BlackBerry Software Development Kit

Version 2.5

Messaging API Reference Guide

BlackBerry Software Development Kit Version 2.5
Messaging API Reference Guide
Last revised: 18 July 2002

Part number: PDF-04636-001

At the time of publication, this documentation complies with the RIM Wireless Handheld version 2.5.

Research In Motion Limited
295 Phillip Street
Waterloo, ON N2L 3W8
Canada

Published in Canada

Contents

About this guide

The Messaging API includes two components:

- standard functions, declared in the `msg_api.h` header file, which provide basic capabilities for sending and receiving email messages and handling attachments
- enhanced functions, declared in the `RimNetworkMessage.h` header file, which provide access to, and control of, underlying message data structures

Related documents

The *BlackBerry SDK Developer Guide* explains how to use the BlackBerry Software Development Kit, with tutorials and sample code to demonstrate how to write handheld applications.

For additional information, visit the BlackBerry Developer Zone at `http://www.blackberry.net/developers`.

Chapter 1
Messaging API

This section describes the functions in the standard Messaging API. The Messaging API, defined in the `msg_api.h` header file, provides standard functions for sending and receiving email messages and handling attachments.

An enhanced Messaging API provides access to underlying message data structures. Refer to "Enhanced Message API" on page 29 for more information.

Messaging API functions

The following is a list of Messaging functions, in alphabetical order. These functions are declared in the header file `msg_api.h.`

abort_message

Aborts the current message.

```
void abort_message()
```

add_attachment_data

Adds an attachment to the message.

```
bool add_attachment_data(
    DbFieldHandle field_handle,
    char const * data,
    int data_length = DB_TEXT)
```

Parameters

`field_handle`	The handle to the attachment to modify.
`data`	A pointer to the data; multiple calls are concatenated.
`data_length`	The length of the data to add to the attachment. If the value is `DB_TEXT`, the data is assumed to be a NULL-terminated string.

Returns True if successful; false if it fails to attach the data.

Description This function appends to any existing data in an attachment. You can use a series of calls to `add_attachment_data` to build an attachment incrementally.

add_field_text

Adds text to the associated field for the message.

```
DbFieldHandle add_field_text(
    DbFieldTag field_tag,
    char const * text,
    int length = DB_TEXT)
```

Parameters

`field_tag`	The message field to which text is to be added. It can be one of `SUBJECT_FIELD_TAG`, `MESSAGE_FIELD_TAG`, `MESSAGE_ID_FIELD_TAG`, `ORIGINAL_MESSAGE_ID_TAG`, `TRANSPORT_ID_FIELD_TAG`, or `GLOBAL_DELIVERY_DATA_TAG`.
`text`	A pointer to the text for the field.
`length`	The length of the text to be added, including the terminating null character.

Returns The handle to the message field to which to append text.

Description You can call `add_field_text` as many times as you want. Text that is pointed to in subsequent calls is appended to the end of the current message. It is possible to build a message piece by piece instead of building it in one call.

There is no call for editing the field text directly. Use the handle that is returned by `add_field_text` to delete the field (with `delete_field`) and recreate it with the correct text.

See also `create_message`

add_recipient

Adds recipients to the associated field for the message.

```
DbFieldHandle add_recipient(
    DbFieldTag field_tag,
    AddressFieldType address_type,
    char const * address,
    char const * name = "")
```

Parameters

`field_tag`	An address field; one of `TO_FIELD_TAG`, `CC_FIELD_TAG`, or `BCC_FIELD_TAG`.
`address_type`	The type of communication desired. The type of communication constrains the type of address that you can use for the message. It is one of `EMAIL`, `FAX`, `PHONE`, `ONE_WAY_PAGE`, and `TWO_WAY_PAGE`. Supported types depend on the network provider. The `EMAIL` type is always supported.
`address`	The email address.
`name`	The full display name that is associated with the email address.

Returns The handle to the email address field.

create_attachment

Creates an attachment for the message.

```
DbFieldHandle create_attachment(
    char const * content_string,
    char const * content_string_concatenation = "",
    char const * data = NULL,
    int data_length = DB_TEXT,
    DbFieldHandle field_handle = DB_INVALID_FIELD_HANDLE)
```

Parameters

`content_string`	Text string that uniquely identifies the attachment type. The actual value that is used depends on the application that interprets the attachment.
`content_string_concatenation`	A valid subtype of the attachment type specified in `content_string`. This parameter is concatenated with `content_string`. For example, if your application iterates through a set of image files that are generated on the handheld, `content_string` might have the value "image/" and `content_string_concatenation` might differ depending on the image type, being "jpeg" in one call and "gif" in another.
`data`	The initial data that makes up the body of the attachment.
`data_length`	The length of the data that is passed to the function. If the value is `DB_TEXT`, then the data is assumed to be a NULL-terminated text string.
`field_handle`	Updates the attachment that is specified by `field_handle`. If the value is `DB_INVALID_FIELD_HANDLE`, a new attachment is created.

Returns A handle to the attachment.

Description The values for `content_string` and `content_string_concatenation` depend on the application that will receive the attachment. If you send an email message to an arbitrary email application, consider using the standard MIME types that are defined in RFC 2046, or use `create_attachment_extended`.

create_attachment_extended

Creates an attachment for the message.

```
DbFieldHandle create_attachment_extended(
    char const * content_type_subtype,
    char const * data = NULL,
    int data_length = DB_TEXT,
    DbFieldHandle field_handle = DB_INVALID_FIELD_HANDLE)
```

Parameters

`content_type_subtype`	Text string that uniquely identifies the attachment type, one of the types defined for MIME in RFC 2046.
`data`	The data that comprises the body of the attachment.
`data_length`	The length of the data that is passed to the function. If the value is `DB_TEXT`, the data is assumed to be a NULL-terminated text string.
`field_handle`	Updates the attachment specified by `field_handle`. If the value is `DB_INVALID_FIELD_HANDLE`, a new attachment is created.

Returns A handle to the attachment.

Description This extended API recognizes the content types that are defined by Internet RFC 2046.

create_message

Creates a message and associates it with the current task.

```
bool create_message()
```

Returns True if successful in associating a new message with the current task.

Description Calling `create_message` a second time will delete the first message.

create_message_extended

Creates a message and associates it with the specified task.

```
bool create_message_extended(long associated_account)
```

Parameters

`associated_account`	Account number for the task; you can retrieve this account handle using a function such as `get_account_handle_from_message` or `query_account_address.`

Returns True if successful; false if the message could not be created or if there is no address type associated with this account.

Description Calling `create_message_extended` a second time with the same `associated_account` will delete the first message.

delete_field

Deletes a field that is associated with the current message that is being created.

```
void delete_field(DbFieldHandle field_handle)
```

Parameters	`field_handle`	The handle to the field (of any type) to be deleted.

Returns True if the field is deleted successfully.

delete_stale_messages

Deletes read or transmitted messages to free space.

```
int delete_stale_messages(int desired_block_size = 5000)
```

Parameters	`desired_block_size`	Specifies how many bytes of free space you want to have in the file system. **Note:** This is not the number of bytes that you want to free.

Returns The number of free bytes in the system.

Description The value 5000 is the largest contiguous amount of space that the system can provide.

Remember that the parameter to this function is the amount of total free space that you want in the file system, not how many bytes to free. The following code sample shows how to specify how many bytes an application wants freed:

```
int i;
// attempt to free 32K
i = delete_stale_messages( 32768 );
if( i < 32768 ) {
    // Error! Could not free enough space for the data file
    return;
}
// continue with file creation
```

Incidentally, deleting messages to free 32 KB could cause problems on systems with large address books; all messages might be deleted.

get_account_handle_from_message

Retrieves a handle that identifies that service that sent the active message.

```
bool get_account_handle_from_message(long & account_handle)
```

Parameters

`account_handle`	Address of a `long` for storing the account handle.

