



Winwap Technologies Oy

MMS Stack SDK Library

MMS Stack SDK Library version: 1.0  
WAP specification version: 2.0  
Document dated: 16 April 2009

## NOTICE OF CONFIDENTIALITY

This document contains proprietary and confidential information, belonging to Winwap Technologies Oy.

## REVISION HISTORY

Date	Author	Description
13 October 2004	O Aizatulin	Initial versions of the document.
21 January 2005	K.Sandell	<ul style="list-style-type: none"> <li>Additions, Corrections and document format changes.</li> <li>Corrected mms_decode_sms_message description &amp; error codes</li> <li>Restructured all the API description layouts</li> </ul>
27 January 2005	K.Sandell	Updated mms_set_decoding_version description
28 January 2005	K.Sandell	<ul style="list-style-type: none"> <li>Changed mms_retrieve_msg description</li> <li>Added mmss_init_log API call</li> <li>Added document version information to page headers</li> </ul>
2 February 2005	O Aizatulin	<ul style="list-style-type: none"> <li>MMS message encoding/decoding documentation added</li> <li>Using the MMS Stack section added</li> </ul>
23 March 2005	M Shkirja K.Sandell	<ul style="list-style-type: none"> <li>Style and formatting corrections</li> <li>Spelling errors corrected</li> </ul>
28 May 2007	S Malugin	Function description mms_retrieve_message_ex
28 March 2008	S Malugin	Function description for GPRS session management
12 August 2008	S Malugin	Function description mmss_get_connection_param
21 August 2008	S Malugin	Function description mmss_connect2
16 April 2009	A Brundle	Document review and minor updates.

## TO UNDERSTAND THE CONTENTS OF THIS DOCUMENT

The following addition knowledge is required for clear understanding and evaluating of the information, given in this document:

- Basic knowledge in programming techniques
- Basic understanding of networking connections and client/server architecture where user-agents retrieve and render information (e.g. Internet browsers and servers with services)
- The interaction between a WAP user-agent, a WAP Gateway and a WEB Server
- Basic knowledge of MMS architecture

## Content

<b>1. SCOPE.....</b>	<b>5</b>
<b>2. REFERENCES .....</b>	<b>6</b>
<b>3. MMS STACK LIBRARY OVERVIEW.....</b>	<b>7</b>
<b>4. MMSC COMPATIBILITY .....</b>	<b>8</b>
<b>5. USING THE MMS STACK.....</b>	<b>9</b>
SENDING MMS MESSAGES.....	9
RETRIEVING MMS MESSAGES .....	9
DECODING MMS NOTIFICATION INDICATION SMS MESSAGES .....	10
<b>6. MMS STACK SDK LIBRARY API .....</b>	<b>11</b>
<b>DEFINED VALUES .....</b>	<b>11</b>
FCALLBACK_T.....	11
CONNECTION_T.....	11
<b>MMS STRUCTURES &amp; DEFINITIONS .....</b>	<b>11</b>
MMS_MESSAGE.....	11
MMS_HEADER.....	11
MMS_MESSAGE_TYPE.....	13
MMS_RESPONSE_STATUS.....	14
MMS_YES_NO .....	16
MMS_SENDER_VISIBILITY .....	17
MMS_STATUS.....	17
MMS_RETRIEVE_STATUS .....	18
MMS_READ_STATUS.....	18
MMS_REPLY_CHARGING.....	19
MMS_MM_STATE.....	19
MMS_STORE_STATUS .....	20
ENCODED_STRING.....	20
MMS_CONTENT .....	21
WSP_PUSH_MESSAGE.....	21
MMS_ATTRIBUTES .....	21
MMS_PREVIOUSLY_SENT_BY .....	23
MMS_PREVIOUSLY_SENT_DATE .....	23
<b>MMSS STRUCTURES &amp; DEFINITIONS.....</b>	<b>23</b>
MMSS_ERROR_T.....	23
CONN_MODE_T .....	24
REASON_T.....	25
ADDRESS_INFO_T.....	25
ERROR_INFO_T.....	25
MMSC_REPLY_T.....	26
RESULT_TYPE_T.....	26
REPLY_INFO_T.....	27
CONN_INFO_T .....	27
CONN_INFO2_T .....	28
MMS_INFO_T.....	28
DECODING_VERSION_T.....	29
MMS_URI_T.....	29

<b>MMS STACK SDK API FUNCTIONS.....</b>	<b>30</b>
MMSS_INIT .....	30
MMSS_FINI .....	30
MMSS_CONNECT .....	31
MMSS_CONNECT2 .....	31
MMSS_DISCONNECT .....	32
MMSS_RECONNECT .....	32
MMSS_SEND_MESSAGE .....	33
MMSS_RETRIEVE_MESSAGE.....	34
MMSS_RETRIEVE_MESSAGE_EX .....	35
MMSS_SET_DECODING_VERSION.....	35
MMSS_INIT_LOG .....	36
MMSS_GET_CONNECTION_PARAM .....	37
<b>MMS STACK SDK API – MESSAGE FUNCTIONS .....</b>	<b>37</b>
MMS_MESSAGE_CREATE .....	37
MMS_MESSAGE_DESTROY .....	38
MMS_SET_HEADER_STR .....	39
MMS_SET_HEADER_ENCSTR.....	40
MMS_GET_HEADER_STR.....	41
MMS_GET_HEADER_ENCSTR .....	42
MMS_GET_HEADER_STR_ARRAY .....	43
MMS_GET_HEADER_ENCSTR_ARRAY .....	44
MMS_SET_HEADER_LONG .....	46
MMS_GET_HEADER_LONG.....	47
MMS_MESSAGE_ENCODE .....	48
MMS_MESSAGE_DECODE .....	49
MMS_ADD_CONTENT .....	49
MMS_GET_CONTENT.....	50
MMS_GET_MESSAGE_TYPE.....	51
WSP_PUSH_MESSAGE_INIT .....	52
WSP_PUSH_MESSAGE_RELEASE .....	52
WSP_PUSH_MESSAGE_DECODE.....	53
MMS_DECODE_SMS_MESSAGE .....	54
MMS_GET_DECODING_VERSION .....	55
MMS_SET_DECODING_VERSION .....	55
MMS_ADD_ATTRIBUTES .....	56
MMS_GET_ATTRIBUTES .....	57
<b><u>7. GPRS SESSION MANAGEMENT .....</u></b>	<b><u>59</u></b>
ERROR CODES .....	59
GPRS_DEVICE_ENUM .....	59
GPRS_DEVICE_CONNECT.....	59
GPRS_DEVICE_DISCONNECT .....	60
<b><u>8. SYNCHRONOUS CONNECTION.....</u></b>	<b><u>61</u></b>
<b><u>9. ASYNCHRONOUS CONNECTION.....</u></b>	<b><u>62</u></b>

## 1. Scope

This document contains information about Winwap Technologies Oy MMS Stack library and its API specification.

## 2. References

- [MMSENCAPS] “Wireless Application Protocol, MMS Encapsulation Protocol WAP-209-MMSEncapsulation-20020105-a, WAP Forum, 5-January-2002.  
URL: <http://www.wapforum.org/>
- [MMSENCAPS1\_2] “Multimedia Messaging Service Encapsulation Protocol”, OMA-MMS-ENC-v1\_2-20030915-C, Open Mobile Alliance, 15-September-2003.  
URL: <http://www.openmobilealliance.org/>
- [WAPWPS] “Wireless Application Protocol, Wireless Session Protocol Specification”, WAP-230-WSP Approved Version, 5-July-2001  
URL: <http://www.openmobilealliance.org/>

### 3. MMS Stack Library overview

Winwap Technologies Oy MMS Stack Library allows performing an exchange of MMS messages between MMS client application and Multimedia Messaging Service Center (MMSC). Based on Winwap Technologies Oy MMS Library and Winwap Technologies Oy WAP Stack the library gives a simple way to transfer MMS messages. It supports both [synchronous](#) and [asynchronous](#) connections.

## 4. MMSC Compatibility

The MMS Stack SDK has been tested and verified with the following MMSC vendors and versions:

Vendor	Version	Comments
Tecnomen MMSC	v2.4.3.c-p14	Send / Retrieve fully supported
NowSMS/MMSC		
Ericsson MMSC		
Nokia MMSC		



## 5. Using the MMS Stack

The MMS Stack SDK library API calls are grouped in two naming convention categories.

The categories are MMSS and MMS. The constants and structures beginning with the MMSS name deal with message sending/retrieving to/from the MMSC and the names beginning with MMS deal with MMS message manipulation, such as creation, encoding and content manipulation.

### Sending MMS messages

When sending MMS messages with the API the following steps need to be taken:

1. Initialize the API by calling the [mmss\\_init](#) function
2. Connect to the HTTP Proxy or WAP Gateway using the [mmss\\_connect](#) function
3. Create MMS message of type [SEND\\_REQUEST\\_TYPE](#) by calling [mms\\_message\\_create](#) function
4. Add MMS headers using [mms\\_set\\_header\\_str](#), [mms\\_set\\_header\\_encstr](#) or [mms\\_set\\_header\\_long](#) functions. Mandatory headers [TRANSACTION\\_ID](#), [FROM](#) and [CONTENT\\_TYPE](#) shall be set. Also at least one of the headers [TO](#), [CC](#) or [BCC](#) shall be set. Other headers suitable for this type of message are optional
5. Add content to MMS Message by the help of [mms\\_add\\_content](#) function
6. Call the [mmss\\_send\\_message](#) function
7. Destroy MMS message using [mms\\_message\\_destroy](#) function
8. Disconnect from the HTTP Proxy or WAP Gateway using the [mmss\\_disconnect](#) function
9. Finalize the API by calling the [mmss\\_fini](#) function

If multiple messages are to be sent steps 3 - 7 can be repeated without disconnecting.

### Retrieving MMS messages

For retrieving MMS messages from MMS Center the following API functions shall be called:

1. Initialize the API by calling the [mmss\\_init](#) function
2. Connect to the HTTP Proxy or WAP Gateway using the [mmss\\_connect](#) function
3. Retrieve MMS message by calling the [mmss\\_retrieve\\_message](#) function. To initiate the retrieval activity a message URI delivered in the Notification Indication message shall be used. For more details refer to [Decoding MMS Notification Indication SMS messages](#)
4. Get MMS headers values using functions [mms\\_get\\_header\\_str](#), [mms\\_get\\_header\\_encstr](#), [mms\\_get\\_header\\_str\\_array](#), [mms\\_get\\_header\\_encstr\\_array](#) or [mms\\_get\\_header\\_long](#).
5. Obtain the content of MMS message by the help of [mms\\_get\\_content](#) function
6. Destroy MMS message using [mms\\_message\\_destroy](#) function
7. Disconnect from the HTTP Proxy or WAP Gateway using the [mmss\\_disconnect](#) function
8. Finalize the API by calling the [mmss\\_fini](#) function

If multiple messages are to be retrieved steps 3 - 6 can be repeated without disconnecting. Steps 4 and 5 are optional.

## Decoding MMS Notification Indication SMS messages

MMS Notification Indication message is sent by MMS Center as the message body of WAP PUSH message. Usually MMS Center delivers WAP PUSH message as user data of SMS message encoded in the PDU format. To decode MMS Notification indication SMS message the following steps need to be executed:

1. SMS message shall be saved in the file or files if it is received as concatenated parts.
2. Create MMS message of type **NOTIFICATION\_INDICATION\_TYPE** by calling **mms message create** function
3. Decode SMS message using **mms decode sms message** function
4. Get a location of MMS message to be retrieved calling **mms get header str** function. **MMS\_HEADER** parameter shall be **CONTENT\_LOCATION**
5. Destroy MMS message using **mms message destroy** function

## 6. MMS Stack SDK Library API

This chapter describes available structures and API functions.

### Defined Values

#### **fcallback\_t**

**fcallback\_t** defines the pointer to a callback function which is used in case of [asynchronous](#) connection.

```
typedef void (*fcallback_t)(void *context, const reply_info_t *data);
```

#### **connection\_t**

**connection\_t** is a handle of an established connection. It is returned from a call to [mms\\_connect](#) function.

```
typedef void *connection_t;
```

## MMS Structures & Definitions

### MMS\_MESSAGE

This is a handle that application uses to point the message between library function calls.

### MMS\_HEADER

The following table describes the constants defined for available MMS message headers and their corresponding types. For more detail information refer to [\[MMSENCAPS\]](#) .

The column "Value type" indicates what function shall be used to set or get the header value. In case of Number header [mms\\_set\\_header\\_long](#) and [mms\\_get\\_header\\_long](#) functions shall be used. In case of String header [mms\\_set\\_header\\_str](#) and [mms\\_get\\_header\\_str](#) functions shall be used. In case of Encoded-String header [mms\\_set\\_header\\_encstr](#) and [mms\\_get\\_header\\_encstr](#) functions shall be used. The only exception is the address of the recipient headers (TO, CC and BCC). Any number of addresses is allowed, so special function shall be used: [mms\\_get\\_header\\_str\\_encarray](#).

