMS library. When that happens, the client application is sent the kMessageStat event. The enumeration type SMSMessageStatus defines the possible values for the status field.

Table 6. Message Status

Status	Category	Notes
kNone	Saved	This incoming message has not yet been reported to the user, or this unsent outgoing message has been 'saved' by the user.
kReceiving	InBox	This indicates that it is a partially received message.
kReceived	InBox	This is a completely received message.
kPending	Pending	All pending messages have this status. Even messages sent immediately briefly have this status.
kSending	Pending	The message is currently being sent.
kSent	Sent	The message has been sent, either successfully or with an error.

An incoming message can have the status of kNone, kReceiving, and kReceived. As long as a message is being received, its status is kReceiving. After all parts have been received, its status is changed to kReceived.

An outgoing message can have the status of kNone, kPending, kSending, or kSent. If an outgoing message is saved in the SMS database, its status is kNone. If an application calls GSMLibSendMessage and the network is not available for sending, the message's status is changed to kPending. All messages with the kPending

status will be sent when the network becomes available at a later time. If a message is currently being sent, the status is changed to `kSending`. After all parts of a message have been sent, its status is set to `kSent`.

5.5. Sending Options

When an SMS message is sent to the GSM network, several options may be defined by the sending entity. The sending options are defined in the structure `SMSSendOptions`.

```
struct SMSSendOptions {
     Boolean freeReply;
     Boolean statusReport;
     UInt8 validity;
};
```

Table 7. Sending Options

Field Name	Description
`freeReply`	This indicates to the network that the sender of the message is charged with the cost of a reply to this message.
`statusReport`	This indicates if the network sends a report indicating whether the message was delivered to the recipient. If a status report is requested for a segmented message, the SMS library requests only a status report for the first part of the message. If a status report is requested for a message sent to a group of recipients, the SMS library requests a status report for each recipient.
`validity`	This defines how long the network stores a message if it cannot be sent to the recipient immediately.

The `validity` field contains a value specifying the time that the network should store a message before it is discarded. This value is defined in the GSM specification.

Table 8. Validity Values

Value	Validity Period
0 to 143	(x + 1) x 5 minutes (i.e., 5 minute intervals up to 12 hours)
144 to 167	12 hours + (x – 143) x 30 minutes)
168 to 196	(x – 166) x 1 day
197 to 255	(x – 192) x 1 week

6. SMS Library API

6.1. Registering with the Library

In order for your application to work with the SMS Library, it must register itself with the library.

Table 9. Registering

Function	Description

`Err`
`GSMLibRegister``(UInt16,`
`  UInt32 creator,`
`  UInt16 services);`
	A client application uses this function to register with the GSM library when it is starting up. If the registration was successful, it returns 0. If there is no application with the given creator, `gsmErrUnknownApp` is returned.
	
	This function is also used to cancel the registration with the library. The client application just passes 0 as `services`.

`Err`
`GSMLibSetDataApplication``(UInt16,`
`  UInt32 creator);`
	This function is used to set the application receiving `kButton` events when the user presses the data button on the VisorPhone. If the application is not found when the event is to be sent, the library resets the creator to the one of the built-in SMS application.

6.2. Controlling the VisorPhone

There are functions that allow you to check the status of the VisorPhone and turn it on or off.

Table 10. Controlling the VisorPhone

Function	Description
`Err` **`GSMLibModulePowered`**`(UInt16);`	This function returns `true` if the VisorPhone is currently powered and `false` if it is not. Note: A return value of `true` does not necessarily mean that the VisorPhone has registered with the GSM network.

`Err`
`GSMLibSetModulePower``(UInt16,`
`  Boolean on);`
	This function may be used to turn the VisorPhone on or off.

`Err` **`GSMLibRegistered`**`(UInt16);`	This function returns `true` if the VisorPhone is current properly registered with the GSM network and `false` if it is not.
`Err` **`GSMLibRoaming`**`(UInt16);`	This function returns `true` if the VisorPhone is currently registered with a foreign network. The client application may display an indication when roaming is active.

6.3. Managing Messages

The SMS library provides functions for creating, modifying, and deleting messages in the message database. Applications should not manipulate the data of a message record stored in the database; they should use the functions provided by the library.

