Winwap Technologies Oy

MMS Stack SDK Library

MMS Stack SDK Library version: 1.0
WAP specification version: 2.0
Document dated: 23 March 2005, Rev B4

NOTICE OF CONFIDENTIALITY

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

The recipient agrees to maintain this information in confidence and neither reproduce nor disclose this information to any person outside of the group directly responsible for the evaluation of its contents.

REVISION HISTORY

Date	Author	Description
13 October 2004	O Aizatulin	Initial versions of the document.
21 January 2005	K.Sandell	• Additions, Corrections and document format changes. • Corrected mms_decode_sms_message description & error codes • Restructured all the API description layouts
27 January 2005	K.Sandell	• Updated mms_set_decoding_version description
28 January 2005	K.Sandell	• Changed mms_retreive_msg description • Added mms_init_log API call • Added document version information to page headers
2 February 2005	O Aizatulin	• MMS message encoding/decoding documentation added • Using the MMS Stack section added
23 March 2005	M Shkirja K.Sandell	• Style and formatting corrections • Spelling errors corrected

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

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 ProtocolWAP-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.

5.1. Sending MMS messages

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

1. Initialize the API by calling the **mms_init** function
2. Connect to the HTTP Proxy or WAP Gateway using the **mms_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 **mms_send_message** function
7. Destroy MMS message using **mms_message_destroy** function
8. Disconnect from the HTTP Proxy or WAP Gateway using the **mms_disconnect** function
9. Finalize the API by calling the **mms_fini** function

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

5.2. Retrieving MMS messages

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

1. Initialize the API by calling the **mms_init** function
2. Connect to the HTTP Proxy or WAP Gateway using the **mms_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 **mms_disconnect** function

8. Finalize the API by calling the **mms_fini** function

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

5.3. 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.

6.1. Defined Values

6.1.1. 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);
```

6.1.2. connection_t

connection_t is a handle of an established connection. It is returned from a call to **mmss_connect** function.