#### CONSTANTS

Constant name	Description	Value type
<a href="#">BCC</a>	Address of the recipient	Encoded-String
<a href="#">CC</a>	Address of the recipient	Encoded-String
<a href="#">CONTENT_LOCATION</a>	Location of the message	String
<a href="#">CONTENT_TYPE</a>	Content type of the message	String

<b>ARRIVAL_DATE</b>	Arrival time of the message at MMSC	Number
<b>DELIVERY_REPORT</b>	Specifies whether the user wants a delivery report from each recipient.	Number
<b>DELIVERY_TIME</b>	Time of desired delivery.	Number
<b>EXPIRY</b>	Length of time the message will be stored in MMSC time to delete the message.	Number
<b>FROM</b>	Address of the message sender.	String
<b>MESSAGE_CLASS</b>	Class of the message.	Number
<b>MESSAGE_ID</b>	This is a unique reference assigned to message. The ID enables a client to match delivery reports with previously sent messages.	String
<b>MESSAGE_TYPE</b>	Identifies the message type.	Number
<b>MMS_VERSION</b>	The MMS version number.	Number
<b>MESSAGE_SIZE</b>	Full size of message in octets.	Number
<b>PRIORITY</b>	Priority of the message for the recipient.	Number
<b>READ_REPLY</b>	Specifies whether the user wants a read report from each recipient as a new message.	Number
<b>REPORT_ALLOWED</b>	Sending of delivery report allowed to the user or not.	Number
<b>RESPONSE_STATUS</b>	MMS specific status.	Number
<b>RESPONSE_TEXT</b>	Description that qualifies the response status value.	Encoded-String
<b>SENDER_VISIBILITY</b>	Show or not address of the sender to the recipient	Number
<b>STATUS</b>	The status of the message.	Number
<b>SUBJECT</b>	Subject of the message.	Encoded-String
<b>TO</b>	Address of the recipient	Encoded-String
<b>TRANSACTION_ID</b>	A unique identifier for the message.	String
<b>RETRIEVE_STATUS</b>	The status preceding retrieval operation.	Number
<b>RETRIEVE_TEXT</b>	Description which qualifies the retrieve status value.	Encoded-String
<b>READ_STATUS</b>	The status of the message.	Number
<b>REPLY_CHARGING</b>	Indication of the originator will to pay for the reply of the recipient.	Number
<b>REPLY_CHARGING_DEAD_LINE</b>	Specifies the latest time of the recipient(s) to submit the Reply-MM	Number
<b>REPLY_CHARGING_ID</b>	The same value as the Message-ID of the Original-MM that is replied to.	String
<b>REPLY_CHARGING_SIZE</b>	Specifies the maximum size for the Reply-MM.	Number
<b>STORE</b>	Specifies whether the originator MMS Client wants the submitted MM to be saved in the user's MMSBox.	Number

<b>MM_STATE</b>	Specifies the value to set in the MM State field of the stored MM.	Number
<b>STORE_STATUS</b>	Indicates whether the submitted MM was successfully stored into the MMBox.	Number
<b>STORE_STATUS_TEXT</b>	Description that qualifies the Store_Status field value.	Encoded-String
<b>STORED</b>	Indicates whether MM was stored to the user's MMBox.	Number
<b>TOTALS</b>	Gives the total number of messages or bytes in the user's MMBox.	Number
<b>QUOTAS</b>	Gives the quotas defined for the user's MMBox in messages and/or bytes.	Number
<b>MESSAGE_COUNT</b>	Number of messages listed in content of the PDU.	Number
<b>START</b>	A number, indicating the index of the first MM of those selected to have information returned in the response.	Number
<b>DISTRIBUTION_INDICATOR</b>	This field MAY be present for an MM that originated from a Value Added Service Provider and the original included this indicator.	Number
<b>LIMIT</b>	A number indicating the maximum number of selected MMs whose information SHALL be returned in the response.	Number
<b>HEADER_UNKNOWN</b>	The header is unknown. Can be returned after parsing of the received message. Not allowed as the value of the type argument of set or get functions.	

## MMS\_MESSAGE\_TYPE

The following table describes the possible values of the Message-Type header. These constants are used to indicate what message shall be created, or to identify the type of the message.

### CONSTANTS

<b>SEND_REQUEST_TYPE</b>	This kind of messages sent by the client to the MMSC,
<b>SEND_CONFIRMATION_TYPE</b>	When the MMSC has received the Send request, it sends a response message back to the client indicating the status of the operation.
<b>NOTIFICATION_INDICATION_TYPE</b>	Notification indications inform the client about the contents a received message.
<b>NOTIFYRESP_INDICATION_TYPE</b>	When the client has received the Notification indication, it sends Notify response message back to the MMSC in order to acknowledge the transaction to the MMSC.
<b>RETRIEVE_CONFIRMATION_TYPE</b>	When client retrieve the message by sending a WSP/HTTP GET request to the MMSC (containing a URI to the received message) the MMSC response to the request with Retrieve confirmation message.

<b>ACKNOWLEDGE_INDICATION_TYPE</b>	Acknowledge message confirms the delivery of the message from the client to the MMSC.
<b>DELIVERY_INDICATION_TYPE</b>	Delivery Report MUST be sent from the MMSC to the originating client when the originator has requested a delivery report and the recipient has not explicitly requested for denial of the report.
<b>READREC_INDICATION_TYPE</b>	The message is used for the handling Read Reports on the MM recipient side.
<b>READORIG_INDICATION_TYPE</b>	The message is used for the handling Read Reports on the MM originating side.
<b>FORWARD_REQUEST_TYPE</b>	MMS Client sends this message to the MMS Proxy-Relay to request forwarding of an MMS message.
<b>FORWARD_CONFIRMATION_TYPE</b>	When the MMS Proxy-Relay has received the Forward request it SHALL send a confirmation message back to the MMS Client indicating the status of the operation.
<b>MBOX_STORE_REQUEST_TYPE</b>	When MMS Client decides to save a MM that has not been retrieved as yet, it sends this message to the MMS Proxy-Relay.
<b>MBOX_STORE_CONFIRMATION_TYPE</b>	When MMS Proxy-Relay has received the Store request it sends a confirmation to the MMS Client via this message.
<b>MBOX_VIEW_REQUEST_TYPE</b>	MMS Client may request a listing of MM that are stored in the user's MMBBox or obtain information concerning a set of MM known to be stored in the MMBBox.
<b>MBOX_VIEW_CONFIRMATION_TYPE</b>	When MMS Proxy-Relay has received the View request it SHOULD reply with this message.
<b>MBOX_UPLOAD_REQUEST_TYPE</b>	When MMS Client decides to save a MM that is on the MMS Terminal to the user's MMBBox, it transfers a MM using this message.
<b>MBOX_UPLOAD_CONFIRMATION_TYPE</b>	When MMS Proxy-Relay has received the Upload request it sends a confirmation to the MMS Client via this message.
<b>MBOX_DELETE_REQUEST_TYPE</b>	To delete one or more MM from the user's MMBBox the MMS Client SHALL send this message indicating the location reference of all messages to be deleted.
<b>MBOX_DELETE_CONFIRMATION_TYPE</b>	MMS Proxy-Relay SHALL respond to the deletion request with this message indicating if the deletion was successful.
<b>MBOX_DESCRIPTION_TYPE</b>	This PDU SHALL be used by the View confirmation and the Upload request data payload to describe the MM.
<b>UNKNOWN_TYPE</b>	The message type is unknown. Can be returned after parsing of the received message. Not allowed as the value of the type argument of the <a href="#">mms_message_create</a> function. Also can be returned by the <a href="#">mms_get_message_type</a> function.

## MMS\_RESPONSE\_STATUS

Response status header can only appear in the Send confirmation message. It indicates the status of send operation. The following table describes the possible values of the Response-Status header.

### CONSTANTS

<b>STATUS_OK</b>	The corresponding request and some or all of its contents were accepted without errors.
<b>ERROR_UNSPECIFIED</b>	An unspecified error occurred during the processing or reception of the corresponding request
<b>ERROR_SERVICE_DENIED</b>	The client does not have permission or funds to perform the requested operation
<b>ERROR_MESSAGE_FORMAT_CORRUPT</b>	An inconsistency with the message format was detected when the corresponding request was parsed
<b>ERROR_SENDING_ADDRESS_UNRESOLVED</b>	There were no MMS address (From:, To:, Cc:, Bcc:) in its proper format or none of the addresses belong to the MMSC.
<b>ERROR_MESSAGE_NOT_FOUND</b>	This status code is obsolete
<b>ERROR_NETWORK_PROBLEM</b>	The MMSC was not able to accept the corresponding request due to capacity overload
<b>ERROR_CONTENT_NOT_ACCEPTED</b>	The MM content was not accepted due to size, media type, copyrights or some other reason.
<b>ERROR_UNSUPPORTED_MESSAGE</b>	The MMSC does not support the corresponding request abstract message.
<b>ERROR_TRANSIENT_FAILURE</b>	The corresponding M-Send.req as received was valid and understood by the MMS Proxy-Relay, but some temporary condition or event caused an error to occur.
<b>ERROR_TRANSIENT_SENDING_ADDRESS_UNRESOLVED</b>	This X-Mms-Response-Status value SHOULD not be used in the M-Send.conf PDU.
<b>ERROR_TRANSIENT_MESSAGE_NOT_FOUND</b>	This X-Mms-Response-Status value SHOULD not be used in the M-Send.conf PDU.
<b>ERROR_TRANSIENT_NETWORK_PROBLEM</b>	The MMS Proxy-Relay was not able to handle the corresponding MSend.req due to unspecified error on the transport layer or capacity overload.
<b>ERROR_TRANSIENT_PARTIAL_SUCCESS</b>	This X-Mms-Response-Status value SHOULD not be used in the M-Send.conf PDU.
<b>ERROR_PERMANENT_FAILURE</b>	An unspecified permanent error occurred during the processing or reception of the corresponding MSend.req.
<b>ERROR_PERMANENT_SERVICE_DENIED</b>	The corresponding M-Send.req was rejected due to failure of authentication or authorization due to different reasons of the originating MMS Client.
<b>ERROR_PERMANENT_MESSAGE_FORMAT_CORRUPT</b>	An inconsistency with the formats for optional or mandatory header fields or an error with header field values was detected when the corresponding M-Send.req was parsed.
<b>ERROR_PERMANENT_SENDING_ADDRESS_UNRESOLVED</b>	The MMS Proxy-Relay was not able to resolve the insert-address-token into a valid sending address.
<b>ERROR_PERMANENT_MESSAGE_NOT_FOUND</b>	This X-Mms-Response-Status value SHOULD not be used in the M-Send.conf PDU.



<b>ERROR_PERMANENT_CONTENT_NOT_ACCEPTED</b>	The MM content in the M-Send.req was not accepted due to size, media type, copyrights or some other reason.
<b>ERROR_PERMANENT_REPLY_CHARGING_LIMITATIONS_NOT_MET</b>	The corresponding request contained a reply MM that was too large, not within the reply charging deadline and/or contained non-text media elements when only text was allowed.
<b>ERROR_PERMANENT_REPLY_CHARGING_REQUEST_NOT_ACCEPTED</b>	The M-Send.req contained an XMms-Reply-Charging header field with the value 'Accepted' or 'Accepted text only'.
<b>ERROR_PERMANENT_REPLY_CHARGING_FORWARDING_DENIED</b>	This X-Mms-Response-Status value SHOULD not be used in the M-Send.conf PDU.
<b>ERROR_PERMANENT_REPLY_CHARGING_NOT_SUPPORTED</b>	The MMS Proxy-Relay does not support reply charging. The corresponding M-Send.req contained reply-charging parameters and was rejected.
<b>ERROR_PERMANENT_ADDRESS_HIDING_NOT_SUPPORTED</b>	The MMS Proxy-Relay does not support address hiding. The corresponding M-Send.req had the XMms-Sender-Visibility set to 'Hide' and was thus rejected.
<b>RESPONSE_STATUS_UNKNOWN</b>	The response status is unknown. Can be returned after parsing of the received message as the <b>value</b> argument of the <a href="#">mms_get_header_long</a> function. Not allowed as the <b>value</b> of the <b>value</b> argument of the <a href="#">mms_set_header_long</a> function.

## MMS\_YES\_NO

Values for Yes, No headers such as Delivery-Report, Read-Reply, Report-Allowed

### CONSTANTS

**YES**

**NO**

**MMS\_MESSAGE\_CLASS**This header describes the class of the message. The following table describes the possible values of the Message-Class header.

### CONSTANTS

<b>PERSONAL</b>	The message is personal
<b>ADVERTISEMENT</b>	The message is advertisement
<b>INFORMATIONAL</b>	The message is informational
<b>CLASS_UNKNOWN</b>	The message class is unknown. Can be returned after parsing of the received message as the <b>value</b> argument of the <a href="#">mms_get_header_long</a> function. Not allowed as the <b>value</b> of the <b>value</b> argument of the <a href="#">mms_set_header_long</a> function.

**MMS\_PRIORITY**This header indicates the priority (importance) of the message. The following table describes the possible values of the Priority header.

### CONSTANTS

<b>LOW</b>	The message has low priority
------------	------------------------------



<b>NORMAL</b>	The message has normal priority
<b>HIGH</b>	The message has high priority
<b>PRIORITY_UNKNOWN</b>	The message priority is unknown. Can be returned after parsing of the received message as the <b>value</b> argument of the <a href="#">mms_get_header_long</a> function. Not allowed as the value of the <b>value</b> argument of the <a href="#">mms_set_header_long</a> function.

## MMS\_SENDER\_VISIBILITY

Header indicates a request to show or hide the sender's identity when the message is delivered to the recipient. The following table describes the possible values of the Sender-Visibility header.