Returns True if successful; false if an error occurs.

Description The account handle is a service record identifier that uniquely identifies a communication method (for example, email, PIN, or fax). This is the same as a "service record ID" in the `RimNetworkMessage` class. If no service has been set (for example, the message is being composed but no recipients have been selected), the `account_handle` is filled with the identifier for email, the default account.

If the call returns false, no message is loaded.

get_address_data

Retrieves address information for the specified field of the active message.

```
bool get_address_data(
    DbFieldTag field_tag,
    AddressFieldType & address_field_type,
    char const * & address_ptr,
    char const * & name_ptr,
    DbFieldHandle field_handle = DB_INVALID_FIELD_HANDLE)
```

Parameters

`field_tag`	An address field; one of `TO_FIELD_TAG`, `CC_FIELD_TAG`, `BCC_FIELD_TAG`, or `FROM_FIELD_TAG`.
`address_field_type`	A return parameter that returns the type of address data: `EMAIL`, `FAX`, `PHONE`, `ONE_WAY_PAGE`, or `TWO_WAY_PAGE`.
`address_ptr`	A return parameter that returns the address.
`name_ptr`	A return parameter that returns the name that is associated with the address.
`field_handle`	Field from which to extract data; if this is `DB_INVALID_FIELD_HANDLE`, data is extracted from the first field of the specified `field_tag`.

Returns True if the request was successful; false otherwise.

Description If no `field_handle` is specified, the `get_address_data` function retrieves information for the first occurrence of an address in the specified `field_tag`.

get_attachment_data

Retrieves the data that is associated with the incoming attachment (deprecated).

```
bool  get_attachment_data(
    DbFieldHandle field_handle,
    char const * & content_type,
    char const * & data_ptr,
    int & data_length,
    char const ** filename_pptr = NULL)
```

Parameters

`field_handle`	The handle for the attachment.
`content_type`	A string identifying the attachment type.
`data_ptr`	A pointer to the attachment data.
`data_length`	The number of bytes returned in `data_ptr`.
`filename_pptr`	Returns a `const` pointer to the file name found in the attachment.

Returns True if the `field_handle` is valid and the data is available.

Description The value for `field_handle` is a parameter that is provided in the `register_view_attachment` callback function.

This function is deprecated; use `get_attachment_data_ex` instead. Do not combine calls to the deprecated and extended versions of this function.

get_attachment_data_ex

Retrieves the data associated with the incoming attachment.

```
bool  get_attachment_data_ex(
    DbFieldHandle field_handle,
    char const * & parms_ptr,
    int & parms_length,
    char const * & data_ptr,
    int & data_length,
    char const ** filename_pptr = NULL)
```

Parameters

`field_handle`	The handle for the attachment.
`parms_ptr`	An out parameter populated with a read-only (flash) pointer to a list of parameters that are associated with the attachment.
`parm_length`	An out parameter populated with the length of the `parms_ptr` parameter.

`data_ptr`	An out parameter populated with a read-only (flash) pointer to the attachment data.
`data_length`	An out parameter populated with the length of the attachment data.
`filename_pptr`	An out parameter populated with a read-only (flash) pointer to the file name that is associated with the attachment.

Returns True if the `field_handle` is valid and the data is available; otherwise false. The function can fail for the following reasons:

- `field_handle` does not refer to a valid attachment field
- message cannot be found
- general system failure

Description The value for `field_handle` is a parameter provided in the `register_view_attachment_ex` callback function.

This function is a replacement for and `get_attachment_data` (now deprecated). Do not combine calls to the deprecated and new functions.

See also `register_view_attachment_ex` (refer to page 25)

get_current_message_handle

Retrieves the message handle that corresponds to the current task.

```
DbRecordHandle get_current_message_handle();
```

Returns The handle to the current message.

get_field_handle

Retrieves the handle associated with a field tag.

```
DbFieldHandle get_field_handle(DbFieldTag field_tag)
```

Parameters

`field_tag`	The message field tag (refer to page 65).

Returns The field handle that is associated with `field_tag`.

get_field_text

Retrieves text that is associated with a message field.

```
const char * get_field_text(DbFieldTag field_tag)
```

Parameters	`field_tag`	The field tag (refer to page 65).

Returns Read-only pointer to the text of a message field.

get_first_address

Retrieves a handle to the first address that is associated with a field.

```
DbFieldHandle get_first_address(DbFieldTag field_tag)
```

Parameters	`field_tag`	Tag associated with an address field.

Description Iterate over the list of addresses with `get_next_address`.

Returns Address handle for first address in field.

get_first_attachment

Retrieves first attachment in current message (if any).

```
DbFieldHandle get_first_attachment()
```

Returns A handle to the first attachment that the current message contains, or `DB_INVALID_FIELD_HANDLE` if the message has no attachments.

get_flags

Retrieves message flags for the current message.

```
unsigned int get_flags()
```

Returns A bitmask of the flags for the message; one or more of these return values:

- `MSG_MORE_AVAILABLE_FLAG`
- `MSG_PRIORITY_FLAG`
- `MSG_TX_FLAG`
- `MSG_UNREAD_FLAG`

Note: It is possible for multiple priority level flags to be set on a given message. As a result, you should interpret the return value by ANDing the flags together.

get_message_reference_id

Retrieves the message reference ID, which can be used to reply to the message.

```
bool get_message_reference_id(int & message_reference_id)
```

Parameters	`message_reference_id`	Holder for the message reference ID number.

Returns True if successful; false otherwise.

See also `set_original_message_reference_id` (refer to page 27)

get_message_to_modify

Retrieves the root `databaseRecord` that is associated with the message handle.

```
DatabaseRecord get_message_to_modify(
    DbRecordHandle handle,
    Database ** theDatabase = NULL)
```

Parameters	`handle`	Handle to the message to retrieve.
	`theDatabase`	If specified, this holder parameter is populated with a pointer to the message database object.

Returns The `DatabaseRecord` associated with the message.

Description Modifying the record that this function returns affects the underlying data directly.

get_next_address

Retrieves the next address that is associated with a particular field.

```
DbFieldHandle get_next_address(DbFieldHandle field_handle)
```

Parameters	`field_handle`	Handle to a previous address field.

Returns The field handle that is associated with the next address in the field, or `DB_INVALID_FIELD_HANDLE` if there are no more addresses.

See also `get_first_address`

get_next_attachment

Retrieves handle to next attachment in current message.

```
bool get_next_attachment(DbFieldHandle db_field_handle)
```

Parameters

`db_field_handle`	Handle to previous attachment.

Returns Handle to the next attachment field or `DB_INVALID_FIELD_HANDLE` if there are no more attachments.

get_timestamps

Retrieves the time stamps that are associated with a message.

```
bool get_timestamps(
    TIME & local_timestamp,
    TIME & remote_timestamp)
```

Parameters

`local_timestamp`	Time that the message was received.
`remote_timestamp`	Time that the message was sent.

Returns True if successful; false otherwise.

is_composing_message

Determines if there is a message currently being composed.

```
bool is_composing_message()
```

Returns True if a message is associated with the current task's ID (that is, after `create_message` has been called).

load_first_message

Loads the oldest message in the database.

```
DbRecordHandle load_first_message()
```

Returns Handle to the oldest message in the message database.

load_message

Loads the specified message into the current context.

```
bool load_message(DbRecordHandle db_record_handle)
```

Parameters

`db_record_handle`	Database handle identifying a message.