```
typedef void *connection_t;
```

6.2. MMS Structures & Definitions

6.2.1. MMS_MESSAGE

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

6.2.2. 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
BCC	Address of the recipient	Encoded-String
CC	Address of the recipient	Encoded-String
CONTENT_LOCATION	Location of the message	String
CONTENT_TYPE	Content type of the message	String
ARRIVAL_DATE	Arrival time of the message at MMSC	Number
DELIVERY_REPORT	Specifies whether the user wants a delivery report from each recipient.	Number
DELIVERY_TIME	Time of desired delivery.	Number
EXPIRY	Length of time the message will be stored in MMSC time to delete the message.	Number
FROM	Address of the message sender.	String
MESSAGE_CLASS	Class of the message.	Number
MESSAGE_ID	This is a unique reference assigned to message. The ID enables a client to match delivery reports with previously sent messages.	String
MESSAGE_TYPE	Identifies the message type.	Number
MMS_VERSION	The MMS version number.	Number
MESSAGE_SIZE	Full size of message in octets.	Number
PRIORITY	Priority of the message for the recipient.	Number
READ_REPLY	Specifies whether the user wants a read report from each recipient as a new message.	Number
REPORT_ALLOWED	Sending of delivery report allowed to the user or not.	Number
RESPONSE_STATUS	MMS specific status.	Number
RESPONSE_TEXT	Description that qualifies the response status value.	Encoded-String

SENDER_VISIBILITY	Show or not address of the sender to the recipient	Number
STATUS	The status of the message.	Number
SUBJECT	Subject of the message.	Encoded-String
TO	Address of the recipient	Encoded-String
TRANSACTION_ID	A unique identifier for the message.	String
RETRIEVE_STATUS	The status preceding retrieval operation.	Number
RETRIEVE_TEXT	Description which qualifies the retrieve status value.	Encoded-String
READ_STATUS	The status of the message.	Number
REPLY_CHARGING	Indication of the originator will to pay for the reply of the recipient.	Number
REPLY_CHARGING_DEAD_LINE	Specifies the latest time of the recipient(s) to submit the Reply-MM	Number
REPLY_CHARGING_ID	The same value as the Message-ID of the Original-MM that is replied to.	String
REPLY_CHARGING_SIZE	Specifies the maximum size for the Reply-MM.	Number
STORE	Specifies whether the originator MMS Client wants the submitted MM to be saved in the user's MMBox.	Number
MM_STATE	Specifies the value to set in the MM State field of the stored MM.	Number
STORE_STATUS	Indicates whether the submitted MM was successfully stored into the MMBox.	Number
STORE_STATUS_TEXT	Description that qualifies the Store_Status field value.	Encoded-String
STORED	Indicates whether MM was stored to the user's MMBox.	Number
TOTALS	Gives the total number of messages or bytes in the user's MMBox.	Number
QUOTAS	Gives the quotas defined for the user's MMBox in messages and/or bytes.	Number
MESSAGE_COUNT	Number of messages listed in content of the PDU.	Number
START	A number, indicating the index of the first MM of those selected to have information returned in the response.	Number

DISTRIBUTION_INDICATOR	This field MAY be present for an MM that originated from a Value Added Service Provider and the original included this indicator.	Number
LIMIT	A number indicating the maximum number of selected MMs whose information SHALL be returned in the response.	Number
HEADER_UNKNOWN	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.	

6.2.3. 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

SEND_REQUEST_TYPE	This kind of messages sent by the client to the MMSC,
SEND_CONFIRMATION_TYPE	When the MMSC has received the Send request, it sends a response message back to the client indicating the status of the operation.
NOTIFICATION_INDICATION_TYPE	Notification indications inform the client about the contents a received message.
NOTIFYRESP_INDICATION_TYPE	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.
RETRIEVE_CONFIRMATION_TYPE	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.
ACKNOWLEDGE_INDICATION_TYPE	Acknowledge message confirms the delivery of the message from the client to the MMSC.
DELIVERY_INDICATION_TYPE	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.
READREC_INDICATION_TYPE	The message is used for the handling Read Reports on the MM recipient side.
READORIG_INDICATION_TYPE	The message is used for the handling Read Reports on the MM originating side.
FORWARD_REQUEST_TYPE	MMS Client sends this message to the MMS Proxy-Relay to request forwarding of an MMS message.
FORWARD_CONFIRMATION_TYPE	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.

MBOX_STORE_REQUEST_TYPE	When MMS Client decides to save a MM that has not been retrieved as yet, it sends this message to the MMS Proxy-Relay.