### CONSTANTS

<b>HIDE</b>	Do not show the sender's identity
<b>SHOW</b>	Show the sender's identity
<b>VISIBILITY_UNKNOWN</b>	The sender visibility is unknown. Can be returned after parsing of the received message as the <b>value</b> argument of the <a href="#">mms_get_header_long</a> function. Not allowed as the value of the <b>value</b> argument of the <a href="#">mms_set_header_long</a> function.

## MMS\_STATUS

The header indicates status of the MMS message. The following table describes the possible values of the Status header.

### CONSTANTS

<b>EXPIRED</b>	Message has been expired
<b>RETRIEVED</b>	Message has been retrieved
<b>REJECTED</b>	Message has been rejected
<b>DEFERRED</b>	Message has been deferred
<b>UNRECOGNISED</b>	Status not available
<b>INDETERMINATE</b>	It is not possible to determine if the MM in question reached its destination.
<b>FORWARDED</b>	The MM in question reached the recipient MS Proxy-Relay but the recipient MMS Client forwarded it without retrieving it first.
<b>UNREACHABLE</b>	The recipient MMS Client is not reachable due to MMS Proxy-Relay interworking, routing failure or some other condition that prevents the MM to reach it's final destination.
<b>STATUS_UNKNOWN</b>	The status is unknown. Can be returned after parsing of the received message as the <b>value</b> argument of the <a href="#">mms_get_header_long</a> function. Not allowed as the value of the <b>value</b> argument of the <a href="#">mms_set_header_long</a> function.

## MMS\_RETRIEVE\_STATUS

Retrieve status header can only appear in the Retrieve confirmation message. It indicates errors, if any that occurred during the preceding retrieval operation. The following table describes the possible values of the Retrieve Status header.

### CONSTANTS

<a href="#">STATUS_OK_RET</a>	The retrieve request was accepted and processed without any errors.
<a href="#">ERROR_TRANSIENT_FAILURE_RET</a>	The corresponding retrieve request as received was valid and understood by the MMS Proxy-Relay, but some temporary condition or event caused an error to occur.
<a href="#">ERROR_TRANSIENT_MESSAGE_NOT_FOUND_RET</a>	The MMS Proxy-Relay has temporarily lost contact with where the MM's are stored, e.g. the MMS Server.
<a href="#">ERROR_TRANSIENT_NETWORK_PROBLEM_RET</a>	The MMS Proxy-Relay was not able to handle the corresponding retrieve request due to capacity overload.
<a href="#">ERROR_PERMANENT_FAILURE_RET</a>	An unspecified permanent error occurred during the processing of the retrieval attempt of the MM in question.
<a href="#">ERROR_PERMANENT_SERVICE_DENIED_RET</a>	The corresponding retrieval attempt was rejected due to failure of authentication or authorization of the originating MMS Client.
<a href="#">ERROR_PERMANENT_MESSAGE_NOT_FOUND_RET</a>	The content location URL in the retrieval attempt does not point to an MM.
<a href="#">ERROR_PERMANENT_CONTENT_UNSUPPORTED_RET</a>	The MM that the retrieval attempt referred to contained media elements that the recipient MMS Client cannot handle, and the recipient MMS Proxy-Relay cannot perform the proper content adaptation.
<a href="#">RETRIEVE_STATUS_UNKNOWN</a>	The status is unknown. Can be returned after parsing of the received message as the <b>value</b> argument of the <a href="#">mms_get_header_long</a> function. Not allowed as the value of the <b>value</b> argument of the <a href="#">mms_set_header_long</a> function.

## MMS\_READ\_STATUS

Read status header must be included in the Read recipient indication and Read originator indication messages. It is used to convey information from the recipient MMS Client to the originating MMS Client whether an MM, for which a read report was requested, was read or not. The following table describes the possible values of the Read Status header.

### CONSTANTS

<a href="#">READ</a>	Some or all of the multimedia contents of the MM in question was rendered or played by the recipient terminal.
----------------------	--

**DELETED\_WITHOUT\_BEING\_READ**

The MM in question was retrieved by the MS Client but deleted without any of its contents being rendered or played by the recipient terminal.

**READ\_STATUS\_UNKNOWN**

The status is unknown. Can be returned after parsing of the received message as the **value** argument of the [mms\\_get\\_header\\_long](#) function. Not allowed as the value of the **value** argument of the [mms\\_set\\_header\\_long](#) function.

**MMS\_REPLY\_CHARGING**

This header field SHALL only be present if the originator is willing to pay for the Reply-MM of the recipient(s). The following table describes the possible values of the Reply Charging header.

**CONSTANTS****REQUESTED**

The originator will pay for the reply-MM of the recipient(s).

**REQUESTED\_TEXT\_ONLY**

The originator will pay for the text only reply-MM of the recipient(s).

**ACCEPTED**

MMS Service provider accepts the reply charging.

**ACCEPTED\_TEXT\_ONLY**

MMS Service provider accepts the text only reply charging.

**REPLY\_CHARGING\_UNKNOWN**

The value is unknown. Can be returned after parsing of the received message as the **value** argument of the [mms\\_get\\_header\\_long](#) function. Not allowed as the value of the **value** argument of the [mms\\_set\\_header\\_long](#) function.

**MMS\_MM\_STATE**

This header field SHALL indicate the current stored state of the MM. The following table describes the possible values of the MM State header.

**CONSTANTS****MM\_DRAFT****MM\_SENT****MM\_NEW****MM\_RETRIEVAL****MM\_FORWARDED****MM\_STATE\_UNKNOWN**

The value is unknown. Can be returned after parsing of the received message as the **value** argument of the [mms\\_get\\_header\\_long](#) function. Not allowed as the value of the **value** argument of the [mms\\_set\\_header\\_long](#) function.

## MMS\_STORE\_STATUS

Store status header is used by the MMS Proxy Relay to inform the MMS Client, which has requested that a MM be copied or moved to the user's MMBox, the result of the storing action. The following table describes the possible values of the Store Status header.

### CONSTANTS

<a href="#">STATUS_OK_STR</a>	The store request was accepted and processed without any errors.
<a href="#">ERROR_TRANSIENT_FAILURE_STR</a>	The corresponding store request as received was valid and understood by the MMS Proxy-Relay, but some temporary condition or event caused an error to occur.
<a href="#">ERROR_TRANSIENT_NETWORK_PROBLEM_STR</a>	The MMS Proxy-Relay was not able to handle the corresponding store request due to capacity overload.
<a href="#">ERROR_PERMANENT_FAILURE_STR</a>	An unspecified permanent error occurred during the processing or reception of the corresponding request.
<a href="#">ERROR_PERMANENT_SERVICE_DENIED_STR</a>	The corresponding storage request was rejected due to failure of authentication or authorization due to store the MM into the MMBox.
<a href="#">ERROR_PERMANENT_MESSAGE_FORMAT_CORRUPT_STR</a>	This X-Mms-Store-Status value SHOULD not be used in the M-Send.conf or M-orward.conf PDU.
<a href="#">ERROR_PERMANENT_MESSAGE_NOT_FOUND_STR</a>	This X-Mms-Store-Status value SHOULD not be used in the M-Send.conf or M-orward.conf PDU.
<a href="#">ERROR_PERMANENT_MMBOX_FULL_STR</a>	The corresponding storage request failed since the user's MMBox quota was exhausted.
<a href="#">STORE_STATUS_UNKNOWN</a>	The status is unknown. Can be returned after parsing of the received message as the <b>value</b> argument of the <a href="#">mms_get_header_long</a> function. Not allowed as the value of the <b>value</b> argument of the <a href="#">mms_set_header_long</a> function.

## ENCODED\_STRING

[ENCODED\\_STRING](#) specifies an encoded string.

```
typedef struct _ENCODED_STRING
{
    const char *str,
    size_t      strlen,
    unsigned    charset,
} ENCODED_STRING;
```

Field	Description
<a href="#">str</a>	String data
<a href="#">strlen</a>	Length of the string data

<b>charset</b>	Char-set of the string. Char-set values registered by IANA as MIBEnum value.
----------------	--

## MMS\_CONTENT

```
typedef struct
_MMS_CONTENT
{
    char *content_type;
    char *headers;
    void *data;
    size_t data_size;
} MMS_CONTENT;
```

The members of the MMS\_CONTENT structure are:

Field	Description
<b>content_type</b>	The content type of the content.
<b>headers</b>	The headers associated with content.
<b>data</b>	The pointer to memory buffer where content is placed.
<b>data_size</b>	The size of the memory buffer pointed by <b>data</b> .

## WSP\_PUSH\_MESSAGE

```
typedef struct _WSP_PUSH_MESSAGE
{
    const char *content_type;
    const char *headers;
    unsigned char transaction_id;
} WSP_PUSH_MESSAGE;
```

The members of the WSP\_PUSH\_MESSAGE structure are:

Field	Description
<b>content_type</b>	The content type of the WSP Push message.
<b>headers</b>	The headers associated with content of the WSP Push message.
<b>transaction_id</b>	The transaction ID of the WSP Push message.

## MMS\_ATTRIBUTES

```
typedef struct _MMS_ATTRIBUTES
{
    ENCODED_STRING *bcc;
    size_t bcc_size;
    ENCODED_STRING *cc;
    size_t cc_size;
    MMS_CONTENT *content;
    size_t content_size;
    char *content_type;
```

```

long date;
unsigned char delivery_report;
long delivery_time;
long expiry;
ENCODED_STRING from;
unsigned char message_class;
char *message_id;
long message_size;
unsigned char priority;
unsigned char read_report;
ENCODED_STRING subject;
ENCODED_STRING *to;
size_t to_size;
unsigned char reply_charging;
char *reply_charging_id;
unsigned char reply_charging_deadline;
long reply_charging_size;
MMS_PREVIOUSLY_SENT_BY previously_sent_by;
MMS_PREVIOUSLY_SENT_DATE previously_sent_date;
char **additional_headers;
size_t additional_headers_size;
} MMS_ATTRIBUTES;

```

Field	Description
<b>bcc</b>	Bcc field value.
<b>bcc_size</b>	The size of Bcc field value.
<b>cc_size</b>	Cc field value.
<b>cc_size</b>	The size of Cc field value.
<b>content</b>	Content field value.
<b>content_size</b>	The size of Content field value.
<b>content_type</b>	Content-Type field value.
<b>date</b>	Date field value.
<b>delivery_report</b>	X-Mms-Delivery-Report field value.
<b>delivery_time</b>	X-Mms-Delivery-Time field value.
<b>expiry</b>	X-Mms-Expiry field value.
<b>from</b>	From field value.
<b>message_class</b>	X-Mms-Message-Class field value.
<b>message_id</b>	X-Mms-Message-Id field value.
<b>message_size</b>	X-Mms-Message-Size field value.
<b>priority</b>	X-Mms-Priority field value.
<b>read_report</b>	X-Mms-Read-Report field value.
<b>subject</b>	Subject field value.
<b>to</b>	To field value.
<b>to_size</b>	The size of To field value.
<b>reply_charging</b>	X-Mms-Reply-Charging field value.

<code>reply_charging_id</code>	X-Mms-Reply-Charging-Id field value.
<code>reply_charging_deadline</code>	X-Mms-Reply-Charging-Deadline field value.
<code>reply_charging_size</code>	X-Mms-Reply-Charging-Size field value.
<code>previously_sent_by</code>	X-Mms-Previously-Sent-By field value.
<code>previously_sent_date</code>	X-Mms-Previously-Sent-Date field value.
<code>additional_headers</code>	Additional headers field value.
<code>additional_headers_size</code>	The size of Additional headers field value.

## MMS\_PREVIOUSLY\_SENT\_BY

```
typedef struct _MMS_PREVIOUSLY_SENT_BY
{
    int            forwarded_count;
    ENCODED_STRING str;
} MMS_PREVIOUSLY_SENT_BY;
```

## MMS\_PREVIOUSLY\_SENT\_DATE

```
typedef struct _MMS_PREVIOUSLY_SENT_DATE
{
    int    forwarded_count;
    long   date;
} MMS_PREVIOUSLY_SENT_DATE;
```

## MMSS Structures & Definitions

### mmss\_error\_t

**mmss\_error\_t** includes the list of possible errors that can be returned by the library API functions.

```
typedef enum mmss_error_
{
    mmss_success,
    mmss_invalid_arg,
    mmss_cant_open_wps,
    mmss_cant_bind_wps,
    mmss_cant_connect,
    mmss_timeout,
    mmss_msg_encode_err,
    mmss_msg_post_err,
    mmss_msg_get_err,
    mmss_bad_content,
    mmss_bad_decoding_ver,
    mmss_msg_decode_err,
    mmss_unexpected_msg,
    mmss_cant_init_log
} mmss_error_t;
```

Field	Description
<code>mmss_success</code>	Function has performed its function without errors.
<code>mmss_invalid_arg</code>	One of the passed arguments is invalid.
<code>mmss_cant_open_wps</code>	Can not create WAP Stack.
<code>mmss_cant_bind_wps</code>	Can not bind WAP Stack to a custom system network address.
<code>mmss_cant_connect</code>	Can not establish an asynchronous connection to the WAP gateway in case of connection-oriented (CO), secure connectionless (SCL) and secure connection-oriented (SCO) modes.  Can not assign WAP gateway addresses in case of connectionless (CL) mode.  Can not assign HTTP Proxy address in case of HTTP and HTTPS modes.
<code>mmss_timeout</code>	Function exits by timeout.
<code>mmss_msg_encode_err</code>	Function can not encode MMS message.
<code>mmss_msg_post_err</code>	Failed to send a POST request to WAP Gateway or HTTP Proxy.
<code>mmss_msg_get_err</code>	Failed to send a GET request to WAP Gateway or HTTP Proxy.
<code>mmss_bad_content</code>	WAP Gateway or HTTP Proxy reply is not an MMS message.
<code>mmss_bad_decoding_ver</code>	Function can not set the MMS message decoding version for WAP WSP.
<code>mmss_msg_decode_err</code>	Function can not decode MMS message.
<code>mmss_unexpected_msg</code>	WAP Gateway or HTTP Proxy reply is an MMS message of unknown type.
<code>mmss_cant_init_log</code>	Failed to initialize a log file.