Table 11. Message Management

Function	Description

`GSMDatabaseID`
`GSMLibNewMessage``(UInt16,`
`SMSMessageType type);`
	This function creates a new message of the given message type and returns the ID of the new record in the message database. All fields are either empty (message text) or contain default values (sending options). This function returns either the ID of the new message record, or `GSMkUnknownID` if there was not sufficient memory to allocate a new message.

`Err`
`GSMLibDeleteMessage``(UInt16,`
`GSMDatabaseID msgID,`
`Boolean archive);`
	This function deletes the message with the given `msgID` from the message database. This function returns 0 if there was no error during the deletion of the record. If there is no message with the given `id`, this function returns `gsmErrUnknownID`. If the parameter `archive` is `true`, the message is deleted, but some archive information is kept for use during the next HotSync process.

`Err`
`GSMLibSendMessage``(UInt16,`
`GSMDatabaseID msgID,`
`Boolean substitution);`
	This function sends the message to all the recipients. If there is no message with the given ID, this function returns `gsmErrUnknownID`. If the message's status is not `kNone` or `kPending`, `gsmErrIllegalStatus` is returned. If the error flag is set, it returns `errNotAllowed`. If `substitution` is true, the substitution strings specified in the table 1 are replaced by their equivalent GSM library characters. If the message contains more than 160 characters, this function automatically segments the message.
	
	Note: If a client application wants to send a message that was already sent successfully (i.e., status is `kSent`), it must set the message status to `kPending` using `GSMLibSetStatus`.

6.4. Managing Message Data

Table 12. Message Data Management

Function Name	Description
`Err` **`GSMLibSetText`**`(UInt16, GSMDatabaseID msgID, const char* data, Int16 size);`	This function sets the text of the message to the given text. This function returns 0 if the text was updated successfully. If the text contains illegal characters, it returns the error code `gsmErrIllegalChars`; however, the message text is updated.
`Err` **`GSMLibSetDate`**`(UInt16, GSMDatabaseID msgID, UInt32 date);`	This function sets the date of the given message. The date must be given in Palm OS format.
`Err` **`GSMLibSetOptions`**`(UInt16, GSMDatabaseID msgID, const SMSSendOptions* options);`	This function updates the sending options for the given message. The options can only be set for outgoing messages that have not been sent. This function returns `gsmErrNotAllowed` if it is called on either an incoming message or an outgoing message that has already been sent.
`Err` **`GSMLibSetAddresses`**`(UInt16, GSMDatabaseID msgID, const GSMAddressList list);`	This function updates the list of addresses with the given list of addresses. The list of addresses may only be changed for incoming and unset outgoing messages. This function returns `gsmErrNotAllowed` if it is called for any other type of message.
`Err` **`GSMLibSetStatus`**`(UInt16, GSMDatabaseID msgID, SMSMessageStatus status);`	This function sets the status of the given message. This function returns `gsmErrIllegalStatus` if a given status is not allowed for the message. For example, you cannot set the status to `kSent` for an incoming message. This function is normally not called by applications since the `status` field is used exclusively by the SMS library.
`Err` **`GSMLibSetFlags`**`(UInt16, GSMDatabaseID msgID, UInt32 flags);`	This function updates a message's flags. All flags will be set by this call, so care must be taken in order not to overwrite flags used by the library. This function returns 0 if the flags were set without encountering an error.
`Err` **`GSMLibSetOwner`**`(UInt16, GSMDatabaseID msgID, UInt32 owner);`	This function updates the given message's owner. The owner of a message is notified when a message has been sent or deleted. For a newly created message, the client application must use this function to set the message's owner. There is no default owner assigned by the SMS library. **Note**: If a message's owner is changed, ensure that the former owner has 'released' the message. The former owner should no longer maintain a reference to this message in any data structures. A fake `kMessageDel` event may be used to achieve this.