MBOX_STORE_CONFIRMATION_TYPE	When MMS Proxy-Relay has received the Store request it sends a confirmation to the MMS Client via this message.
MBOX_VIEW_REQUEST_TYPE	MMS Client may request a listing of MM that are stored in the user's MMBox or obtain information concerning a set of MM known to be stored in the MMBox.
MBOX_VIEW_CONFIRMATION_TYPE	When MMS Proxy-Relay has received the View request it SHOULD reply with this message.
MBOX_UPLOAD_REQUEST_TYPE	When MMS Client decides to save a MM that is on the MMS Terminal to the user's MMBox, it transfers a MM using this message.
MBOX_UPLOAD_CONFIRMATION_TYPE	When MMS Proxy-Relay has received the Upload request it sends a confirmation to the MMS Client via this message.
MBOX_DELETE_REQUEST_TYPE	To delete one or more MM from the user's MMBox the MMS Client SHALL send this message indicating the location reference of all messages to be deleted.
MBOX_DELETE_CONFIRMATION_TYPE	MMS Proxy-Relay SHALL respond to the deletion request with this message indicating if the deletion was successful.
MBOX_DESCRIPTION_TYPE	This PDU SHALL be used by the View confirmation and the Upload request data payload to describe the MM.
UNKNOWN_TYPE	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 **mms_message_create** function. Also can be returned by the **mms_get_message_type** function.

6.2.4. 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

STATUS_OK	The corresponding request and some or all of its contents were accepted without errors.
ERROR_UNSPECIFIED	An unspecified error occurred during the processing or reception of the corresponding request
ERROR_SERVICE_DENIED	The client does not have permission or funds to perform the requested operation
ERROR_MESSAGE_FORMAT_CORRUPT	An inconsistency with the message format was detected when the corresponding request was parsed

ERROR_SENDING_ADDRESS_UNRESOLVED	There were no MMS address (From:, To:, Cc:, Bcc:) in its proper format or none of the addresses belong to the MMSC.
ERROR_MESSAGE_NOT_FOUND	This status code is obsolete
ERROR_NETWORK_PROBLEM	The MMSC was not able to accept the corresponding request due to capacity overload
ERROR_CONTENT_NOT_ACCEPTED	The MM content was not accepted due to size, media type, copyrights or some other reason.
ERROR_UNSUPPORTED_MESSAGE	The MMSC does not support the corresponding request abstract message.
ERROR_TRANSIENT_FAILURE	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.
ERROR_TRANSIENT_SENDING_ADDRESS_UNRESOLVED	This X-Mms-Response-Status value SHOULD not be used in the M-Send.conf PDU.
ERROR_TRANSIENT_MESSAGE_NOT_FOUND	This X-Mms-Response-Status value SHOULD not be used in the M-Send.conf PDU.
ERROR_TRANSIENT_NETWORK_PROBLEM	The MMS Proxy-Relay was not able to handle the corresponding MSend.req due to unspecified error on the transport layer or capacity overload.
ERROR_TRANSIENT_PARTIAL_SUCCESS	This X-Mms-Response-Status value SHOULD not be used in the M-Send.conf PDU.
ERROR_PERMANENT_FAILURE	An unspecified permanent error occurred during the processing or reception of the corresponding MSend.req.
ERROR_PERMANENT_SERVICE_DENIED	The corresponding M-Send.req was rejected due to failure of authentication or authorization due to different reasons of the originating MMS Client.
ERROR_PERMANENT_MESSAGE_FORMAT_CORRUPT	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.
ERROR_PERMANENT_SENDING_ADDRESS_UNRESOLVED	The MMS Proxy-Relay was not able to resolve the insert-address-token into a valid sending address.
ERROR_PERMANENT_MESSAGE_NOT_FOUND	This X-Mms-Response-Status value SHOULD not be used in the M-Send.conf PDU.
ERROR_PERMANENT_CONTENT_NOT_ACCEPTED	The MM content in the M-Send.req was not accepted due to size, media type, copyrights or some other reason.

ERROR_PERMANENT_REPLY_CHARGING_LIMITATIONS_NOT_MET	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.
ERROR_PERMANENT_REPLY_CHARGING_REQUEST_NOT_ACCEPTED	The M-Send.req contained an XMms-Reply-Charging header field with the value 'Accepted' or 'Accepted text only'.
ERROR_PERMANENT_REPLY_CHARGING_FORWARDING_DENIED	This X-Mms-Response-Status value SHOULD not be used in the M-Send.conf PDU.
ERROR_PERMANENT_REPLY_CHARGING_NOT_SUPPORTED	The MMS Proxy-Relay does not support reply charging. The corresponding M-Send.req contained reply-charging parameters and was rejected.
ERROR_PERMANENT_ADDRESS_HIDING_NOT_SUPPORTED	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.
RESPONSE_STATUS_UNKNONW	The response 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.

6.2.5. MMS_YES_NO

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

CONSTANTS

YES

NO

6.2.6. MMS_MESSAGE_CLASS

This header describes the class of the message. The following table describes the possible values of the Message-Class header.