## conn\_mode\_t

**conn\_mode\_t** contains WAP connection modes.

```
typedef enum
{
    mode_wap_cl,
    mode_wap_co,
    mode_wap_scl,
    mode_wap_sco,
    mode_http,
    mode_https,
} conn_mode_t;
```

Field	Description
<code>mode_wap_cl</code>	Connection less mode
<code>mode_wap_co</code>	Connection oriented mode
<code>mode_wap_scl</code>	Secure connection less mode
<code>mode_wap_sco</code>	Secure connection oriented mode



<code>mode_http</code>	HTTP mode
<code>mode_https</code>	Secure HTTP mode, using SSL

## reason\_t

**reason\_t** indicates the reason of callback function calling.

```
typedef enum
{
    reason_connect_ok,
    reason_connect_redirect,
    reason_connect_error,
    reason_reply,
    reason_request_error,
} reason_t;
```

Field	Description
<code>reason_connect_ok</code>	Connection is successful
<code>reason_connect_redirect</code>	Redirection request received
<code>reason_connect_error</code>	Connection error
<code>reason_reply</code>	MMS Proxy/Relay replay received
<code>reason_request_error</code>	Send request error

## address\_info\_t

**address\_info\_t** consists of server address and port.

```
typedef struct
{
    const char *addr;
    int port;
} address_info_t;
```

Field	Description
<code>addr</code>	WAP Gateway or HTTP Proxy address.
<code>port</code>	WAP Gateway or HTTP Proxy port.

## error\_info\_t

**error\_info\_t** describes an error.

```
typedef struct
{
    const char *descr;
    int code;
} error_info_t;
```

Field	Description
descr	Contains an error description.
code	Contains an error code.

## mmsc\_reply\_t

**mmsc\_reply\_t** contains MMSC reply, if it was done in the format another than MMS message.

```
typedef struct mmsc_reply_
{
    int      status;
    char     *headers;
    char     *content_type;
    void     *content;
    size_t   content_size;
} mmsc_reply_t;
```

Field	Description
status	Status code. Possible codes are: WAP Gateway, HTTP Proxy or HTTP server status codes.
headers	Data headers
content_type	Data content type
content	Data content pointer
content_size	Size of content data in bytes

## result\_type\_t

**result\_type\_t** determines type of a received result.

```
typedef enum result_type_
{
    dt_mmsc_reply,
    dt_mms_message,
    dt_redirection,
    dt_error
} result_type_t;
```

Field	Description
dt_mmsc_reply	MMSC reply is received in the format another than MMS message.
dt_mms_message	MMSC reply is an MMS message.
dt_redirection	MMSC reply is a redirection request.
dt_error	An error occurs on request to MMSC.

## reply\_info\_t

**reply\_info\_t** contains information about the MMSC reply.

```
typedef struct
{
    Void                *context;
    reason_t            reason;
    result_type_t       type;
    union
    {
        mmsc_reply_t    reply;
        MMS_MESSAGE     msg;
        address_info_t  redirection;
        error_info_t    error;
    } result;
} reply_info_t;
```

Field	Description
context	Application defined value
reason	Reason for callback function. Please see <a href="#">reason_t</a> for values
type	The type of the result. Please see <a href="#">result_type_t</a> for values
result	This value depends on the type field value. reply - MMS Proxy reply data, see <a href="#">mms_reply_t</a> for values msg - MMS Proxy/Relay reply as mms-message redirection - Redirection info. Please see <a href="#">address_info_t</a> for values error - Error description. Please see <a href="#">error_info_t</a> for values

## conn\_info\_t

**conn\_info\_t** provides values to establish connection.

```
typedef struct
{
    conn_mode_t         conn_mode;
    char                *addr;
    unsigned short      port;
    char                *user_name;
    char                *password;
    char                *headers;
    fcallback_t         callback;
} conn_info_t;
```

Field	Description
conn_mode	Connection mode to use. See <a href="#">conn_mode_t</a> for values
addr	WAP Gateway or HTTP Proxy IP/Hostname
port	HTTP Proxy port if HTTP mode is used. Otherwise set to 0

<code>user_name</code>	The username to use when connecting
<code>password</code>	The password to use when connecting
<code>headers</code>	WSP session headers in WSP mode
<code>callback</code>	Application defined callback. If set, <a href="#">Asynchronous</a> mode is used.

## conn\_info2\_t

[conn\\_info2\\_t](#) provides values to establish connection.

```
typedef struct
{
    unsigned int    struct_size;
    conn_mode_t    conn_mode;
    char           *addr;
    unsigned short port;
    char           *user_name;
    char           *password;
    char           *headers;
    fcallback_t    callback;
    const char     *local_addr;
    unsigned short local_port;
} conn_info2_t;
```

Field	Description
<code>struct_size</code>	Specifies the length, in bytes, of the structure
<code>conn_mode</code>	Connection mode to use. See <a href="#">conn_mode_t</a> for values
<code>addr</code>	WAP Gateway or HTTP Proxy IP/Hostname
<code>port</code>	HTTP Proxy port if HTTP mode is used. Otherwise set to 0
<code>user_name</code>	The username to use when connecting
<code>password</code>	The password to use when connecting
<code>headers</code>	WSP session headers in WSP mode
<code>callback</code>	Application defined callback. If set, <a href="#">Asynchronous</a> mode is used.
<code>local_addr</code>	IP address of the available local system network interface. The MMSStack will bind to all available system network interfaces if the parameter is NULL.
<code>local_port</code>	The parameter sets the correct listening port number. It is applicable only for the connectionless and the HTTP connection mode. The port value is ignored for other connection modes and instead a port number provided by the system will be used.

## mms\_info\_t

[mms\\_info\\_t](#) defines necessary data allowing sending MMS message.

```
typedef struct
{
    char        *mmsc_uri;
    char        *user_name;
    char        *password;
    char        *headers;
    MMS_MESSAGE msg;
} mms_info_t;
```

Field	Description
mmsc_uri	Fully qualified URI to MMSC
user_name	Username to use when sending
password	Password to use when sending
headers	Additional headers
msg	Pointer to MMS message to send

## decoding\_version\_t

**decoding\_version\_t** defines MMS headers decoding version.

```
typedef enum decoding_version_
{
    decoding_version_12 = 0x12,
    decoding_version_13,
    decoding_version_14
} decoding_version_t;
```

Field	Description
decoding_version_12	Headers decoding 1.2
decoding_version_13	Headers decoding 1.3
decoding_version_14	Headers decoding 1.4

## mms\_uri\_t

**decoding\_version\_t** defines MMS headers decoding version.

```
typedef struct
{
    const char *msg_uri;
    const char *user_name;
    const char *password;
    const char *headers;
} mms_uri_t;
```

## MMS Stack SDK API Functions

### mmss\_init

**Description** [mmss\\_init](#) initializes the MMS Stack library.

**Synopsis C/C++**

```
#include "mmslib.h"  
#include "mmsstack.h"  
  
mms_error_t MMSSTACK_API mmss_init();
```

**Parameters** There are no parameters for this function.

**Return value** On success, [mmss\\_init](#) returns [mms\\_success](#)

**Errors** [mms\\_success](#)

**See also** [mms\\_fini](#)

### mmss\_fini

**Description** [mmss\\_fini](#) releases an initialization data.

**Synopsis C/C++**

```
#include "mmslib.h"  
#include "mmsstack.h"  
  
void MMSSTACK_API mmss_fini();
```

**Parameters** There are no parameters for this function.

**Return value** On success, [mmss\\_init](#) returns [mms\\_success](#)

**Errors** [mms\\_success](#)

**See also** [mms\\_init](#)

## mmss\_connect

**Description** [mmss\\_connect](#) establishes a connection with a MMSC. This function must be called prior to sending or retrieving MMS messages from a MMSC.

**Synopsis C/C++**

```
#include "mmslib.h"
#include "mmsstack.h"

mms_error_t MMSSTACK_API mmss_connect( const conn_info_t
*ci, connection_t *conn, void *context);
```

**Parameters**

`ci` [in] – pointer to the [conn\\_info\\_t](#) structure  
`conn` [out] – pointer to the [connection\\_t](#) connection handle  
`context` [in] – application defined value.

**Return value** If [mmss\\_success](#) is returned, `conn` defines a connection handle  
On error, one of the error codes listed below is returned.

**Errors**

- [mmss\\_invalid\\_arg](#) – One of the passed arguments has an invalid value
- [mmss\\_cant\\_open\\_wps](#) - Can not open WAP Stack object
- [mmss\\_cant\\_bind\\_wps](#) - Can not bind WAP Stack object
- [mmss\\_cant\\_connect](#) - Can not establish connection
- [mmss\\_timeout](#) - Can not establish connection, exit by timeout

**See also** [mmss\\_disconnect](#)

## mmss\_connect2

**Description** [mmss\\_connect2](#) establishes a connection with a MMSC. This function must be called prior to sending or retrieving MMS messages from a MMSC.

**Synopsis C/C++**

```
#include "mmslib.h"
#include "mmsstack.h"

mms_error_t MMSSTACK_API mmss_connect2( const
conn_info2_t *ci, connection_t *conn, void *context);
```

**Parameters**

`ci` [in] – pointer to the [conn\\_info2\\_t](#) structure  
`conn` [out] – pointer to the [connection\\_t](#) connection handle

`context` [in] – application defined value.

- Return value** If [mmss\\_success](#) is returned, `conn` defines a connection handle  
On error, one of the error codes listed below is returned.
- Errors**
- [mmss\\_invalid\\_arg](#) – One of the passed arguments has an invalid value
  - [mmss\\_cant\\_open\\_wps](#) - Can not open WAP Stack object
  - [mmss\\_cant\\_bind\\_wps](#) - Can not bind WAP Stack object
  - [mmss\\_cant\\_connect](#) - Can not establish connection
  - [mmss\\_timeout](#) - Can not establish connection, exit by timeout
- See also** [mmss\\_disconnect](#)

## mmss\_disconnect

- Description** [mmss\\_disconnect](#) disconnects from a MMSC that has been connected using the [mmss\\_connect](#) function.
- Synopsis C/C++**
- ```
#include "mmslib.h"
#include "mmsstack.h"

mms_error_t MMSSTACK_API mmss_disconnect( connection_t
conn);
```
- Parameters** `conn` [in] – the [connection\\_t](#) handle from [mmss\\_connect](#)
- Return value** If [mmss\\_success](#) is returned, disconnect from MMSC was successful. On error, one of the error codes listed below is returned.
- Errors** [mmss\\_success](#) – Disconnected successfully from the MMSC
- See also** [mmss\\_connect](#)

## mmss\_reconnect

- Description** [mmss\\_reconnect](#) reconnects to a MMSC. This function can be used to make sure the connection to the MMSC is still active.



This function is also used to connect to the new MMSC when the [Asynchronous](#) connection mode receives a [dt redirection](#) reply from a MMSC in the callback function.

**Synopsis C/C++**

```
#include "mmslib.h"
#include "mmsstack.h"

mms_error_t MMSSTACK_API mms_reconnect( connection_t
conn, const char *ga, int gp);
```

**Parameters**

**conn** [in] – pointer to the [connection\\_t](#) handle from [mms\\_connect](#)

**ga** [in] – Pointer to the gateway IP / Hostname

**gp** [in] – The gateway port number

**Return value** If [mms\\_success](#) is returned the reconnection was successful

**Errors**

[mms\\_success](#) – Reconnected successfully to the MMSC

[mms\\_cant\\_connect](#) - Can not establish connection

**See also** [mms\\_connect](#), [mms\\_disconnect](#)

## mmss\_send\_message

**Description** [mmss\\_send\\_message](#) sends a MMS message to the MMSC for delivery. The **minfo** structure member **MMSC\_URI** must point to the absolute URI for the MMSC.

**Synopsis C/C++**

```
#include "mmslib.h"
#include "mmsstack.h"

mms_error_t MMSSTACK_API mmss_send_message( connection_t
conn, const mms_info_t *minfo, reply_info_t *reply);
```

**Parameters**

**conn** [in] – pointer to the [connection\\_t](#) handle from [mms\\_connect](#)

**minfo** [in] – pointer to the MMS message structure [mms\\_info\\_t](#)

**reply** [out] – pointer to the reply information structure [reply\\_info\\_t](#)

**Return value** If [mms\\_success](#) is returned the **reply** structure will contain information returned from the MMSC

On error, one of the error codes listed below is returned

**Errors**

- [mmss\\_success](#) – Message successfully sent to the MMSC
- [mms\\_msg\\_post\\_err](#) - Failed to send the message to the MMSC
- [mmss\\_timeout](#) – A timeout occurred when trying to send the message

**See also** [mmss\\_retrieve\\_message](#)

## mmss\_retrieve\_message

**Description** [mmss\\_retrieve\\_message](#) retrieves a MMS message from the MMSC. The URI of the message to retrieve must be known.