Returns True if successful; false otherwise.

Description Each thread can have a single message loaded; the thread is referred to as the context. You do not need to load a message to read it, but you do need to load it to modify it.

Loading a message unloads any previous message automatically.

load_next_message

Load next message in the database.

```
DbRecordHandle load_next_message(DbRecordHandle db_record_handle)
```

Parameters

`db_record_handle`	Handle to a message in the database.

Returns Handle to the next message, or `DB_NULL_HANDLE` if this is the last message.

Description With this call and `load_first_message()`, you can iterate over all messages in the database.

query_account_address

Retrieves the email address that is associated with a specified account.

```
bool query_account_address(
    long account_to_query,
    char const * account_buffer_ptr,
    int account_buffer_size)
```

Parameters

`account_to_query`	The account handle for the account.
`account_buffer_ptr`	Pointer to a buffer that contains the email address; if the buffer is too small for the address, the address is truncated to fit.
`account_buffer_size`	Size of the buffer that is indicated by `account_buffer_ptr`.

Returns True if successful; false if the address is unknown or the account handle is invalid.

Description You can retrieve the `account_to_query` handle using the function `get_account_handle_from_message`.

register_add_attachment

Adds a menu string to the menu that is associated with the compose screen and provides a callback when the user clicks that menu item.

```
bool register_add_attachment(
    char const * menu_string,
    bool (* callback_fcn)(void))
```

Parameters

`menu_string`	A string value that is displayed to the user on the menu during the compose message function. When the user selects the menu item, `callback_fn` is called.
`callback_fn`	A function supplied by the application that handles the actual attachment processing; `callback_fn` has a return value of true if the attachment is added to the message successfully.

Returns True if able to register the attachment; false if not able (for example, if there are already 16 attachment types registered).

Description The current maximum number of attachment types is 16.

register_msg_notification

Registers the current task to receive notification of new messages.

```
void register_msg_notification()
```

Description After registering to receive messages with `register_msg_notification`, an application receives `MESSAGE`s from the system when a new message is added to the message database.

For a received message:

- The handheld member is `DEVICE_MESSAGE`
- The event member is `MSG_DATABASE_NOTIFICATION`
- The `SubMsg` is one of the actions from `Database::Action` (such as `ADD`, `DELETE`, or `UPDATE`)
- The `Data[0]` element contains a `DbRecordHandle` for the message

Note: The handle is valid regardless of the action in the `SubMsg`, even if the action is `DELETE`; the message is not deleted until the notification callback completes. Your application could save the message elsewhere to restore it.

register_mvs_menu_item

Adds menu item to the Message View screen (default viewer for messages).

```
bool register_mvs_menu_item(
    int (*callback_fcn)(
        DbRecordHandle record_handle,
        DbFieldHandle field_handle,
        int offset_in_field),
    const char * menuString,
    long bitmask = 0)
```

Parameters

`(* callback_fcn)`	Function invoked when the menu item is selected; the only valid return codes are: `MVS_CONTINUE` (continue processing message normally) `MVS_DELETE` (delete message).
`record_handle`	Parameter to the `callback_fcn`; handle to the current message.
`field_handle`	Parameter to the `callback_fcn`; handle to the current field.
`offset_in_field`	Parameter to the `callback_fcn`; offset into current field.
`menuString`	String to display in the Message View Screen menu.
`bitmask`	Bitmask that determines when not to display the menu item; defaults to 0 (always display). The acceptable values describe field elements; if a bit is set, the menu item is not displayed if that field is in focus when the menu is invoked. Values for each field are as follows: • Attachment — MVS_BITFIELD_ATTACHMENT_FIELD • BCC — MVS_BITFIELD_BCC_FIELD • CC — MVS_BITFIELD_CC_FIELD • Date Sent — MVS_BITFIELD_DATE_SENT_FIELD • Folder — MVS_BITFIELD_FOLDER_FIELD • From — MVS_BITFIELD_FROM_FIELD • Message Body — MVS_BITFIELD_MSG_BODY_FIELD • Message Status — MVS_BITFIELD_MSG_STATUS_FIELD • Subject — MVS_BITFIELD_SUBJECT_FIELD • To — MVS_BITFIELD_TO_FIELD

Returns True if the callback registers successfully; false if the callback fails to register, for example, as a result of an invalid parameter.

register_open_notification

Registers a callback function to be invoked when a user opens a message.

```
bool register_open_notification (int (* callback_fcn)
        (DbRecordHandle record_handle))
```

Parameters

`(* callback_fcn)` A callback function; the function acts on the message whose handle is `record_handle`, and should use either the `RimNetworkMessage` class to act on the data or should use the C-style functions in `msg_api.h`. The return value of this callback function determines the action that is taken by the message application, as shown in this table:

Value	Macro	Meaning
0		use default message UI
0x1	NO_DEFAULT_UI	do not display regular user interface; regular UI does not display if a flag is set
0x4	VIEW_PREVIOUS_MSG	move to previous message
0x6	VIEW_NEXT_MSG	move to the next message
0x8	DELETE_MSG	delete the message
0x10	VIEW_NEXT_UNREAD	move to the next unread message

`record_handle` Parameter to the `callback_fcn`; specifies the handle to the message that the user opened.

Description The maximum number of callbacks that can be registered is `MAX_OPEN_NOTIFICATION`, which is defined 32.

Returns True if the notification is registered successfully; otherwise false.

register_view_attachment

Registers a callback function to open and display the attachment type (deprecated).

```
bool register_view_attachment(
    char const * content_type_prefix,
    ViewAction (* callback_fcn )(
        DbFieldHandle field_handle,
        bool transmit_flag)
    )
```

Parameters

`content_type_prefix`	A string identifying the attachment type; set in the first parameter of `create_attachment()`.
`callback_fcn`	A function that is called when the user requests to view or edit an attachment. The function returns one of the following values: • `NO_ACTION` (do nothing) • `DELETE_ATTACHMENT` (the attachment is deleted from the message) • `DELETE_MESSAGE`
`field_handle` `(parameter to callback_fcn)`	A handle to the attachment.
`transmit_flag` `(parameter to callback_fcn)`	True if the message containing the attachment was composed (but not necessarily sent) on the device.

Returns True if able to register the attachment.

Description The `register_view_attachment_ex()` function is deprecated. Use `register_view_attachment_ex()` instead. Do not mix calls to the deprecated and extended functions.

register_view_attachment_bitmap

Associates a bitmap with an attachment type.

```
bool register_view_attachment_bitmap(
    char const * content_type,
    BitMap const * bitmap_ptr)
```

Parameters

`content_type`	The content type to associate with the `BitMap`.
`bitmap_ptr`	A pointer to a `BitMap` resource to associate with the named content type.

Returns True if the operation was successful; false otherwise.