CONSTANTS

PERSONAL	The message is personal
ADVERTISEMENT	The message is advertisement
INFORMATIONAL	The message is informational
CLASS_UNKNONW	The message class 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.

6.2.7. MMS_PRIORITY

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

CONSTANTS

LOW	The message has low priority
NORMAL	The message has normal priority
HIGH	The message has high priority
PRIORITY_UNKNONW	The message priority 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.

6.2.8. 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

HIDE	Do not show the sender's identity
SHOW	Show the sender's identity
VISIBILITY_UNKNOWN	The sender visibility 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.

6.2.9. MMS_STATUS

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

CONSTANTS

EXPIRED	Message has been expired
RETRIEVED	Message has been retrieved
REJECTED	Message has been rejected
DEFERRED	Message has been deferred
UNRECOGNISED	Status not available
INDETERMINATE	It is not possible to determine if the MM in question reached its destination.

FORWARDED	The MM in question reached the recipient MS Proxy-Relay but the recipient MMS Client forwarded it without retrieving it first.
UNREACHABLE	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.
STATUS_UNKNONW	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.

6.2.10. 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

STATUS_OK_RET	The retrieve request was accepted and processed without any errors.
ERROR_TRANSIENT_FAILURE_RET	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.
ERROR_TRANSIENT_MESSAGE_NOT_FOUND_RET	The MMS Proxy-Relay has temporary lost contact with where the MM's are stored, e.g. the MMS Server.
ERROR_TRANSIENT_NETWORK_PROBLEM_RET	The MMS Proxy-Relay was not able to handle the corresponding retrieve request due to capacity overload.
ERROR_PERMANENT_FAILURE_RET	An unspecified permanent error occurred during the processing of the retrieval attempt of the MM in question.
ERROR_PERMANENT_SERVICE_DENIED_RET	The corresponding retrieval attempt was rejected due to failure of authentication or authorization of the originating MMS Client.
ERROR_PERMANENT_MESSAGE_NOT_FOUND_RET	The content location URL in the retrieval attempt does not point to an MM.
ERROR_PERMANENT_CONTENT_UNSUPPORTED_RET	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.

RETRIEVE_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.

6.2.11. 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

READ	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.

6.2.12. 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.

6.2.13. 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.

6.2.14. 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

STATUS_OK_STR	The store request was accepted and processed without any errors.
ERROR_TRANSIENT_FAILURE_STR	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.
ERROR_TRANSIENT_NETWORK_PROBLEM_STR	The MMS Proxy-Relay was not able to handle the corresponding store request due to capacity overload.
ERROR_PERMANENT_FAILURE_STR	An unspecified permanent error occurred during the processing or reception of the corresponding request.
ERROR_PERMANENT_SERVICE_DENIED_STR	The corresponding storage request was rejected due to failure of authentication or authorization due to store the MM into the MMBox.
ERROR_PERMANENT_MESSAGE_FORMAT_CORRUPT_STR	This X-Mms-Store-Status value SHOULD not be used in the M-Send.conf or M-orward.conf PDU.
ERROR_PERMANENT_MESSAGE_NOT_FOUND_STR	This X-Mms-Store-Status value SHOULD not be used in the M-Send.conf or M-orward.conf PDU.

ERROR_PERMANENT_MMBOX_FULL_STR	The corresponding storage request failed since the user's MMBox quota was exhausted.
STORE_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.

6.2.15. ENCODED_STRING

ENCODED_STRING specifies an encoded string.

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

Field	Description
str	String data
strlen	Length of the string data
charset	Char-set of the string. Char-set values registered by IANA as MIBEnum value.

6.2.16. 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
content_type	The content type of the content.
headers	The headers associated with content.
data	The pointer to memory buffer where content is placed.
data_size	The size of the memory buffer pointed by data.

6.2.17. 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
content_type	The content type of the WSP Push message.
headers	The headers associated with content of the WSP Push message.
transaction_id	The transaction ID of the WSP Push message.

6.2.18. 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_                    additional_headers_size;
} MMS_ATTRIBUTES;
```