On a successful retrieval the pmsg MMS message must be freed using the `mms_message_destroy` function.

**Synopsis C/C++**

```
#include "mmslib.h"
#include "mmsstack.h"

mms_error_t MMSSTACK_API mms_retrieve_message(
connection_t conn, const char *msg_uri, reply_info_t
*reply);
```

**Parameters**

- `conn` [in] – pointer to the [connection\\_t](#) handle from [mmss\\_connect](#)
- `msg_uri` [in] – URI to MMS message to retrieve
- `reply` [out] – pointer to the reply information structure [reply\\_info\\_t](#) in [synchronous](#) mode. NIL if [asynchronous](#) mode is used.

**Return value**

If [mmss\\_success](#) is returned `reply` will contain the pointer to the [reply\\_info\\_t](#) structure.

On error, one of the error codes listed below is returned

**Errors**

- [mmss\\_success](#) – Message successfully sent to the MMSC
- [mmss\\_invalid\\_arg](#) – One of the arguments passed is invalid
- [mmss\\_msg\\_get\\_err](#) – The MMS message was not found in the MMSC
- [mmss\\_timeout](#) – A timeout occurred when trying to retrieve the message
- [mmss\\_bad\\_content](#) – The retrieved data is not a valid MMS message
- [mmss\\_unexpected\\_data](#) – The retrieved message has an unknown type

**See also** [mmss\\_send\\_message](#)

## mmss\_retrieve\_message\_ex

**Description** [mmss retrieve message ex](#) retrieves a MMS message from the MMSC. The URI of the message to retrieve must be known.

On a successful retrieval the pmsg MMS message must be freed using the [mms\\_message\\_destroy](#) function.

**Synopsis C/C++**

```
#include "mmslib.h"
#include "mmsstack.h"

mms_error_t MMSSTACK_API mms_retrieve_message_ex(
connection_t conn, mms_uri_t const *muri, reply_info_t
*reply);
```

**Parameters**

[conn](#) [in] – pointer to the [connection\\_t](#) handle from [mmss connect](#)

[muri](#) [in] – URI to MMS message to retrieve and additional data

[reply](#) [out] – pointer to the reply information structure [reply\\_info\\_t](#) in [synchronous](#) mode. NIL if [Asynchronous](#) mode is used.

**Return value** If [mmss success](#) is returned [reply](#) will contain the pointer to the [reply\\_info\\_t](#) structure.

On error, one of the error codes listed below is returned

**Errors**

[mmss success](#) – Message successfully sent to the MMSC

[mmss invalid arg](#) – One of the arguments passed is invalid

[mmss msg\\_get\\_err](#) – The MMS message was not found in the MMSC

[mmss timeout](#) – A timeout occurred when trying to retrieve the message

[mmss bad content](#) – The retrieved data is not a valid MMS message

[mmss unexpected data](#) – The retrieved message has an unknown type

**See also** [mmss send message](#), [mmss retrieve message](#)

## mmss\_set\_decoding\_version

**Description** [mmss set decoding version](#) sets the MMS header decoding version to use when retrieving MMS messages.

This function should be called before calling the [mms retrieve msg](#) function.

*Note:*

Version 1.4 decoding is not backwards compatible with versions 1.3 and 1.2.

Version 1.3 decoding is not backwards compatible with version 1.2.

**Synopsis C/C++**

```
#include "mmslib.h"
#include "mmsstack.h"

mms_error_t MMSSTACK_API mms_set_decoding_version(
connection_t conn, decoding_version_t version);
```

**Parameters**

`conn` [in] – pointer to the [connection\\_t](#) handle from [mms\\_connect](#)

`version` [in] – specifies a value of a decoding version.

**Return value**

If [mms\\_success](#) is returned the decoding version has been successfully set.

On error, one of the error codes listed below is returned

**Errors**

[mms\\_success](#) – Decoding version successfully set

`mms_bad_decoding_ver - version` argument has an invalid value

**See also**

## mmss\_init\_log

**Description**

[mmss\\_init\\_log](#) is used to initialize a log-file.

**Synopsis C/C++**

```
#include "mmslib.h"
#include "mmsstack.h"

mms_error_t MMSSTACK_API mmss_init_log(const char
*file_name);
```

**Parameters**

`file_name` [in] – Specifies the filename to use for the log-file

**Return value**

If [mms\\_success](#) is returned the log-file has been initialized.

**Errors**

[mms\\_success](#) – Log-file initialized successfully

[mms\\_cant\\_init\\_log](#) – The log-file could not be initialized

## mmss\_get\_connection\_param

|                       |                                                                                                                                                                                                                                                                                                                                                    |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b>    | <p><a href="#">mmss_get_connection_param</a> retrieves connection specific parameter</p> <p>Function must be used after connection that successfully established with <a href="#">mmss_send_message</a>, <a href="#">mmss_retrieve_message</a>, <a href="#">mmss_retrieve_message_ex</a>.</p> <p>Function can not be used in synchronous mode.</p> |
| <b>Synopsis C/C++</b> | <pre>#include "mmslib.h" #include "mmsstack.h"  mms_error_t MMSSTACK_API mmss_get_connection_param(connection_t conn, req_param_t param_id, int *val);</pre>                                                                                                                                                                                       |
| <b>Parameters</b>     | <p><code>conn</code> [in] – handle of established connection</p> <p><code>param_id</code> [in] – parameter identifier</p> <p><code>val</code> [out] – where parameter value will be placed</p>                                                                                                                                                     |
| <b>Return value</b>   | If <a href="#">mmss_success</a> is returned then parameter retrieved successfully                                                                                                                                                                                                                                                                  |
| <b>Errors</b>         | <p><a href="#">mmss_success</a> – parameter retrieved successfully</p> <p><a href="#">mmss_invalid_arg</a> – one of the arguments passed is invalid</p> <p><a href="#">mms_cant_connect</a> – connection not established</p>                                                                                                                       |

## MMS Stack SDK API – Message Functions

### mms\_message\_create

|                       |                                                                                                                                                                                                                                                 |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b>    | Creates new pointed multimedia message. Finally the newly created pointed multimedia message shall be destroyed by <a href="#">mms_message_destroy</a> .                                                                                        |
| <b>Synopsis C/C++</b> | <pre>#include "mmslib.h"  MMS_ERROR MMSLIB_API mms_message_create(MMS_MESSAGE *pmsg, MMS_MESSAGE_TYPE type);</pre>                                                                                                                              |
| <b>Parameters</b>     | <p><code>pmsg</code> [in, out] – Points the location where the newly created pointed multimedia message will be stored. It must not be NULL</p> <p><code>type</code>[in] - Specifies the type of the multimedia message. The allowed values</p> |

have [MMS\\_MESSAGE\\_TYPE](#) type.

**Return value** On success, the newly created pointed multimedia message is stored in the location pointed by the pmsg argument, and [MMS\\_ERR\\_SUCCESS](#) is returned.

On error, one of the listed below error codes is returned.

**Errors** [MMS\\_ERR\\_INVALID\\_ARG](#) – One of the passed arguments has wrong value  
[MMS\\_ERR\\_NOMEMORY](#) – Not enough system resources to create new multimedia message

**See also** [mms\\_message\\_destroy](#)

#### EXAMPLE

```
MMS_MESSAGE msg;
MMS_ERROR err;
err = mms_message_create(&msg, SEND_REQUEST_TYPE);
if (MMS_ERR_SUCCESS != err)
    fprintf(stderr, "Could not create new multimedia
                message.\n");
```

## mms\_message\_destroy

**Description** Destroys multimedia message object, freeing the resources it might hold.

**Synopsis C/C++** `#include "mmslib.h"`

```
MMS_ERROR MMSLIB_API mms_message_destroy(MMS_MESSAGE
msg);
```

**Parameters** msg [in] – Shall be existing multimedia message, i.e. it shall be pre-allocated by [mms\\_message\\_create](#) function.

**Return value** On success returns [MMS\\_ERR\\_SUCCESS](#).  
On error, one of the listed below error codes is returned.

**Errors** [MMS\\_ERR\\_INVALID\\_ARG](#) –Passed argument has wrong value

**See also** [mms\\_message\\_create](#)

#### EXAMPLE

```
MMS_MESSAGE msg;
MMS_ERROR err;
err = mms_message_create(&msg, SEND_REQUEST_TYPE);
```

```

if (MMS_ERR_SUCCESS != err)
    fprintf(stderr, "Could not create mew multimedia
                message.\n");
else
{
    /* Some operators */

    /* ... */

    /* Finally destroying the multimedia message object */
    mms_message_destroy(msg);
}

```

## mms\_set\_header\_str

**Description** Sets the string value of the given MMS header. The MMS headers are defined in [\[MMSENCAPS\]](#)

**Synopsis C/C++** `#include "mmslib.h"`

```

MMS_ERROR MMSLIB_API mms_set_header_str(MMS_MESSAGE msg,
MMS_HEADER header, const char *value);

```

**Parameters**

msg [in] – Specifies the multimedia message object.

header [in] - Identifies the MMS header to set its value. The header argument can be set to one of these values:

**BCC, CC, CONTENT\_LOCATION, CONTENT\_TYPE, FROM, MESSAGE\_ID, RESPONSE\_TEXT, SUBJECT, TO, TRANSACTION\_ID, RETRIEVE\_TEXT, REPLY\_CHARGING\_ID, STORE\_STATUS\_TEXT**

value [in] - specifies the header value to be set. The header value shall be ANSI 0-terminated string. For **FROM** header **NULL** value is allowed. In this case **Insert-address-token** value will be set. For more details refer to [\[MMSENCAPS\]](#)

**Return value** On success returns **MMS\_ERR\_SUCCESS**.  
On error, one of the listed below error codes is returned.

**Errors**

**MMS\_ERR\_INVALID\_ARG** – One of the passed arguments has wrong value

**MMS\_ERR\_NOMEMORY** - Not enough system resources to set header value

**MMS\_ERR\_HEADER\_MISSED** - The required header is not supported by this function. The following situations are possible: header requires integer value or header is not allowed for the message type.

**See also** [mms\\_set\\_header\\_long](#), [mms\\_get\\_header\\_str](#), [mms\\_get\\_header\\_str\\_array](#), [mms\\_get\\_header\\_long](#)

**EXAMPLE**

```
MMS_MESSAGE msg;
MMS_ERROR err;
err = mms_message_create(&msg, SEND_REQUEST_TYPE);
if (MMS_ERR_SUCCESS == err)
    mms_set_header_str(mms, TO, "mms@mail.com");
```

**mms\_set\_header\_encstr**

**Description** Sets the encoded-string value of the given MMS header. The MMS headers are defined in [\[MMSENCAPS\]](#)

**Synopsis C/C++** `#include "mmslib.h"`

```
MMS_ERROR MMSLIB_API mms_set_header_encstr(MMS_MESSAGE
msg, MMS_HEADER header, const ENCODED_STRING *value);
```

**Parameters**

msg [in] – Specifies the multimedia message object.

header [in] - Identifies the MMS header to set its value. The header argument can be set to one of these values:

**[BCC](#), [CC](#), [RESPONSE\\_TEXT](#), [SUBJECT](#), [TO](#), [RETRIEVE\\_TEXT](#), [STORE\\_STATUS\\_TEXT](#)**

value [in] - Specifies the header value to be set. The header value shall point to the [ENCODED STRING](#) structure.

**Return value**

On success returns **[MMS\\_ERR\\_SUCCESS](#)**.

On error, one of the listed below error codes is returned.

**Errors**

**[MMS\\_ERR\\_INVALID\\_ARG](#)** – One of the passed arguments has wrong value

**[MMS\\_ERR\\_NOMEMORY](#)** - Not enough system resources to set header value

**[MMS\\_ERR\\_HEADER\\_MISSED](#)** - The required header is not supported by this function. The following situations are possible: header requires integer value or header is not allowed for the message type.

**See also**

[mms set header long](#), [mms get header str](#),  
[mms get header str array](#), [mms get header long](#),  
[mms get header encstr](#), [mms get header encstr array](#)

**EXAMPLE**

```
MMS_MESSAGE msg;
MMS_ERROR err;
err = mms_message_create(&msg, SEND_REQUEST_TYPE);
if (MMS_ERR_SUCCESS == err)
{
    ENCODED_STRING encstr;
    char value[] = "mms@mail.com";
```



```

    encstr.str = value;
    encstr.strlen = strlen(value) + 1;
    encstr.charset = MMS_DEFAULT_CHARSET;
    mms_set_header_encstr(mms, TO, &encstr);
}

```

## mms\_get\_header\_str

**Description** Returns the string value of the given MMS header. The MMS headers are defined in [\[MMSENCAPS\]](#).

**Synopsis C/C++**

```

#include "mmslib.h"

MMS_ERROR MMSLIB_API mms_get_header_str(MMS_MESSAGE msg,
MMS_HEADER header, const char **value);

```

**Parameters**

msg [in] – Specifies the multimedia message object.

header [in] - Identifies the MMS header to set its value. The header argument can be set to one of these values:

**CONTENT\_LOCATION, CONTENT\_TYPE, FROM, MESSAGE\_ID, RESPONSE\_TEXT, SUBJECT, TRANSACTION\_ID, RETRIEVE\_TEXT, REPLY\_CHARGING\_ID, STORE\_STATUS\_TEXT**

value [out] - points the location where the address of the first character of the header value will be set. The header value is ANSI 0-terminated string.