Description This function associates a `BitMap` with a particular attachment type. The `BitMap` is displayed instead of the attachment string when the user views a message.

register_view_attachment_ex

Registers the application to display a specific attachment type.

```
bool register_view_attachment_ex(
    char const * content_type,
    ViewAction ( *callback_fcn )( DbFieldHandle field_handle,
            bool transmit_flag ),
    char const * menu_string,
    char const * file_name_ext = "" )
```

Parameters

`content_type`	String that describes a media type/subtype, such as "`Application/vCard`".
`callback_fcn`	Pointer to a function (application) that opens and views the attachment. The function has a return value of one of `NO_ACTION` (do nothing), `DELETE_ATTACHMENT` (delete attachment from the message), or `DELETE_MESSAGE`.
`field_handle` `(parameter to callback_fcn)`	A handle to the attachment; a parameter to the `callback_fcn`.
`transmit_flag` `(parameter to callback_fcn)`	True if the message that contains the attachment was composed (but not necessarily sent) on the handheld; a parameter to the `callback_fcn`.
`menu_string`	The string that appears on the menu when the user selects the attachment. This string should be in the form *Action-Object*, such as "View Vcard" or "Open Attachment".
`file_name_ext`	The file name extension for this kind of attachment; defaults to an empty string, but if the content type is "Application/Octet-Stream" this string must have a value.

Returns True if registration is successful; false if registration fails. For example, the function call can fail for the following reasons:

- The system has already registered its maximum number of attachment viewers. (The `MAX_NUM_VIEW_ATTACHMENT_CALLBACKS_EX` value is 32.)
- The length of `content_type+file_name_ext` is greater than 256.
- The `content_type` is "application/octet-stream" and the `file_name_ext` is empty.

Description Strings are not case-sensitive. The `callback_fcn` should call `get_attachment_data_ex` to retrieve the data.

This function is a replacement for `register_view_attachment` (now deprecated). Do not combine calls to the deprecated and new functions.

remove_add_attachment

Removes the specified menu item on the message compose screen for adding attachments of a specific type.

```
bool remove_add_attachment(
    char const * menu_string,
    bool (* callback_fcn)(void))
```

Parameters

`menu_string`	The menu string specified in a call to `register_add_attachment()`; the case of the letters must be the same.
`(*callback_fcn)`	The callback function as specified in the same call to `register_add_attachment()`.

Returns True if the menu item is removed; false otherwise.

See also `register_add_attachment` (refer to page 21).

save_message

Saves the current message.

```
bool save_message(bool save_as_sent_message)
```

Parameters

`save_as_sent_message`	If set to true, the message status is set to `TX_SENT` (in the Messages list, the message appears as a message sent from the handheld). If set to false, the message status is set to `RX_RECEIVED` (in the Messages list, the message appears as though it is a received message, with an envelope icon beside the message).

Returns True if the operation was successful; false otherwise.

Description This call can be used to artificially insert messages into the system for testing purposes.

send_message

Sends the message (deprecated).

```
bool send_message(bool prompt_user_flag)
```

Parameters

`prompt_user_flag`	When set to true, `send_message` displays a confirmation dialog box to the user before sending the message.

Returns True if the message was sent successfully; false otherwise.

Description This function submits the message to the queue for later transmission.

This function is deprecated. Use `send_message_ex` instead. Do not combine calls to deprecated and new functions.

send_message_ex

Sends the message that is associated with the current task.

```
bool send_message_ex(
    bool prompt_user_flag,
    TASK previousTask )
```

Parameters

`prompt_user_flag`	When set to true, `send_message_ex` displays a confirmation dialog box to the user before sending the message.
`previousTask`	Task to return to after queuing the message; for example, after following a `mailto:` link in a browser, the application should return to the browser task.

Returns True if the message was sent successfully; false otherwise.

set_original_message_reference_id

In a reply, sets the reference ID of the original message.

```
bool set_original_message_reference_id(int message_ref_id)
```

Parameters

`message_ref_id`	Message reference ID of the original message.

Returns True if successful; false otherwise.

See also `get_message_reference_id (refer to page 18)`

set_read_status

Changes read status of a message.

```
void set_read_status(
    DbRecordHandle db_record_handle,
    bool read = true)
```

Parameters

`db_record_handle`	A handle to the message to modify.
`read`	If true, the message is set as "read". If false, the message is set as "unread."

Description The message handle is typically obtained through an event message, or through the `load_first_message`/`load_next_message` pair of calls.

unload_message

Unloads the message from the current context.

```
void unload_message()
```

Description This is a convenience function that frees resources. You are not required to call `unload_message` before loading another message.

Chapter 2
Enhanced Message API

This section provides information on the following topics:

RimNetworkMessage static functions

Create

Creates a new instance of the `RimNetworkMessage` class.

```
Form 1:
    static RimNetworkMessage * Create(DbRecordHandle handle)
```

```
Form 2:
    static RimNetworkMessage * Create()
```

Parameters	`handle`	A handle to an existing message.

Returns Form 1 returns a loaded (write ready) message instance; Form 2 returns a new empty instance.

Description Do not use the `delete` operator on the object that is returned; call the `Destroy` function instead.

Destroy

Destroys an instance of the `RimNetworkMessage` class.

```
static void Destroy(RimNetworkMessage * instanceToDestroy)
```

Parameters	`instanceToDestroy`	A pointer to the `RimNetworkMessage` object to destroy.

Description You must use the `Destroy` function to delete a `RimNetworkMessage` object. Do not use the `delete` operator.

get_RimNetworkMessage_ptr

Retrieves the `RimNetworkMessage` instance that is associated with the current task.

```
static RimNetworkMessage * get_RimNetworkMessage_ptr()
```

Returns Pointer to `RimNetworkMessage` instance that is associated with the current task.

Description The `RimNetworkMessage` instance is created by a call to either of the `create_message*()` calls.

See also `create_message` and `create_message_extended` (refer to page 12).

RimNetworkMessage functions

The following functions are listed alphabetically.

change_attachment_state

Shows or hides the attachment at the given index.

```
virtual void change_attachment_state(bool state, int index)
```

Parameters		
	`state`	If true, the attachment at `index` is shown.
	`index`	The index of the attachment on which to act.

clear_db_record_handle

Clears the `db_record_handle` associated with a message.

```
virtual bool clear_db_record_handle()
```

Returns True if successful; false otherwise.

Description This effectively resets the object to an invalid message.

count_attachments

```
virtual int count_attachments()
```

Returns The number of attachments in the message.

get_all_requested_flag

```
virtual bool get_all_requested_flag()
```

Returns Returns true if all of the message has been requested.

See Also `set_transport_id`

get_add_in_bitfield

Retrieves current value of `add_in_bitfield`.

```
virtual unsigned char get_add_in_bitfield()
```

Returns The current settings of the `add_in_bitfield`.

get_attachment_state

Returns the state (hidden or visible) of the attachment at `index`.

```
virtual bool get_attachment_state(int index)
```

Parameters		
	`index`	The index of the attachment on which to act.

Returns The state of the attachment: true indicates a visible attachment.

get_body_truncated_flag

Determines if the message body was truncated.

```
virtual bool get_body_truncated_flag()
```

Returns True if the body of the message was truncated; false otherwise.

get_create_date

Retrieves the creation date of the message.

```
virtual Date & get_create_date()
```

Returns The creation date of the message.

get_create_time

Retrieves the creation time of the message.

```
virtual Time & get_create_time()
```

Returns The creation time of the message.

get_cursor_position

Returns the position of the cursor in the message (as an offset into the message body, and is only applicable when the user is navigating the message body.)

```
virtual unsigned short get_cursor_position()
```

Returns The position of the cursor in the message.

get_display_date

Retrieves the date as displayed in the message list.

```
virtual Date & get_display_date()
```

Returns The date displayed in the message list.

get_display_time

Retrieves the time as displayed in the message list.

```
virtual Time & get_display_time()
```

Returns The time displayed in the message list.

get_forward_flag

Determines if the message was forwarded.

```
virtual bool get_forward_flag()
```

Returns True if the message was forwarded; false otherwise.

get_gme_transaction_id

Retrieves transaction identifier from the general message envelope (GME) layer.

```
virtual int get_gme_transaction_id()
```

Returns The GME level transaction ID.