Field	Description

bcc	Bcc field value.
bcc_size	The size of Bcc field value.
cc_size	Cc field value.
cc_size	The size of Cc field value.
content	Content field value.
content_size	The size of Content field value.
content_type	Content-Type field value.
date	Date field value.
delivery_report	X-Mms-Delivery-Report field value.
delivery_time	X-Mms-Delivery-Time field value.
expiry	X-Mms-Expiry field value.
from	From field value.
message_class	X-Mms-Message-Class field value.
message_id	X-Mms-Message-Id field value.
message_size	X-Mms-Message-Size field value.
priority	X-Mms-Priority field value.
read_report	X-Mms-Read-Report field value.
subject	Subject field value.
to	To field value.
to_size	The size of To field value.
reply_charging	X-Mms-Reply-Charging field value.
reply_charging_id	X-Mms-Reply-Charging-Id field value.
reply_charging_deadline	X-Mms-Reply-Charging-Deadline field value.
reply_charging_size	X-Mms-Reply-Charging-Size field value.
previously_sent_by	X-Mms-Previously-Sent-By field value.
previously_sent_date	X-Mms-Previously-Sent-Date field value.
additional_headers	Additional headers field value.
additional_headers_size	The size of Additional headers field value.

6.2.19. MMS_PREVIOUSLY_SENT_BY

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

```
} MMS_PREVIOUSLY_SENT_BY;
```

6.2.20. MMS_PREVIOUSLY_SENT_DATE

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

6.3. MMSS Structures & Definitions

6.3.1. 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
mmss_success	Function has performed its function without errors.
mmss_invalid_arg	One of the passed arguments is invalid.
mmss_cant_open_wps	Can not create WAP Stack.
mmss_cant_bind_wps	Can not bind WAP Stack to a custom system network address.
mmss_cant_connect	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.
mmss_timeout	Function exits by timeout.
mmss_msg_encode_err	Function can not encode MMS message.
mmss_msg_post_err	Failed to send a POST request to WAP Gateway or HTTP Proxy.
mmss_msg_get_err	Failed to send a GET request to WAP Gateway or HTTP Proxy.
mmss_bad_content	WAP Gateway or HTTP Proxy reply is not an MMS message.
mmss_bad_decoding_ver	Function can not set the MMS message decoding version for WAP WSP.
mmss_msg_decode_err	Function can not decode MMS message.
mmss_unexpected_msg	WAP Gateway or HTTP Proxy reply is an MMS message of unknown type.
mmss_cant_init_log	Failed to initialize a log file.

6.3.2. 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
mode_wap_cl	Connection less mode
mode_wap_co	Connection oriented mode
mode_wap_scl	Secure connection less mode
mode_wap_sco	Secure connection oriented mode
mode_http	HTTP mode
mode_https	Secure HTTP mode, using SSL

6.3.3. 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
reason_connect_ok	Connection is successful
reason_connect_redirect	Redirection request received
reason_connect_error	Connection error
reason_reply	MMS Proxy/Relay replay received
reason_request_error	Send request error

6.3.4. address_info_t

address_info_t consists of server address and port.

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

Field	Description
addr	WAP Gateway or HTTP Proxy address.
port	WAP Gateway or HTTP Proxy port.

6.3.5. 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.

6.3.6. 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

6.3.7. 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.

6.3.8. 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 **reason_t** for values
type	The type of the result. Please see **result_type_t** for values
result	This value depends on the type field value. reply - MMS Proxy reply data, see **mms_reply_t** for values msg - MMS Proxy/Relay reply as mms-message redirection - Redirection info. Please see **address_info_t** for values error - Error description. Please see **error_info_t** for values

6.3.9. 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 **conn_mode_t** for values
addr	WAP Gateway or HTTP Proxy IP/Hostname

port	HTTP Proxy port if HTTP mode is used. Otherwise set to 0
user_name	The username to use when connecting
password	The password to use when connecting
headers	WSP session headers in WSP mode
callback	Application defined callback. If set, **Asynchronous** mode is used.

6.3.10. 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

6.3.11. 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

6.4. MMS Stack SDK API Functions

6.4.1. mmss_init

Description **mmss_init** initializes the MMS Stack library.

Synopsis C/C++

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

mmss_error_t MMSSTACK_API mmss_init();
```

Parameters There are no parameters for this function.

Return value On success, **mmss_init** returns **mmss_success**

Errors **mmss_success**

See also **mmss_fini**

6.4.2. 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_fini** returns **mmss_success**

Errors **mmss_success**

See also **mmss_init**

6.4.3. 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"

mmss_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**

6.4.4. 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"

mmss_error_t MMSSTACK_API mmss_disconnect( connection_t
conn);
```

Parameters

conn [in] – pointer to the **connection_t** handle from **mmss_connect**

Return value If **mmss_success** is returned, conn defines a connection handle

Errors **mmss_success** – Disconnected successfully from the MMSC

See also **mmss_connect**

6.4.5. 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"