**Return value** On success, the address of the first character of the header value is placed in the location pointed by the value argument, and the **MMS\_ERR\_SUCCESS** is returned. The header value is ANSI 0-terminated string.

On error, one of the listed below error codes is returned.

**Errors**

**MMS\_ERR\_INVALID\_ARG** – One of the passed arguments has wrong value

**MMS\_ERR\_HEADER\_MISSED** - No required header.

**See also** [mms\\_get\\_header\\_long](#), [mms\\_get\\_header\\_str\\_array](#), [mms\\_set\\_header\\_str](#), [mms\\_set\\_header\\_long](#)

### EXAMPLE

```

MMS_MESSAGE msg;
MMS_ERROR err;
err = mms_message_create(&msg, SEND_REQUEST_TYPE);
if (MMS_ERR_SUCCESS == err)
{
    const char *val;
    mms_get_header_str(mms, TO, &val);
    printf("To: %s\n", val);
}

```

## mms\_get\_header\_encstr

|                       |                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b>    | Returns the encoded-string value of the given MMS header. The MMS headers are defined in <a href="#">[MMSENCAPS]</a> .                                                                                                                                                                                                                                                                     |
| <b>Synopsis C/C++</b> | <pre>#include "mmslib.h"  MMS_ERROR MMSLIB_API mms_get_header_encstr(MMS_MESSAGE msg, MMS_HEADER header, ENCODED_STRING *value);</pre>                                                                                                                                                                                                                                                     |
| <b>Parameters</b>     | <p>msg [in] – Specifies the multimedia message object.</p> <p>header [in] - Identifies the MMS header to set its value. The header argument can be set to one of these values:<br/> <a href="#">RESPONSE_TEXT</a>, <a href="#">SUBJECT</a>, <a href="#">RETRIEVE_TEXT</a>, <a href="#">STORE_STATUS_TEXT</a></p> <p>value [out] - points the <a href="#">ENCODED_STRING</a> structure.</p> |
| <b>Return value</b>   | <p>On success, function fills the <a href="#">ENCODED_STRING</a> structure pointed by the <a href="#">value</a> parameter, and the <a href="#">MMS_ERR_SUCCESS</a> is returned.</p> <p>On error, one of the listed below error codes is returned.</p>                                                                                                                                      |
| <b>Errors</b>         | <p><a href="#">MMS_ERR_INVALID_ARG</a> – One of the passed arguments has wrong value</p> <p><a href="#">MMS_ERR_HEADER_MISSED</a> - No required header.</p>                                                                                                                                                                                                                                |
| <b>See also</b>       | <p><a href="#">mms_get_header_long</a>, <a href="#">mms_get_header_str_array</a>,<br/> <a href="#">mms_set_header_str</a>, <a href="#">mms_set_header_long</a>, <a href="#">mms_set_header_encstr</a>,<br/> <a href="#">mms_get_header_encstr_array</a></p>                                                                                                                                |

### EXAMPLE

```
MMS_MESSAGE msg;
MMS_ERROR err;
err = mms_message_create(&msg, SEND_CONFIRMATION_TYPE);
if (MMS_ERR_SUCCESS == err)
{
    ENCODED_STRING encstr;
    err = mms_get_header_encstr(mms, RESPONSE_TEXT, &encstr);
    if (MMS_ERR_SUCCESS == err)
    {
        char *val;
        val = malloc(encstr.strlen + 1);
        strncpy(val, encstr.str, encstr.strlen);
        val[encstr.strlen] = '\0';
        printf("To: %s\n", val);
        free(val);
    }
}
```

## mms\_get\_header\_str\_array

**Description** Returns the list of the string values of the given MMS header. The MMS headers are defined in [\[MMSENCAPS\]](#). The function [mms\\_get\\_header\\_str\\_array](#) is similar to [mms\\_get\\_header\\_str](#), but it is applied to the MMS headers that can contain several values (multivalued), for example "To: [mms@mail.com](#), [wap@post.org](#)". Returned list is array of ANSI 0-terminated strings. Size of the array (number of the array elements) is returned in the [arr\\_size](#) argument. The client application shall locate enough space for the array and pass address of the first byte of the located space through the [array](#) argument and its size through the [arr\\_size](#) argument (number of the array elements). Also client application can set the argument [array](#) to **NULL** to get required number of the array elements to place all values of the multivalued MMS header.

**Synopsis C/C++** `#include "mmslib.h"`

```
MMS_ERROR MMSLIB_API mms_get_header_str_array(MMS_MESSAGE
msg, MMS_HEADER header, const char *array[], size_t *
arr_size);
```

**Parameters**

`msg [in]` – Specifies the multimedia message object.

`header [in]` - Identifies the MMS header to set its value. The header argument can be set to one of these values:  
**TO, CC, BCC**

`array [out]` - Points the location where array of addresses of the first characters of every header value will be set. The header values are ANSI 0-terminated strings.

`arr_size [in,out]` - Points the location where the number of the values will be placed. Input argument [arr\\_size](#) specifies the size of the located array given in the [array](#) argument. If the count value in the [arr\\_size](#) argument is not enough to place all header values, it returns number of the values placed into the array and the function exits with the error code **MMS\_ERR\_MORE\_DATA**. If the [array](#) argument set to **NULL** the [arr\\_size](#) argument returns required size of the array to place all values of the multivalued MMS header and the function exits with the error code **MMS\_ERR\_MORE\_DATA**.

**Return value** On success, the array pointed by the [array](#) argument is filled with addresses of the first characters of the header values and the number of the placed values is put into location pointed by the [arr\\_size](#) argument, and the **MMS\_ERR\_SUCCESS** is returned. The header values are ANSI 0-terminated strings. On error, one of the listed below error codes is returned.

**Errors**

**MMS\_ERR\_INVALID\_ARG** – One of the passed arguments has wrong value.

**MMS\_ERR\_MORE\_DATA** - Not enough space for placing all values.

**MMS\_ERR\_HEADER\_MISSED** - No required header.

**See also** [mms\\_get\\_header\\_str](#), [mms\\_get\\_header\\_long](#), [mms\\_set\\_header\\_str](#), [mms\\_set\\_header\\_long](#)

#### EXAMPLE

```
MMS_MESSAGE msg;
MMS_ERROR err;
err = mms_message_create(&msg, SEND_REQUEST_TYPE);
if (MMS_ERR_SUCCESS == err)
{
    const char **values;
    size_t count;
    err = mms_get_header_str_array(mms, TO, NULL, &count);
    if (MMS_ERR_MORE_DATA == err)
    {
        values = (const char **) malloc(count *
                                       sizeof(char *));
        err = mms_get_header_str_array(mms, TO, values,
                                       &count);

        if (MMS_SUCCEEDED(err))
        {
            int i;
            for (i = 0; i < count; ++i)
                printf("To: %s\n", values[i]);
        }
    }
}
```

## mms\_get\_header\_encstr\_array

**Description** Returns the list of the encoded-string values of the given MMS header. The MMS headers are defined in [\[MMSENCAPS\]](#). The function [mms\\_get\\_header\\_encstr\\_array](#) is similar to [mms\\_get\\_header\\_encstr](#), but it is applied to the MMS headers that can contain several values (multivalue), for example "To: [mms@mail.com](#), [wap@post.org](#)". Returned list is array of the [ENCODED STRING](#) structures. Size of the array (number of the array elements) is returned in the [arr\\_size](#) argument. The client application shall locate enough space for the array of the [ENCODED STRING](#) structures and pass the address of the array through the [array](#) argument and its size through the [arr\\_size](#) argument (number of the array elements). Also client application can set the argument [array](#) to [NULL](#) to get required number of the array elements to place all values of the multivalue MMS header.

**Synopsis C/C++** `#include "mmslib.h"`

```
MMS_ERROR MMSLIB_API
mms_get_header_encstr_array(MMS_MESSAGE msg, MMS_HEADER
header, ENCODED_STRING *array[], size_t * arr_size);
```

**Parameters** `msg [in]` – Specifies the multimedia message object.  
`header [in]` - Identifies the MMS header to set its value. The header argument

can be set to one of these values:

**TO, CC, BCC**

array [out] - Points the location where array of **ENCODED\_STRING** structures of every header value will be set. The header values are encoded-strings.

arr\_size [in,out] - Points the location where the number of the values will be placed. Input argument **arr\_size** specifies the size of the located array given in the **array** argument. If the count value in the **arr\_size** argument is not enough to place all header values, it returns number of the values placed into the array and the function exits with the error code **MMS\_ERR\_MORE\_DATA**. If the **array** argument set to **NULL** the **arr\_size** argument returns required size of the array to place all values of the multivalued MMS header and the function exits with the error code **MMS\_ERR\_MORE\_DATA**.

#### Return value

On success, the array pointed by the **array** argument is filled with values of **ENCODED\_STRING** structures and the number of the placed values is put into location pointed by the **arr\_size** argument, and the **MMS\_ERR\_SUCCESS** is returned. The header values are encoded-strings. On error, one of the listed below error codes is returned.

#### Errors

**MMS\_ERR\_INVALID\_ARG** – One of the passed arguments has wrong value.

**MMS\_ERR\_MORE\_DATA** - Not enough space for placing all values.

**MMS\_ERR\_HEADER\_MISSED** - No required header.

#### See also

[mms\\_get\\_header\\_str](#), [mms\\_get\\_header\\_long](#), [mms\\_set\\_header\\_str](#), [mms\\_set\\_header\\_long](#), [mms\\_get\\_header\\_encstr](#), [mms\\_set\\_header\\_encstr](#)

#### EXAMPLE

```
MMS_MESSAGE msg;
MMS_ERROR err;
err = mms_message_create(&msg, SEND_REQUEST_TYPE);
if (MMS_ERR_SUCCESS == err)
{
    ENCODED_STRING *values;
    size_t count;
    err = mms_get_header_encstr_array(mms, TO, NULL,
                                     &count);

    if (MMS_ERR_MORE_DATA == err)
    {
        values = (ENCODED_STRING *) malloc(count *
   sizeof(ENCODED_STRING));
        err = mms_get_header_encstr_array(mms, TO, values,
  &count);

        if (MMS_SUCCEEDED(err))
        {
            unsigned int i;
            for (i = 0; i < count; ++i)
                printf("To: %s\n", values[i].str);
        }
    }
}
```

```

        free(values);
    }
}

```

## mms\_set\_header\_long

**Description** Sets the long value of the given MMS header. The MMS headers are defined in [\[MMSENCAPS\]](#). The function [mms\\_set\\_header\\_long](#) is similar to the [mms\\_set\\_header\\_str](#) function, except that it sets the long value instead of string.

**Synopsis C/C++** `#include "mmslib.h"`

```

MMS_ERROR MMSLIB_API mms_set_header_long(MMS_MESSAGE msg,
MMS_HEADER header, long value, unsigned long flags);

```

**Parameters**

`msg [in]` – Specifies the multimedia message object.

`header [in]` - Identifies the MMS header to set its value. The header argument can be set to one of these values:

[ARRIVAL\\_DATE](#), [DELIVERY\\_REPORT](#), [DELIVERY\\_TIME](#), [EXPIRY](#), [MESSAGE\\_CLASS](#), [MMS\\_VERSION](#), [MESSAGE\\_SIZE](#), [PRIORITY](#), [READ\\_REPLY](#), [REPORT\\_ALLOWED](#), [RESPONSE\\_STATUS](#), [SENDER\\_VISIBILITY](#), [STATUS](#), [RETRIEVE\\_STATUS](#), [READ\\_STATUS](#), [REPLY\\_CHARGING](#), [REPLY\\_CHARGING\\_DEADLINE](#), [REPLY\\_CHARGING\\_SIZE](#), [STORE](#), [MM\\_STATE](#), [STORE\\_STATUS](#), [STORED](#), [TOTALS](#), [QUOTAS](#), [MESSAGE\\_COUNT](#), [START](#), [DISTRIBUTION\\_INDICATOR](#), [LIMIT](#)

`value [in]` - Specifies the header value to set. The header value shall correspond to the value of the `header` argument that is listed above.

`flags [in]` - Specifies interpreting value. Currently only the flag [RELATIVE\\_DATE\\_FLAG](#) to point, that the date is relative, is defined for using with headers [EXPIRY](#) and [DELIVERY\\_TIME](#).

**Return value** On success, the header value is set, and [MMS\\_ERR\\_SUCCESS](#) is returned.

**Errors**

[MMS\\_ERR\\_INVALID\\_ARG](#) – One of the passed arguments has wrong value.

[MMS\\_ERR\\_HEADER\\_MISSED](#) - The required header is not supported by this function. The following situations are possible: the header requires integer value or the header is not allowed for the message type.