Description For internal use only. The RIM GME protocol enables the transfer of compressed and encrypted data to and from the handheld.

See Also `set_forward_flag`

get_hidden_flag

Determines if the message is hidden from the user.

```
virtual bool get_hidden_flag()
```

Returns True if the message is hidden; false otherwise.

get_in_reply_to_reference_id

Retrieves the reference ID for the original message.

```
virtual int get_in_reply_to_reference_id()
```

Returns The reference ID of the original message to which the active message is a reply.

get_in_reply_to_transport_id

For a reply, returns the transport ID of the original message.

```
virtual int get_in_reply_to_transport_id()
```

Returns The `transport_id` of the original message.

get_local_date

Gets the local date of the message.

```
virtual Date & get_local_date()
```

Returns The local date of the message.

Description If the message is an inbound message, `get_local_date` returns the date on which the message was received on the handheld. Otherwise, this function returns the date on which the message was created.

get_local_time

Gets the local time of the message.

```
virtual Time & get_local_time()
```

Returns The local time of the message.

Description If the message is an inbound messages, `get_local_time` returns the time that the message was received. Otherwise, this function returns the time the message was created.

get_message_body_content_id

Retrieves the message body content ID.

```
virtual int get_message_body_content_id()
```

Returns The message body content ID.

Description The message body content ID is a unique identifier of the body portion of the message.

See Also `set_message_body_content_id`

get_message_body_content_length

Retrieves the content length of the message body.

```
virtual int get_message_body_content_length()
```

Returns The content length of the message body.

Description The content length of the message body is the length of the message body that exists on the handheld.

See Also `set_message_body_content_length`

get_message_body_true_content_length

Retrieves the content length of the total message body.

```
virtual int get_message_body_true_content_length()
```

Returns The content length of the total message body.

Description The total message body content length is the length of the complete message, regardless of whether the complete message exists on the handheld.

get_message_error_string

Retrieves any error string that is associated with the failed transmission of the message.

```
virtual char * get_message_error_string()
```

Returns Any error string that is associated with the failed message.

get_message_status

Retrieves the current message status.

```
virtual MessageStatus get_message_status()
```

Returns The current message status:

- `TX_COMPOSING`
- `TX_RETRIEVING_KEY`
- `TX_COMPRESSING`
- `TX_ENCRYPTING`
- `TX_PENDING`
- `TX_SENDING`
- `TX_SENT`
- `TX_RETURNED`
- `TX_MAILBOXED`
- `TX_DELIVERED`
- `TX_READ`
- `TX_ACKED`
- `TX_ERROR`
- `RX_RECEIVING`
- `RX_RECEIVED`
- `RX_ERROR`

Description The message statuses are defined by the `MessageStatus` enumeration.

get_message_status_string

Returns a string that indicates the status of the message, such as, **Composing** for the TX_COMPOSING status.

```
virtual char * get_message_status_string()
```

Returns The status of the message.

get_message_status_icon

Returns an icon that indicates the status of the message (each status corresponds to a status icon).

```
virtual char * get_message_status_icon()
```

Returns A pointer to a character; each icon is a special character.

get_more_available_flag

Determines whether more of the message is available.

```
virtual bool get_more_available_flag()
```

Returns True if more of the message is available; false otherwise.

Description The user can retrieve the remainder of the message by selecting **More**.

See also `set_transport_id`

get_more_requested_flag

Determines whether more of the message has been requested.

```
virtual bool get_more_requested_flag()
```

Returns True if more of the message has been requested; false otherwise.

Description The first 2 KB of a message is forwarded to the handheld. This function determines whether the user has selected the **More** menu item to request the remaining message content, which is delivered in 2-KB blocks up to a maximum of 32 KB.

See also `set_more_requested_flag` (refer to page 47)

get_more_request_sent

Determines whether a More request has been sent.

```
virtual bool get_more_request_sent ( void ) ;
```

Returns Returns true if a More request has been sent; otherwise false.

Description The first 2 KB of a message is forwarded to the handheld. The user can select the **More** menu item to request the remaining message content, which is delivered in 2-KB blocks up to a maximum of 32 KB. This function determines whether the user's request has been sent.

See also `set_more_request_sent` (refer to page 48)

get_notification_level

Retrieves the notification level for the message.

```
virtual int get_notification_level()
```

Returns An integer from 0 to 7 indicating the notification level of the message.

Description Notification levels are as follows:

- 1 - highest priority
- 2-6 - lower numbers indicate higher priority
- 7 - disables notification

See also `set_transport_id` (refer to page 51)

get_original_service_record_id

Retrieves the service record ID.

```
virtual DbRecordId get_original_service_record_id()
```

Returns The service record ID.

Description The service record ID uniquely identifies the service that sent the original message to which the user is replying or forwarding.

get_priority_flag

Determines whether this message has been marked with priority.

```
virtual bool get_priority_flag()
```

Returns True if this message is marked with priority; false otherwise.

get_received_message_reference_id

Retrieves the message layer reference ID of the message.

```
virtual int get_received_message_reference_id()
```

Returns If the message is inbound, this function returns the message layer reference ID of the message; if the message is outbound, this function returns 0.

get_recipient_list_byte

Retrieves the recipient list byte for the message.

```
virtual unsigned char get_recipient_list_byte()
```

Returns A bitfield for the message as follows:

```
[ h | g | f | e | d | c | b | a ]
```

Description The returned bitfield identifies various properties of the message's recipient list, where:

`a`=1 if the handheld address is in the **To**: field.

`b`=1 if the handheld address is in the **Cc**: field.

`c`=1 if the handheld address is in the **Bcc**: field.

`d`=1 if the **To**: field is truncated.

`e`=1 if the **Cc**: field is truncated.

`f`=1 if the **Bcc**: field is truncated.

`g` and `h` are reserved.

A handheld address is an address in the form 15000000 (Mobitex) or 88000000 (DataTAC).

get_remote_date

Retrieves the remote date of the message.

```
virtual Date & get_remote_date()
```

Returns The remote date of the message.

Description If the message is an inbound message, `get_remote_date` returns the date on which the message was sent. Otherwise, this function returns the date on which the message was created.

get_remote_time

Retrieves the remote time of the message.

```
virtual Time & get_remote_time()
```

Returns The remote time of the message.

Description If the message is an inbound message, `get_remote_time` returns the time on which the message was sent. Otherwise, this function returns the time on which the message was created.

get_reply_allowed_flag

Determines whether a reply to this message is permitted.

```
virtual bool get_reply_allowed_flag()
```

Returns True if the message can be replied to; false otherwise.

See also `set_transport_id` (refer to page 51)

get_reply_flag

Determines if this message was replied to.

```
virtual bool get_reply_flag()
```

Returns True if the message was replied to; false otherwise.

get_saved_flag

Determines whether this message is saved.

```
virtual bool get_saved_flag()
```

Returns True if the message is saved; false otherwise.

Description A message is saved if it has been copied to the Saved Messages folder.

See also `set_saved_flag()`

get_service_record_id

Retrieves the service record identifier for the message.

```
virtual DbRecordId get_service_record_id()
```

Returns The service record ID for the message.

Description The `service_record_id` uniquely identifies the service for which this message is destined.

get_transport_id

Retrieves the transport identifier for the message.

```
virtual int get_transport_id()
```

Returns The transport identifier (message layer identifier) for the message.