6.5. Getting Message Data

Table 13. Getting Message Data

Function Name	Description
Err **GSMLibGetText**(UInt16, GSMDatabaseID msgID, char** data);	This function copies the text of the message into a new block of memory that must be disposed of by the caller. This function returns a 0 if the text was retrieved successfully. If the given message is an incoming segmented message that has not been received completely, the library inserts the string "[*part k*]" for every missing part of the message, where *k* is the part's number.
Err **GSMLibGetDate**(UInt16, GSMDatabaseID msgID, UInt32* date);	This function returns the `date` of the given message in Palm OS format. If there was no error while determining the date, this function returns 0.
Err **GSMLibGetOptions**(UInt16, GSMDatabaseID msgID, SMSSendOptions* options);	This function returns the sending options for the given message. This function returns `gsmErrNotAllowed` if it is called on an incoming message.
Err **GSMLibGetAddresses**(UInt16, GSMDatabaseID msgID, GSMAddressList* list);	This function returns the list of addresses for the given message. The list of addresses returned by this function must be disposed of by the caller. The list of addresses returned is either a message's sender (incoming message) or recipients (outgoing message).
Err **GSMLibGetStatus**(UInt16, GSMDatabaseID msgID, SMSMessageStatus* status);	This function returns the `status` field of the message. A client application typically uses this status to map messages to categories.
Err **GSMLibGetFlags**(UInt16, GSMDatabaseID msgID, UInt32* flags);	This function returns the message's flags. Any flags used internally by the SMS library should be treated with care by the client application.
Err **GSMLibGetOwner**(UInt16, GSMDatabaseID msgID, UInt32* owner);	This function returns the message's owner. An application's creator is used to track the owner of a message.
Err **GSMLibGetType**(UInt16, GSMDatabaseID msgID, SMSMessageType* type);	This function returns the message's type. A client application typically uses the type to map messages to categories.

6.6. Managing Addresses and Address Lists

The SMS library provides functions for manipulating addresses and address lists. Applications should not manipulate the data in the addresses directly, but instead should use the functions provided by the library.

Table 14. Address List Management

Function Name	Description

`GSMAddressHandle`
`GSMLibNewAddress``(UInt16,`
`  const char* number,`
`  GSMDatabaseID id);`
	This function creates new address and fills in the information given in `number` and `id`. In the case of an address for an SMS message, the `id` should be set to `GSMkUnknownID`. This function will return a newly allocated handle on the heap, or 0 if there was an error encountered.

`Err`
`GSMLibSetField``(UInt16,`
`  GSMAddressHandle address,`
`  GSMAddressField field,`
`  const char* data);`
	This function lets the specified field of `address` to the given `data`. This function returns 0 if the field was modified without an error.

`char*`
`GSMLibGetField``(UInt16,`
`  GSMAddressHandle address,`
`  GSMAddressField field);`
	This function returns the field's value for a given address in a newly allocated block. This function returns 0 if there was an error while retrieving the data.
	
	Note: The caller of this function must dispose of this block.

`GSMAddressList` **`GSMLibNewAddressList`**`(UInt16);`	This function returns a new address list. The list is initially empty and must be disposed of by the caller.

`Err`
`GSMLibAddAddress``(UInt16,`
`  GSMAddressList list,`
`  const GSMAddressHandle address);`
	This function adds a copy of `address` to the end of `list`. The address is not disposed of.

`Err`
`GSMLibDisposeAddressList``(UInt16,`
`  GSMAddressList list);`
	This function disposes of the memory used by the given address list. The handle should not be used after this function has been called.

`Err`
`GSMLibGetNth``(UInt16,`
`  const GSMAddressList list,`
`  Int16 index,`
`  GSMAddressHandle* address);`
	This function retrieves an item with the given `index` from `list` and returns it as a new block on the heap specified by `address`. The caller is responsible for disposing of the returned block. If an illegal index is given, this function returns `gsmErrIllegalIndex` and sets `address` to 0.
	
	Note: Legal indices are within the range from 1 to *count*, where *count* is the length of the list. Negative indices may

	be used to access items relative to the end of the list (for example, -1 is the last item of the list).
Err **GSMLibSetNth**(UInt16, GSMAddressList list, Int16 index, const GSMAddressHandle address);	This function replaces the data of the item with the given `index` from `list` by the given `address`. If an illegal index is given, this function returns `gsmErrIllegalIndex` and does not modify the address list. If a legal index was given and address is 0, the corresponding item is removed from the list.
Err **GSMLibCount**(UInt16, GSMAddressList list, UInt16* count);	This function returns the length of the given `list`. If `list` is 0, this function returns `gsmErrParam`.

6.7. Manipulating Characters

The SMS library uses the GSM alphabet. Not all characters supported by the Palm OS are supported by the standard GSM alphabet. The SMS library provides functions to test whether a given character is available in the GSM alphabet, and to map a Palm OS character to a GSM character.