**See also** [mms\\_set\\_header\\_str](#), [mms\\_get\\_header\\_str](#), [mms\\_get\\_header\\_str\\_array](#), [mms\\_get\\_header\\_long](#)

### EXAMPLE

```

MMS_MESSAGE msg;
MMS_ERROR err;
err = mms_message_create(&msg, SEND_REQUEST_TYPE);
if (MMS_ERR_SUCCESS == err)
    mms_set_header_long(mms, PRIORITY, NORMAL, 0);

```

## mms\_get\_header\_long

**Description** Returns the long value of the given MMS header. The MMS headers are defined in [\[MMSENCAPS\]](#).

**Synopsis C/C++**

```
#include "mmslib.h"

MMS_ERROR MMSLIB_API mms_get_header_long(MMS_MESSAGE msg,
MMS_HEADER header, long *value, unsigned long *flags);
```

**Parameters**

msg [in] – Specifies the multimedia message object.

header [in] - Identifies the MMS header to set its value. The header argument can be set to one of these values:

**ARRIVAL\_DATE, DELIVERY\_REPORT, DELIVERY\_TIME, EXPIRY, MESSAGE\_CLASS, MMS\_VERSION, MESSAGE\_SIZE, PRIORITY, READ\_REPLY, REPORT\_ALLOWED, RESPONSE\_STATUS, SENDER\_VISIBILITY, STATUS, RETRIEVE\_STATUS, READ\_STATUS, REPLY\_CHARGING, REPLY\_CHARGING\_DEADLINE, REPLY\_CHARGING\_SIZE, STORE, MM\_STATE, STORE\_STATUS, STORED, TOTALS, QUOTAS, MESSAGE\_COUNT, START, DISTRIBUTION\_INDICATOR, LIMIT**

value [out] - Argument is the address of the variable where the value of the header will be stored.

flags [out] - Argument specifies interpreting value. Currently only the flag **RELATIVE\_DATE\_FLAG** pointing that the date is relative is defined for using with headers **EXPIRY** and **DELIVERY\_TIME**.

**Return value** On success, the header value is placed into the location pointed by the **value** argument, the **flags** are filled with corresponding constants and the **MMS\_ERR\_SUCCESS** is returned.

On error, one of the listed below error codes is returned.

**Errors** **MMS\_ERR\_INVALID\_ARG** – One of the passed arguments has wrong value.  
**MMS\_ERR\_HEADER\_MISSED** - No required header.

**See also** [mms\\_get\\_header\\_str\\_array](#), [mms\\_set\\_header\\_str](#), [mms\\_get\\_header\\_str](#)

### EXAMPLE

```
MMS_MESSAGE msg;
MMS_ERROR err;
err = mms_message_create(&msg, SEND_REQUEST_TYPE);
if (MMS_ERR_SUCCESS == err)
{
    long val;
    unsigned long flags;
    err = mms_get_header_long(mms, PRIORITY, &val,
```

```

  &flags);
    if (MMS_ERR_SUCCESS == err)
        printf("Priority: %d\n", val);
}

```

## mms\_message\_encode

**Description**      [mms\\_message\\_encode](#) encodes given MMS message according to the [\[MMSENCAPS\]](#).

**Synopsis C/C++**    `#include "mmslib.h"`

```

MMS_ERROR MMSLIB_API mms_message_encode(MMS_MESSAGE msg,
const void **buffer, size_t *size);

```

**Parameters**

msg [in] – Specifies the multimedia message object.

buffer [in,out] - Points the location where the address of the first byte of the encoded message will be placed.

size [in,out] - Points the location where the size of the encoded message will be placed.

**Return value**

On success, the address of the first byte of the encoded message is placed into the location pointed by the `buffer` argument and the encoded message size is placed into the location pointed by the `size` argument and the **MMS\_ERR\_SUCCESS** is returned.

On error, one of the listed below error codes is returned.

**Errors**

**MMS\_ERR\_INVALID\_ARG** – One of the passed arguments has wrong value.

**MMS\_ERR\_HEADER\_MISSED** - Not all required header are set.

**MMS\_ERR\_ENCODE** - Error occurred during message encoding

**See also**            [mms\\_get\\_header\\_str\\_array](#), [mms\\_set\\_header\\_str](#), [mms\\_get\\_header\\_str](#)

### EXAMPLE

```

MMS_MESSAGE msg;
MMS_ERROR err;
err = mms_message_create(&msg, SEND_REQUEST_TYPE);
if (MMS_ERR_SUCCESS == err)
{
    const char *buff;
    size_t sz;
    err = mms_set_header_str(mms, TO, "some@some.com");
    /* some operators here */
    err = mms_message_encode(msg, &buff, &sz);
    if (MMS_ERR_SUCCESS == err)
        printf("Encoded message size is: %d\n", sz);
}

```



## mms\_message\_decode

**Description** [mms\\_message\\_decode](#) decodes the MMS message encoded according to the [\[MMSENCAPS\]](#).

**Synopsis C/C++** `#include "mmslib.h"`

```
MMS_ERROR MMSLIB_API mms_message_decode(MMS_MESSAGE msg,
const void *buffer, size_t size);
```

**Parameters**

`msg [in]` – Specifies the multimedia message object.

`buffer [in]` - Shall point the location where the first byte of the encoded message is placed.

`size [in]` - Shall be set to number of bytes of the encoded message.

**Return value**

On success, the `msg` object is filled with the decoded values and the **MMS\_ERR\_SUCCESS** is returned.

On error, one of the listed below error codes is returned.

**Errors**

**MMS\_ERR\_INVALID\_ARG** – One of the passed arguments has wrong value.

**MMS\_ERR\_DECODE** - Error occurred during message encoding.

**See also** [mms\\_get\\_header\\_str\\_array](#), [mms\\_set\\_header\\_str](#), [mms\\_get\\_header\\_str](#), [mms\\_get\\_header\\_str](#)

### EXAMPLE

```
MMS_MESSAGE msg;
MMS_ERROR err;
const char *buff;
size_t sz;
/* fill the buffer with the encoded message and sz with it
size */
err = mms_message_create(&msg, SEND_CONFIRMATION_TYPE);
if (MMS_ERR_SUCCESS == err)
{
    err = mms_message_decode(msg, buff, sz);
    if (MMS_ERR_SUCCESS == err)
        printf("Message successfully decoded \n");
}
```

## mms\_add\_content

**Description** [mms\\_add\\_content](#) attach the content to the MMS message

**Synopsis C/C++** `#include "mmslib.h"`

```
MMS_ERROR MMSLIB_API mms_add_content(MMS_MESSAGE msg,
```

```
const MMS_CONTENT *content);
```

**Parameters** msg [in] – Specifies the multimedia message object to which content can be attached. I.e. the type of the object shall be either **MMS\_SEND\_REQUEST\_TYPE** or **RETRIEVE\_CONFIRMATION\_TYPE**.

content [in] - shall point the **MMS\_CONTENT** structure.

**Return value** On success, the content added to the msg object and the **MMS\_ERR\_SUCCESS** is returned. On error, one of the listed below error codes is returned.

**Errors** **MMS\_ERR\_INVALID\_ARG** – One of the passed arguments has wrong value.

**See also** [mms\\_get\\_content](#)

#### EXAMPLE

```
MMS_MESSAGE msg;
MMS_ERROR err;
MMS_CONTENT cont;
const char SMIL[] = "<smil>...</smil>";
err = mms_message_create(&msg, MMS_SEND_REQUEST_TYPE);
if (MMS_ERR_SUCCESS == err)
{
    cont.content_type = "application/smil";
    cont.headers = "Content-ID:
        <presentation-part>\r\nContent-
        Location: test.smil\r\n\r\n";
    cont.data = (void *)SMIL;
    cont.data_size = sizeof(SMIL) - 1;
    err = mms_add_content(msg, &cont);
    if (MMS_ERR_SUCCESS == err)
        printf("Content successfully attached \n");
}
```

## mms\_get\_content

**Description** [mms\\_get\\_content](#) gets the content of the MMS message as array of the pointers to the **MMS\_CONTENT** structures.

**Synopsis C/C++** `#include "mmslib.h"`

```
MMS_ERROR MMSLIB_API mms_get_content(MMS_MESSAGE msg,
const MMS_CONTENT *content[], size_t *cont_size);
```

**Parameters** msg [in] – Specifies the multimedia message object to which content can be attached. I.e. the type of the object shall be either **MMS\_SEND\_REQUEST\_TYPE** or **RETRIEVE\_CONFIRMATION\_TYPE**.

content [in,out] - Points the location where the array of addresses of the **MMS\_CONTENT** structures is placed.

cont\_size[in,out] - Points the location where the number of the values will be placed. As input argument **cont\_size** specifies the size of the located array given in the **content** argument. If the count value in the **cont\_size** argument is not enough to place all contents values, it returns number of placed content values into the **content** array and the function exits with the error code **MMS\_ERR\_MORE\_DATA**. If the **content** argument set to **NULL** the **cont\_size** argument returns required size of the array to place all contents and the function exits with the error code **MMS\_ERR\_MORE\_DATA**.

**Return value** On success, the array pointed by the **content** argument is filled with addresses of the **MMS\_CONTENT** structures of the MMS message contents and the number of the placed values is placed in the location pointed by the **cont\_size** argument, and the **MMS\_ERR\_SUCCESS** is returned.

On error, one of the listed below error codes is returned.

**Errors**

- MMS\_ERR\_INVALID\_ARG** – One of the passed arguments has wrong value.
- MMS\_ERR\_MORE\_DATA** - Not enough space for placing all values
- MMS\_ERR\_HEADER\_MISSED** - There is no content

**See also** [mms\\_add\\_content](#)

#### EXAMPLE

```
const MMS_MESSAGE msg;
MMS_ERROR err;
MMS_CONTENT *content[20];
size_t sz;
err = mms_message_create(&msg, RETRIEVE_CONFIRMATION_TYPE);
/* here some operators: decoding message */
if (MMS_ERR_SUCCESS == err)
{
    sz = sizeof(content)/sizeof(content[0]);
    err = mms_get_content(msg, content, &sz);
    if (MMS_ERR_SUCCESS == err)
        printf("Content consist from %d parts \n", sz);
}
```

## mms\_get\_message\_type

**Description** [mms\\_get\\_message\\_type](#) gets the type of the MMS message from the buffer where encoded message is placed.

**Synopsis C/C++** `#include "mmslib.h"`

```
MMS_MESSAGE_TYPE MMSLIB_API mms_get_message_type(const
void *buffer);
```

**Parameters**      buffer [in] – Points the location where the encoded message is placed.

**Return value**      On success, one of the [MMS\\_MESSAGE\\_TYPE](#) is returned. On error, [TYPE\\_UNKNOWN](#) is returned.

**Errors**

**See also**

#### EXAMPLE

```
const char *buff;
size_t sz;
MMS_MESSAGE_TYPE mtype;
MMS_MESSAGE msg;
MMS_ERROR err;
/* fill the buffer with the encoded message and sz with its
   size */

mtype = mms_get_message_type(buff);
if (TYPE_UNKNOWN != mtype)
{
    err = mms_message_create(&msg, mtype);
    ...
}
```

## wsp\_push\_message\_init

**Description**      [wsp\\_push\\_message\\_init](#) initializes the WSP Push message structure.

**Synopsis C/C++**    #include "mmslib.h"

```
void MMSLIB_API wsp_push_message_init(WSP_PUSH_MESSAGE
*msg);
```

**Parameters**      msg [in,out] – Shall point the [WSP\\_PUSH\\_MESSAGE](#) structure.

**Return value**      The function returns nothing.

**Errors**

**See also**

#### EXAMPLE

```
WSP_PUSH_MESSAGE push_msg;
wsp_push_message_init(&push_msg);
```

## wsp\_push\_message\_release

**Description**      [wsp\\_push\\_message\\_release](#) releases the resources allocated by the WSP

Push message.

**Synopsis C/C++** `#include "mmslib.h"`

```
void MMSLIB_API wsp_push_message_release(WSP_PUSH_MESSAGE
*msg);
```

**Parameters** msg [in] – Shall point the **WSP\_PUSH\_MESSAGE** structure.

**Return value** The function returns nothing.

**Errors**

**See also**

**EXAMPLE**

```
WSP_PUSH_MESSAGE push_msg;
wsp_push_message_init(&push_msg);
. . .
wsp_push_message_release(&push_msg);
```

## wsp\_push\_message\_decode

**Description** **wsp\_push\_message\_decode** decode the WSP Push message from the buffer where encoded message is placed.

**Synopsis C/C++** `#include "mmslib.h"`

```
int MMSLIB_API wsp_push_message_decode(const void *pdu,
size_t size, WSP_PUSH_MESSAGE *msg);
```

**Parameters** pdu [in] – Shall point the location where the first byte of the encoded Push message is placed.

size[in] - Shall be set to number of bytes of the encoded message.

msg[in,out] - Shall point the **WSP\_PUSH\_MESSAGE** structure. On return this structure contains message transaction ID, content type and headers.

**Return value** On success - the positive offset in bytes from pdu where message body is placed.

On failure - negative number, one of the following error codes

**Errors** **ERR\_NOT\_PUSH** - not push message

**ERR\_WRONG\_PARAMETER** - one of the given parameters is wrong

**ERR\_FORMAT\_CORRUPT** - message format corrupted

In case when error occurs during content type or headers decoding no error

reported and corresponding members of the `msg` parameter set to zero.

## See also

### EXAMPLE

```
WSP_PUSH_MESSAGE push_msg;
unsigned char BUFFER[1024 * 100];
int offset;
wsp_push_message_init(&push_msg);
offset = wsp_push_message_decode(buff, size, &push_msg);
if (offset < 0)
    printf("Can't decode wsp push message. Error: %d\n",
           offset);
else
{
    . . .
}
wsp_push_message_release(&push_msg);
```

## mms\_decode\_sms\_message

**Description** This function decodes an SMS message, or several SMS messages and extracts the MMS Notification Indication message.