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

Parameters conn [in] – pointer to the **connection_t** handle from **mmss_connect**

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

gp [in] – The gateway port number

Return value If **mmss_success** is returned the reconnection was successful

Errors **mmss_success** – Reconnected successfully to the MMSC

mmss_cant_connect - Can not establish connection

See also **mmss_connect**, **mmss_disconnect**

6.4.6. 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"

mmss_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 **mmss_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 **mmss_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_retreive_message**

6.4.7. mmss_retrieve_message

Description **mmss_retreive_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"

mmss_error_t MMSSTACK_API mms_retrieve_msg( 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**

6.4.8. 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_retreive_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”

mmss_error_t MMSSTACK_API mmss_set_decoding_version(
connection_t conn, decoding_version_t version);
```

Parameters conn [in] – pointer to the **connection_t** handle from **mmss_connect**

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

Return value If **mmss_success** is returned the decoding version has been successfully set.
On error, one of the error codes listed below is returned

Errors **mmss_success** – Decoding version successfully set
mmss_bad_decoding_ver - version argument has an invalid value

See also

6.4.9. mmss_init_log

Description mmss_init_log is used to initialize a log-file.

Synopsis C/C++

```
#include “mmslib.h”
#include “mmsstack.h”

mmss_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 **mmss_success** is returned the log-file has been initialized.

Errors **mmss_success** – Log-file initialized successfully
mms_cant_init_log – The log-file could not be initialized

See also

6.5. MMS Stack SDK API – Message Functions

6.5.1. mms_message_create

Description Creates new pointed multimedia message. Finally the newly created pointed multimedia message shall be destroyed by **mms_message_destroy**.

Synopsis C/C++

```
#include “mmslib.h”

MMS_ERROR MMSLIB_API mms_message_create(MMS_MESSAGE *pmsg,
MMS_MESSAGE_TYPE type);
```

Parameters pmsg [in, out] – Points the location where the newly created pointed multimedia message will be stored. It must not be NULL

type[in] - Specifies the type of the multimedia message. The allowed values 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");
```

6.5.2. 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);
}
```

6.5.3. 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");
```

6.5.4. 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);
}
```

6.5.5. 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);
}
```

6.5.6. mms_get_header_encstr

Description Returns 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_get_header_encstr(MMS_MESSAGE msg,
MMS_HEADER header, 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:

RESPONSE_TEXT, SUBJECT, RETRIEVE_TEXT, STORE_STATUS_TEXT

value [out] - points the **ENCODED_STRING** structure.

Return value On success, function fills the **ENCODED_STRING** structure pointed by the value parameter, 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_long**, **mms_get_header_str_array**, **mms_set_header_str**, **mms_set_header_long**, **mms_set_header_encstr**, **mms_get_header_encstr_array**

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);
    }
}
```

6.5.7. 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 ontain several values (multivalue), 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 multivalue 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 multivalue 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 cout;
    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]);
            }
        }
    }
```

6.5.8. 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 multivalue 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);
    }
}
```

6.5.9. 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);
```

6.5.10. 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);
    }
```

6.5.11. 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);
}
```

6.5.12. 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");
}
```

6.5.13. 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>\n\rContent-
          Location: test.smil\n\r\n\r";
     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");
```

```
}
```

6.5.14. 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);
}
```

6.5.15. 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 it
                                                    size */
mtype = mms_get_message_type(buff);
if (TYPE_UNKNOWN != mtype)
{
     err = mms_message_create(&msg, mtype);
     …
}
```

6.5.16. 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);
```

6.5.17. 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);
```

6.5.18. 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);
```

6.5.19. 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

6.5.20. mms_get_decoding_version

Description **mms_get_decoding_version** gets the current MMS message decoding version.

Synopsis

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

C/C++
```
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();
```

6.5.21. 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);
```

6.5.22. 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”);
}
```

6.5.23. 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. 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

8. 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.

8.1. 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.

8.2. mmss_send_message, mmss_retrieve_message

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.