The routines are normally used by editors and/or applications used for composing an SMS message. If an incoming message contains GSM characters not supported by the Palm OS, the conversion is handled by the library. The incoming messages are converted to the Palm OS alphabet and Greek characters are replaced by their substitution strings.

Note: The client application should not assume that the number of characters in a message is 160 or less. Even if the incoming message contains 160 characters or less, the message's text may be longer than 160 characters because some characters not available in the Palm OS were replaced by their substitution strings.

Table 15. Manipulating Characters

Function Name	Description

`Boolean`
`GSMLibIsLegalCharacter(UInt16,`
`  char c);`
	This function returns `true` if the given Palm OS character is a legal GSM character. If this function returns `false`, the given character does not exist within the GSM alphabet.

`char`
`GSMLibMapCharacter(UInt16,`
`  char c);`
	This function maps the given Palm OS character to its equivalent in the GSM alphabet. If there is no equivalent character within the GSM alphabet, this function returns 0. The character code returned is a Palm OS character code and not a GSM character code. GSM character codes are used internally by the SMS library only when a message is encoded for transmission. This is transparent to the client application.

`const char*`
`GSMLibGetSubstitution(UInt16,`
`  char c);`
	This function returns the substitution string for a given GSM character. Character `c` is encoded in the GSM alphabet. If there is no substitution string for the given character, this function returns 0.
	
	Note: The string returned by this function must not be disposed of by the caller. The string return is part of the SMS library, and disposing of this string may crash the system

`int`
`GSMLibLength(UInt16,`
`  const char* text,`
`  Boolean inMessages,`
`  Boolean substitution);`
	This function returns the length of the given text in characters or messages. If `inMessages` is `true`, then the value returned will be the number of messages that are required to hold the text. If segmentation is required, the binary segmentation scheme is used. If `substitution` is `true`, then the calculation will include any characters not in the Palm OS alphabet replaced with substitution strings.

6.8. Handling Preferences

Before a message can be sent using the SMS service, the user must set the number of the server center to be used for sending messages. The currently used number may also be of interest, for instance, when the user is modifying the number within a settings dialog.

Table 16. Handling Preferences

Function Name	Description

`Err`
`GSMLibSetServiceCentreAddress``(UInt16,`
`  const GSMAddressHandle address);`
	This function sets the number of the service center to be used for sending the SMS messages.

`Err`
`GSMLibGetServiceCentreAddress``(UInt16,`
`  GSMAddressHandle* address);`
	This function returns the number of the service center currently used for sending messages in a newly allocated address. The caller must dispose of the returned address. The library tries to get the address stored on the SIM using the `+CSCA` command. If there is no current service center, the `address` is set to 0.

6.9. Accessing the Message Database

The client application should use the public ID-based routing of the SMS library to access messages with the message database. For special purposes -- like rebuilding a secondary index -- the SMS library provides two functions to access the message database on the index level.

The usual index-based routines provided by the Palm OS may be used to access the messages in the message database. Every call to `GSMLibGetDBRef` must be matched with a call to `GSMLibReleaseDBRef`. If `GSMLibReleaseDBRef` is not called, the library cannot close the message database. The next HotSync will fail when it tries to read or write SMS messages.

Warning: The client application must not close the database using `DmCloseDatabase` on the handle returned by `GSMLibGetDBRef`. The library assumes that the message database is open until `GSMLibReleaseDBRef` is called. If the message database is closed by some other means, it may crash the system and cause the loss of incoming messages.

Stepping through the message database by index allows an application to "see" records owned by other application. A message owner should be checked carefully to ensure that only messages belonging to the application are being processed. Deleting messages belonging to another application without notifying the application may cause a system to crash when the other application is started the next time.

Table 17. Accessing Message Database

Function Name	Description
`DMOpenRef` **`GSMLibGetDBRef`**`(UInt16);`	This function returns the database handle of the message database. If the database could not be opened by the SMS library, this function returns 0.

`Err`
`GSMLibReleaseDBRef``(UInt16,`
`DMOpenRef database);`
	This function releases the database handle obtained from `GSMLibGetDBRef`. If the handle does not belong to the SMS library, `gsmErrParam` is returned.

7. History

Date	Revision #	Description of changes
25 Apr 2001	1.0	Initial release.