**Synopsis C/C++** `#include "mmslib.h"`

```
MMS_ERROR MMSLIB_API mms_decode_sms_message (const char
*sms_buf, MMS_MESSAGE msg, const char *max_num, const
char *curr_num);
```

**Parameters**

- `sms_buf` [in] – pointer to the SMS message text in hex characters.
- `msg` [in,out] – pointer to the MMS message that will contain the decode message
- `max_num` [out] - specifies the number of SMS parts that build up the SMS
- `curr_num` [out] – specifies the currently decoded SMS message part number

**Return value**

If [mmss\\_success](#) is returned, the `msg` MMS message contains the decoded values.

If [mms\\_more\\_data](#) is returned, another SMS part must be decoded in order to complete the SMS decoding.

If [mms\\_err\\_msg\\_decode](#) is returned the SMS message is not formatted correctly.

The function may also return other error messages that are related to the MMS decoding. Please see the MMS error codes for more details.

**Errors** [mms\\_more\\_data](#) - The function must be called again with the next SMS part

[mms\\_err\\_msg\\_decode](#) – The SMS message is malformed

[mms\\_invalid\\_arg](#) - One of the passed arguments has wrong value

[mms\\_err\\_decode](#) - Error occurred during MMS message decode

## mms\_get\_decoding\_version

**Description** [mms\\_get\\_decoding\\_version](#) gets the current MMS message decoding version.

**Synopsis C/C++**

```
#include "mmslib.h"

unsigned char MMSLIB_API mms_get_decoding_version();
```

**Parameters** The function has no parameters.

**Return value** Returns current MMS message decoding version.

### Errors

**See also** [mms\\_set\\_decoding\\_version](#)

### EXAMPLE

```
unsigned char decoding_version;
decoding_version = mms_get_decoding_version();
```

## mms\_set\_decoding\_version

**Description** [mms\\_set\\_decoding\\_version](#) sets the MMS message decoding version for WAP WSP. For more details refer to [\[WAPWPS\]](#).

**Synopsis C/C++**

```
#include "mmslib.h"

MMS_ERROR MMSLIB_API mms_set_decoding_version(unsigned
char version);
```

**Parameters** version [in] – shall be one of the following values

[DECODING\\_VERSION\\_12](#)

[DECODING\\_VERSION\\_13](#)

[DECODING\\_VERSION\\_14](#)

The default value of the message decoding version is

[DECODING\\_VERSION\\_13](#)

**Return value** On success – function returns [MMS\\_ERR\\_SUCCESS](#).

On error – [MMS\\_ERR\\_INVALID\\_DECODING\\_VERSION](#).

## Errors

**See also** [mms\\_get\\_decoding\\_version](#)

## EXAMPLE

```
unsigned char decoding_version;
MMS_ERROR err;

decoding_version = DECODING_VERSION_13;
err = mms_set_decoding_version(decoding_version);
```

## mms\_add\_attributes

**Description** [mms\\_add\\_attributes](#) attaches the attributes to the MMS message.

**Synopsis C/C++** `#include "mmslib.h"`

```
MMS_ERROR MMSLIB_API mms_add_attributes(MMS_MESSAGE msg,
const MMS_ATTRIBUTES *attributes);
```

**Parameters** msg [in] – Specifies the multimedia message object to which attributes can be attached. I.e. the type of the object shall be either [MBOX\\_VIEW\\_REQUEST\\_TYPE](#) or [MBOX\\_VIEW\\_CONFIRMATION\\_TYPE](#).

attributes[in] - Shall point the [MMS\\_ATTRIBUTES](#) structure.

**Return value** On success, the attributes are added to the msg object and the [MMS\\_ERR\\_SUCCESS](#) is returned. On error, one of the listed below error codes is returned.

**Errors** [MMS\\_ERR\\_INVALID\\_ARG](#) - One of the passed arguments has wrong value.

**See also** [mms\\_get\\_attributes](#)

## EXAMPLE

```
MMS_MESSAGE msg;
MMS_ERROR err;
MMS_ATTRIBUTES attr;
err = mms_message_create(&msg, MBOX_VIEW_REQUEST_TYPE);
if (MMS_ERR_SUCCESS == err)
{
    memset(&attr, 0, sizeof(MMS_ATTRIBUTES));
    attr.subject.str = "test";
    attr.subject.strlen = strlen("test") + 1;
    attr.subject.charset = MMS_DEFAULT_CHARSET;
    err = mms_add_attributes(msg, &attr);
    if (MMS_ERR_SUCCESS == err)
        printf("Attributes successfully attached \n");
}
```



}

## mms\_get\_attributes

**Description** `mms_get_attributes` gets the attributes of the MMS message as an array of the [MMS\\_ATTRIBUTES](#) structures.

**Synopsis C/C++** `#include "mmslib.h"`

```
MMS_ERROR MMSLIB_API mms_get_attributes(MMS_MESSAGE msg,
MMS_ATTRIBUTES *attributes, size_t *attr_size);
```

**Parameters** `msg [in]` – Specifies the multimedia message object to which attributes can be attached. I.e. the type of the object shall be either [MBOX\\_VIEW\\_REQUEST\\_TYPE](#) or [MBOX\\_VIEW\\_CONFIRMATION\\_TYPE](#).

`attributes[in,out]` - Points the location where the array of the [MMS\\_ATTRIBUTES](#) structures is placed.

`attr_size[in,out]` - Points the location where the number of the values will be placed. As input argument `attr_size` specifies the size of the located array given in the `attributes` argument. If the count value in the `attr_size` argument is not enough to place all attributes values, it returns number of placed attributes values into the `attributes` array and the function exits with the error code [MMS\\_ERR\\_MORE\\_DATA](#). If the `attributes` argument set to `NULL` the `attr_size` argument returns required size of the array to place all attributes and the function exits with the error code [MMS\\_ERR\\_MORE\\_DATA](#). For such fields of each [MMS\\_ATTRIBUTES](#) structure as `bcc`, `cc`, `content`, `previously_sent_by`, `previously_sent_date` and `additional_headers` the necessary memory block shall be allocated and the appropriate size fields `bcc_size`, `cc_size`, `content_size` and `additional_headers_size` shall be initialized with the proper values (see example). If some of size fields values is not enough to place all field values, the function returns number of placed field values into the corresponding field (`bcc`, `cc`, `content` or `additional_headers`) and exits with the error code [MMS\\_ERR\\_MORE\\_DATA](#). If any of fields `bcc`, `cc`, `content` or `additional_headers` are set to `NULL` the `bcc_size`, `cc_size`, `content_size` or `additional_headers_size` returns required size of the array to place all `bcc`, `cc`, `content` or `additional_headers` and the function exits with the error code [MMS\\_ERR\\_MORE\\_DATA](#).

**Return value** On success, the array pointed by the `attributes` argument is filled with the [MMS\\_ATTRIBUTES](#) structures of the MMS message attributes and the number of the placed values is set in the location pointed by the `attr_size` argument, and the [MMS\\_ERR\\_SUCCESS](#) is returned. On error, one of the listed below error codes is returned.

**Errors** [MMS\\_ERR\\_INVALID\\_ARG](#) - One of the passed arguments has wrong value.  
[MMS\\_ERR\\_MORE\\_DATA](#) - Not enough space for placing all values  
[MMS\\_ERR\\_HEADER\\_MISSED](#) - There is no attributes header

**See also**      [mms add attributes](#)**EXAMPLE**

```
const MMS_MESSAGE msg;
MMS_ERROR err;
MMS_ATTRIBUTES attributes[20];
ENCODED_STRING bcc[20*10];
size_t bcc_size, attr_size;
unsigned int i;
err = mms_message_create(&msg, MBOX_VIEW_CONFIRMATION_TYPE);
/* here some operators: decoding message */
if (MMS_ERR_SUCCESS == err)
{
    attr_size = sizeof(attributes)/sizeof(attributes[0]);
    memset(&attributes[0], 0, sizeof(attributes));
    bcc_size = sizeof(bcc)/sizeof(bcc[0]);
    bcc_size = bcc_size/attr_size;
    for(i = 0; i < attr_size; ++ i)
    {
        attributes[i].bcc = &bcc[bcc_size * i];
        attributes[i].bcc_size = bcc_size;
    }
    err = mms_get_attributes(msg, attributes, &attr_size);
    if(MMS_ERR_SUCCESS == err || MMS_ERR_MORE_DATA == err)
        printf("Attributes consist from %d parts \n",
                attr_size);
}
```

## 7. GPRS session management

GPRS functions were included into SDK to provide easy way to manage GPRS connection. These functions allow to enumerate devices that can be used to perform GPRS connection as well as establishing and disconnecting GPRS connection. These functions available for Win32 platform only.

### Error codes

|                                     |      |                                              |
|-------------------------------------|------|----------------------------------------------|
| <b>GPRS_ERROR_NO_MORE_ITEMS</b>     | (-1) | No more items in collection                  |
| <b>GPRS_ERROR_NO_MEMORY</b>         | (-2) | Memory allocation error                      |
| <b>GPRS_ERROR_INVALID_PARAMETER</b> | (-3) | Invalid input parameter                      |
| <b>GPRS_ERROR_PIN_REQUIRED</b>      | (-4) | PIN code is required to establish connection |
| <b>GPRS_ERROR_PIN_INCORRECT</b>     | (-5) | PIN code were provided is incorrect          |
| <b>GPRS_ERROR_UNKNOWN</b>           | (-6) | Unknown error                                |

\* Other error codes which are greater then zero is come from underlying subsystem.

### gprs\_device\_enum

**Description** Enumerate GPRS devices available to establish connection

**Synopsis C/C++** `#include "gprsconn.h"`

```
int gprs_device_enum(char * device, size_t * bytes, size_t index);
```

#### Parameters

device [in] – pointer to array of bytes that receives device name

bytes [in,out] – pointer to a bytes count of device member pointed to. When there are not enough memory to complete operation function returns **GPRS\_ERROR\_NO\_MEMORY** and bytes parameter is set to required count of bytes

index [in] – zero-based index to the list of interfaces in the device information set. You should first call this function with the index parameter set to zero to obtain the first interface. Then, repeatedly increment index and retrieve an interface until this function return **GPRS\_ERROR\_NO\_MORE\_ITEMS**

**Return value** If the function succeeds, the return value is zero. If the function fails, the return value is error code

**Errors** **GPRS\_ERROR\_\***, see Error codes

### gprs\_device\_connect

**Description** Establish GPRS connection using specified device

**Synopsis C/C++** `#include "gprsconn.h"`

```
int gprs_device_connect(const char * device,
```

```

const char * apn,
const char * pin,
const char * login,
const char * password,
const char * title,
struct _gprs_connection ** connection);

```

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Parameters</b>   | <p>device [in] – pointer to a device name used to establish connection</p> <p>apn [in] – pointer to Access Point Name string</p> <p>pin [in,optional] – pointer to PIN string</p> <p>login [in,optional] – pointer to login name</p> <p>password [in,optional] – pointer to password</p> <p>title [in,optional] – pointer to connection title string</p> <p>connection [in,optional] – receives established connection handle</p> |
| <b>Return value</b> | If the function succeeds, the return value is zero. If the function fails, the return value is error code                                                                                                                                                                                                                                                                                                                         |
| <b>Errors</b>       | <b>GPRS_ERROR_*</b> , see Error codes                                                                                                                                                                                                                                                                                                                                                                                             |

## gprs\_device\_disconnect

|                       |                                                                                                                                     |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b>    | Close established GPRS connection                                                                                                   |
| <b>Synopsis C/C++</b> | <pre>#include "gprsconn.h"</pre>                                                                                                    |
| <b>Parameters</b>     | <pre>int gprs_device_disconnect(struct _gprs_connection * connection);</pre> <p>Connection [in] – established connection handle</p> |
| <b>Return value</b>   | If the function succeeds, the return value is zero. If the function fails, the return value is error code                           |
| <b>Errors</b>         | <b>GPRS_ERROR_*</b> , see Error codes                                                                                               |

## 8. Synchronous connection

When **callback** member of the structure [conn\\_info\\_t](#) is set to zero the library API functions does not return until them complete the execution. All output parameters will be valid at this moment. The order of API functions calls is as followed

[mmss\\_init](#)

[mmss\\_connect](#)

[mmss\\_set\\_decoding\\_version](#) (optional)

[mmss\\_send\\_message](#) / [mmss\\_retrieve\\_message](#)

[mmss\\_disconnect](#)

[mmss\\_fini](#)

## 9. Asynchronous connection

The **callback** member of the structure [conn\\_info\\_t](#) can point to a callback function of a client application. In that case, functions [mmss\\_connect](#), [mmss\\_send\\_message](#), [mmss\\_retrieve\\_message](#) return immediately after call. If returned code is **mmss\_success** the completion of an operation will be expected in a callback function. The **reason** member of [reply\\_info\\_t](#) parameter of callback describes why a callback function was called and the **result** member of [reply\\_info\\_t](#) contains received information.

- [mmss\\_connect](#)

After execution of [mmss\\_connect](#) a callback function can receive one of the next results

**reason\_connect\_ok** – a connection has been successfully established.

**reason\_connect\_redirect** – a connection shall be reestablished by call of [mmss\\_reconnect](#) function.

**reason\_connect\_error** – a connection attempt failed.

- [mmss\\_send\\_message](#), [mmss\\_retrieve\\_message](#), [mmss\\_retrieve\\_message\\_ex](#)

These functions shall be called only if [mmss\\_connect](#) has been completed with **reason\_connect\_ok**. After their execution a callback function can receive one of the next results

**reason\_reply** – [reply\\_info\\_t](#) parameter of a callback function contains MMSC reply.

**reason\_request\_error** – an error was occurred.