See also `set_transport_id` (refer to page 51)

get_tx_flag

Determines whether this message was sent (as opposed to received).

```
virtual bool get_tx_flag()
```

Returns True if the message was sent; false otherwise.

get_unread_flag

Determines whether this message has been opened by the user.

```
virtual bool get_unread_flag()
```

Returns True if the message has been read; false if it is unread.

isFiled

Determines if the message has been filed.

```
virtual bool isFiled()
```

Returns True if the message has been filed; false otherwise.

See also `setIsFiled` (refer to page 45)

load

Loads the data from the persistent store.

```
virtual bool load()
```

moved_in_flash

Determines whether data associated with the message has been moved into flash memory.

```
virtual bool moved_in_flash()
```

Returns True if data associated with the message has moved into flash memory; false otherwise.

Description A true return from this function should be followed by a `reconstruct` using the associated `DbRecordHandle`.

reconstruct

Rebuilds the message using the provided `DbRecordHandle`.

```
virtual bool reconstruct(DbRecordHandle db_record_handle)
```

Parameters	`db_record_handle`	The handle to the message.

Returns True if successful; false otherwise.

Description Use this function to rebuild the message using the same handle rather than creating a new object.

save

Saves the data to the persistent store.

```
virtual DbRecordHandle save()
```

set_add_in_bitfield

Sets special flags that the server uses.

```
virtual void set_add_in_bitfield(unsigned char new_add_in_bitfield)
```

Parameters `new_add_in_bitfield` The bitwise OR of the special flags used by the server, used with `set_in_reply_to_reference_id`. Valid values:

- `0x00`

  Equivalent to [h|g|f|e|d|c|b|a] .
- `0x01`

  a = 1 — add text from the original message.
- `0x02`

  b = 1 — add attachments from the original message

Description The `add_in_bitfield` describes special flags used by the server, used with `set_in_reply_to_reference_id()`.

See also `get_add_in_bitfield`

set_all_address_status

Sets the status of all addresses to the same value.

```
virtual void set_all_address_status(MessageStatus new_message_status)
```

Parameters		
	`new_message_status`	The status to set for all addresses; one of the following values: • `TX_COMPOSING` • `TX_RETRIEVING_KEY` • `TX_COMPRESSING` • `TX_ENCRYPTING` • `TX_PENDING` • `TX_SENDING` • `TX_SENT` • `TX_RETURNED` • `TX_MAILBOXED` • `TX_DELIVERED` • `TX_READ` • `TX_ACKED` • `TX_ERROR` • `RX_RECEIVING` • `RX_RECEIVED` • `RX_ERROR`.

Description Message transmission fails for each message recipient. With this function, the status of all recipients can be set to the same value.

set_all_requested_flag

Sets a flag to indicate that the user has requested more of the message.

```
virtual void set_all_requested_flag(bool more_requested_flag)
```

Parameters		
	`more_requested_flag`	Set to true if more of the message has been requested; false otherwise.

set_body_truncated_flag

Sets a flag to indicate that the message body has been truncated.

```
virtual void set_body_truncated_flag(bool new_body_truncated_flag)
```

Parameters		
	`new_body_truncated_flag`	Set to true if the message body as been truncated; false otherwise.

set_cursor_position

Sets the current cursor position.

```
virtual void set_cursor_position(unsigned short cursor_position)
```

Parameters `cursor_position` The current cursor position, as an offset into the message body.

set_forward_flag

Sets a flag to indicate that the message has been forwarded.

```
virtual void set_forward_flag(bool forward_flag)
```

Parameters `forward_flag` True if the message has been forwarded; false otherwise.

See also `get_forward_flag`

set_gme_transaction_id

Sets the transaction identifer in the general message envelope (GME) layer.

```
virtual void set_gme_transaction_id(int new_gme_transaction_id)
```

Parameters `new_gme_transaction_id`

Description For internal use only. The RIM GME protocol enables the transfer of compressed and encrypted data to and from the handheld.

See also `get_gme_transaction_id`

set_hidden_flag

Sets a flag to indicate if the message is marked as hidden from the user.

```
virtual void set_hidden_flag()
```

Warning: This function is irreversible.

set_in_reply_to_reference_id

Sets the `in_reply_to_reference_id`.

```
virtual void set_in_reply_to_reference_id(int refId)
```

Parameters	`refid`	The reference ID of the message.

Description The `in_reply_to_reference_id` uniquely identifies the message to which the user is replying.

set_in_reply_to_transport_id

Sets the `in_reply_to_transport_id.`

```
virtual void set_in_reply_to_transport_id(
    int new_in_reply_to_transport_id)
```

Parameters	`new_in_reply_to_transport_id`	The transport ID of the message.

Description The `in_reply_to_transport_id` uniquely identifies the message to which this message is a reply.

setIsFiled

Sets whether the message has been filed.

```
virtual void setIsFiled(bool filed)
```

Parameters	`filed`	True if this message has been filed; false otherwise.

set_local_time

Sets the local time for the message.

```
virtual void set_local_time(TIME & time)
```

Parameters	`time`	The local time for the message.

set_message_body_content_id

Sets the message body content ID.

```
virtual void set_message_body_content_id(
    int new_message_body_content_id)
```

Parameters	`new_message_body_content_id`	The message body content ID.

Description The message body content ID uniquely identifies the body portion of the message.

set_message_body_content_length

Sets the length of the message body on the handheld.

```
virtual void set_message_body_content_length(
    int new_message_body_content_length)
```

Parameters	`new_message_body_content_length`	The content length of the message body.

Description The content length of the message body is the length of the message body that exists on the handheld. The message body content length can be truncated.

set_message_body_true_content_length

Sets the length of the total message body.

```
virtual void set_message_body_true_content_length(
    int new_message_body_true_content_length)
```

Parameters	`new_message_body_true_content_length`	The total length of the message body.

Description The total length of the message body content is the length of the complete message, regardless of whether the complete message exists on the handheld. The total message body content length cannot be truncated.

See also `set_message_body_content_length` (refer to page 46)

set_more_of_this_part_id

Specifies the part of the message (body or attachment) for which to retrieve more.

```
virtual void set_more_of_this_part_id( int new_more_of_this_part_id )
```

Parameters	`new_more_of_this_part_id`	Integer associated with either the body or attachment part of the message.

Description Each message has a body part and can have one or more attachment parts. The integer associated with the message body part is available using `get_message_body_content_id`.

See also `get_message_body_content_id` (refer to page 36)

set_message_status

Set the message status.

```
virtual void set_message_status(MessageStatus status)
```

Parameters	`status`	The message status; one of the following values: • `TX_COMPOSING` • `TX_RETRIEVING_KEY` • `TX_COMPRESSING` • `TX_ENCRYPTING` • `TX_PENDING` • `TX_SENDING` • `TX_SENT` • `TX_RETURNED` • `TX_MAILBOXED` • `TX_DELIVERED` • `TX_READ` • `TX_ACKED` • `TX_ERROR` • `RX_RECEIVING` • `RX_RECEIVED` • `RX_ERROR`

Description The message statuses are defined by the `MessageStatus` enumeration.

set_more_available_flag

Indicates if additional contents are available for a message.

```
virtual void set_more_available_flag(bool more_available_flag)
```

Parameters	`more_available_flag`	True if More requests are permitted; false otherwise.

Description The first 2 KB of a message is forwarded to the handheld. This function specifies if additional message contents are available so that the user can select the **More** menu item to request more of the message.

See also `get_more_available_flag` (refer to page 46)

set_more_requested_flag

Indicates if More has been requested for this message.

```
virtual void set_more_requested_flag(bool more_requested_flag)
```

Parameters	`more_requested_flag`	True if More has been requested for this message; false otherwise.

Description The first 2 KB of a message is forwarded to the handheld. This function sets a flag to indicate that the user has selected the **More** menu item to request the remaining message content, which is delivered in 2-KB blocks up to a maximum of 32 KB.

See also `See also` (refer to page 46)

set_more_request_sent

Indicates if a “more” request has been sent.

```
virtual void set_more_request_sent ( bool more_requested_sent_flag )
```

Parameters

`more_requested_sent_flag`	True if “more” request has been sent. False if “more” request has not been sent.

Description The first 2 KB of a message is forwarded to the handheld. The user can select the **More** menu item to request the remaining message content, which is delivered in 2 KB blocks up to a maximum of 32 KB. This function indicates whether or not the user’s request has been sent.

See also `get_more_request_sent` (refer to page 38)

set_notification_level

Sets the notification level for this message.

```
virtual void set_notification_level(int notification_level)
```

Parameters

`notification_level`	The notification level to set: • 1 - highest priority • 2 to 6 - lower numbers indicate higher priority • 7 - disables notification

Description Messages set with notification level 1 will appear in bold in the Messages list on the handheld. Messages set with normal notification appear with normal text formatting in the message list. Messages set with a no notification level do not trigger a notification to the user, such as handheld vibration or tone.

set_original_service_record_id

Sets the original service record ID.

```
virtual void set_original_service_record_id(DbRecordId rid)
```

Parameters

`rid`	The record ID of the original service for the message.

Description The original service record ID uniquely identifies the service that sent the original message to which the user is replying or forwarding.

set_priority_flag

Sets the priority of the message.

```
virtual void set_priority_flag(bool priority_flag)
```

Parameters

`priority_flag`	If true, this is a priority message; if false, it is a normal message.

set_received_message_reference_id

Sets the received message reference ID.

```
virtual void set_received_message_reference_id(
        int new_received_message_reference_id)
```

Parameters

`new_received_message_reference_id`	The message reference ID.

Description This function is only applicable to inbound messages.

set_recipient_list_byte

Sets the recipient list byte.

```
virtual void set_recipient_list_byte(
    unsigned char new_recipient_list_byte)
```

Parameters

`new_recipient_list_byte`	The recipient list byte to set.

Description The recipient list byte identifies various properties of the message's recipient list, in the following bitfield:

```
[ h | g | f | e | d | c | b | a ]
```

`a`=1 if the handheld address in the **To**: field.

`b`=1 if the handheld address in the **Cc**: field.

`c`=1 if the handheld address in the **Bcc**: field.

`d`=1 if **To**: field is truncated.

`e`=1 if **Cc**: field is truncated.

`f`=1 if **Bcc**: field is truncated.

`g` and `h` are reserved.

A handheld address is an address of the form 15000000 (Mobitex) or 88000000 (DataTAC).

set_remote_time

Sets the remote time for the message.

```
virtual void set_remote_time(TIME & time)
```

Parameters `time` Remote time for the message.

Description If the message is an inbound message, this function sets the time it that it was sent by the server. If the message is not an inbound message, this function sets the time when the message was sent by the user.

set_reply_allowed_flag

Indicates if a reply to this message is permitted.

```
virtual void set_reply_allowed_flag(bool reply_allowed_flag)
```

Parameters `reply_flag` True if a reply to this message is permitted; false otherwise.

set_reply_flag

Indicates if the message has been replied to.

```
virtual void set_reply_flag(bool reply_flag)
```

Parameters `reply_flag` True if the user has replied to the message; false otherwise.

set_saved_flag

Indicates if this message has been saved.

```
virtual void set_saved_flag(bool flag = true)
```

Parameters `flag` True if the message has been saved; false otherwise.

Description A message is saved if it has been copied into the Saved Messages folder.

set_service_record_id

Sets the service record identifier.

```
virtual void set_service_record_id(DbRecordId rid)
```

Parameters `rid` The service record ID for the message.

Description The `service_record_id` uniquely identifies the service for which this message is destined.

set_transport_id

Sets the transport identifier of the message.

```
virtual void set_transport_id(int new_transport_id)
```

Parameters

`new_transport_id`	The transport identifier of the message.

See also `get_transport_id`

set_unread_flag

Indicates if this message is unread (unopened).

```
virtual void set_unread_flag(bool flag)
```

Parameters

`flag`	True if the message is unread; false if it is read.

usesPeerService

Determines whether a message uses a peer service.

```
virtual bool usesPeerService()
```

Returns True if the message uses a peer service; false otherwise.

Message field functions

The following functions are listed alphabetically.

delete_field

Deletes a field.

```
virtual void delete_field(DbFieldHandle field_handle)
```

Parameters	`field_handle`	Handle to the field to be deleted.

delete_fields

Deletes a group of fields identified by a tag.

```
virtual void delete_fields(DbFieldTag field_tag)
```

Parameters	`field_tag`	The group of fields to delete (refer to page 65).

See also `get_tag_text` (refer to page 55)

get_field_data_length

Retrieves the length of a data field.

```
virtual int get_field_data_length(DbFieldHandle field_handle)
```

Parameters	`field_handle`	Handle to the field.

Returns The length of the data field.

get_field_data_ptr

Retrieves a pointer to a field's data.

```
virtual void * const get_field_data_ptr(DbFieldHandle field_handle)
```

Parameters	`field_handle`	Handle to the field for which to retrieve a pointer.

Returns Pointer to the field data.

get_field_tag

Retrieves the tag for a field.

```
virtual DbFieldHandle get_field_tag(DbFieldHandle field_handle)
```

Parameters

`field_handle`	Handle for the field to return.

Returns Tag for the field specified by `field_handle`.

get_first_field_handle

Retrieves the handle to the first field of the associated database record.

```
Form 1:
    virtual DbFieldHandle get_first_field_handle()
Form 2:
    virtual DbFieldHandle get_first_field_handle(DbFieldTag field_tag)
```

Parameters

`field_tag`	Tag that describes the type of field to retrieve (refer to page 65).

Returns Handle to the first field of the associated message.

Description Form 1 retrieves the first field handle, regardless of its type. Form 2 retrieves the first field handle to a particular type.

get_next_field_handle

Retrieves the handle to the next field of the associated message, after the specified field.

```
Form 1:
    virtual DbFieldHandle
        get_next_field_handle(DbFieldHandle field_handle)

Form 2:
    virtual DbFieldHandle
        get_next_field_handle(DbFieldHandle field_handle,
            DbFieldTag field_tag)
```

Parameters

`field_handle`	Handle for the starting field.
`field_tag`	Tag that describes the type of field to retrieve (refer to page 65).

Returns Handle to the next field in the database.

Description Form 1 retrieves the next field handle, regardless of its type. Form 2 retrieves the next field handle to a particular type.

Text message field functions

The following functions are listed alphabetically.

append_field_text

Appends specified text to the field.

```
virtual bool append_field_text(
    DbFieldHandle field_handle,
    char const * text,
    int text_length = DB_TEXT)
```

Parameters

`field_handle`	Handle to the field to append the text to.
`text`	Text to append.
`text_length`	Length of the text, or DB_TEXT if null terminated.

Returns True if successful; false otherwise.

append_tag_text

Appends text to a field that is identified by tag.

```
virtual bool append_tag_text(
    DbFieldTag field_tag,
    char const * text,
    int text_length = DB_TEXT)
```

Parameters

`field_tag`	Tag that describes the type of field for which to search (refer to page 65).
`text`	Text to append.
`text_length`	Length of the text, or `DB_TEXT` if null terminated.

Returns True if successful.

create_field

Creates a field.

```
virtual DbFieldHandle create_field(
    DbFieldTag field_tag,
    char const * text = NULL,
    int text_length = DB_TEXT)
```

Parameters

`field_tag`	Tag of the field to create (refer to page 65).
`text`	Text to append.
`text_length`	Length of the text, or DB_TEXT if null terminated.

Returns Handle for the field to create.

get_field_text

Retrieves the text of a field.

```
virtual char * get_field_text(DbFieldHandle field_handle)
```

Parameters

`field_handle`	Handle to the field for which to retrieve text (refer to page 65).

Returns The text of the field.

get_tag_text

Retrieve the text of a field tag.

```
virtual char * get_tag_text(DbFieldTag field_tag)
```

Parameters

`field_tag`	Field tag for which to retrieve text (refer to page 65).

prepend_field_text

Adds data to the start of a field identified by a handle.

```
virtual bool prepend_field_text(
    DbFieldHandle field_handle,
    char const * text,
    int text_length = DB_TEXT)
```

Parameters

`field_handle`	handle to the field to which to prepend text.
`text`	Text to prepend.
`text_length`	Length of the text, or `DB_TEXT` if null terminated.

Returns True if successful; false otherwise.

prepend_tag_text

Prepends data to a field identified by tag.

```
virtual bool prepend_tag_text(
    DbFieldTag field_tag,
    char const * text,
    int text_length = DB_TEXT)
```

Parameters

`field_tag`	Tag of the field for which to prepend text (refer to page 65).
`text`	Text to prepend.
`text_length`	Length of the text, or `DB_TEXT` if null terminated.

Returns True if successful; false otherwise.

replace_field_data

Replaces data at a given offset for a given field.

```
virtual bool replace_field_data(
    DbFieldHandle field_handle,
    int offset,
    const void * data_ptr,
    int data_length = DB_TEXT)
```

Parameters

`field_handle`	Handle to the field to replace.
`offset`	Offset into the field to start replacing data.
`data_ptr`	Pointer to the data used as replacement.
`data_length`	Length of the data, or `DB_TEXT` if null terminated.

Returns True if successful; false otherwise.

replace_tag_data

Replaces data at a given offset for a given field tag.

```
virtual bool replace_tag_data(
    DbFieldTag field_tag,
    int offset,
    const void * data_ptr,
    int data_length = DB_TEXT)
```

Parameters

`field_tag`	Tag to replace (refer to page 65).
`offset`	Offset into the field tag to start replacing data.
`data_ptr`	Pointer to the data that is used as a replacement.
`data_length`	Length of the data, or `DB_TEXT` if null terminated.

Returns True if the replace operation was successful; false otherwise.

set_field_text

Sets the text in a field.

```
virtual DbFieldHandle set_field_text(
    DbFieldHandle field_handle,
    char const * text,
    int text_length = DB_TEXT)
```

Parameters

`field_handle`	Handle to the field for which to set text.
`text`	Text to set.
`text_length`	Length of the text, or `DB_TEXT` if null terminated.

Returns A handle to the field that has been modified.

set_tag_text

Sets the text in a field tag.

```
virtual DbFieldHandle set_tag_text(
    DbFieldTag field_tag,
    char const * text,
    int text_length = DB_TEXT)
```

Parameters

`field_tag`	Handle to the field tag for which to set text (refer to page 65).
`text`	Text to prepend.
`text_length`	Length of the text, or `DB_TEXT` if null terminated.

Returns A handle to the field tag that has been modified.

Address field functions

The following functions are listed alphabetically.

add_attachment_data

Adds attachment data to the message.

```
Form 1: virtual bool add_attachment_data(
    DbFieldHandle field_handle,
    char const * data,
    int data_length = DB_TEXT)
```

```
Form 2: virtual bool add_attachment_data(
    DbFieldHandle field_handle,
    DataBuffer data_buffer,
    int data_length)
```

Parameters

`field_handle`	The handle to the attachment field.
`data`	A pointer to data. If the data is a string, data_length can be set to `DB_TEXT`.
`data_length`	In form 1, the length of the data, or `DB_TEXT` if the data is a null terminated string In form 2, the length of the data contained in the data buffer structure; a value of 0 or `DB_TEXT` is illegal.
`data_buffer`	A `DataBuffer` object.

Returns True if the operation was successful; false otherwise.

create_address

Creates a new address.

```
virtual DbFieldHandle create_address(
    DbFieldTag field_tag,
    char const * address,
    AddressFieldType address_type,
    char const * name = "",
    DbFieldHandle field_handle = DB_INVALID_FIELD_HANDLE,
    int address_length = DB_TEXT,
    int name_length = DB_TEXT)
```

Parameters

`field_tag`	The type of address field to create (refer to page 65).
`address`	The address.
`address_type`	The type of address.
`field_handle`	A handle to an existing address field—this causes the method to modify an existing address.
`address_length`	Length of the address field, or `DB_TEXT` if null terminated.
`name_length`	The length of the name field, or `DB_TEXT` if null terminated.

Returns The field handle for the address.

create_attachment

Creates an attachment for the message.

```
virtual DbFieldHandle create_attachment(
    char const * content_string,
    char const * content_string_concatenation = "",
    char const * data = "",
    int data_length = DB_TEXT,
    DbFieldHandle field_handle = DB_INVALID_FIELD_HANDLE)
```

Parameters

`content_string`	The content string to use.
`content_string_concatenation`	An extension appended to the `content_string`.
`data`	The attachment data.
`data_length`	The length of the attachment data; if the data is a null terminated string, `DB_TEXT` is appropriate.
`field_handle`	If not `DB_INVALID_FIELD_HANDLE`, the parameters are applied to an existing attachment field identified by the `field_handle` parameter.

Returns The field handle to the attachment.

get_address

Retrieves the address at the specified `field_handle`.

```
virtual char * get_address(DbFieldHandle field_handle)
```

Parameters

`field_handle`	The handle to the address to retrieve.

Returns The address string.

get_address_error_string

Retrieves the error string that is associated with a failed transmission to a particular address.

```
virtual char * get_address_error_string(DbFieldHandle field_handle)
```

Parameters

`field_handle`	The handle to the address.

Returns The error string.

get_address_name

Retrieves the descriptive name for a given address.

```
virtual char * get_address_name(DbFieldHandle field_handle)
```

Parameters `field_handle` The handle to the address for which to retrieve the descriptive name.

Returns The address descriptive name.

get_address_status

Gets the transmit status for a particular destination address.

```
virtual MessageStatus get_address_status(DbFieldHandle field_handle)
```

Parameters `field_handle` The handle to the address field.

Returns The MessageStatus code.

get_address_type

Gets the address type.

```
virtual AddressFieldType get_address_type(DbFieldHandle field_handle)
```

Parameters `field_handle` The handle to the address field.

Returns The `AddressFieldType` of the address field.

get_attachment_content_type

Retrieves the content type for an attachment.

```
virtual char * get_attachment_content_type(DbFieldHandle field_handle)
```

Parameters `field_handle` Handle to an attachment field.

Returns A null-terminated string identifying the attachment content type.

get_attachment_data

Retrieves the attachment data.

```
virtual char * get_attachment_data(
    DbFieldHandle field_handle,
    int * data_length_ptr = NULL)
```

Parameters

`field_handle`	Handle to an attachment field.
`data_length_ptr`	An optional out parameter, stores the length of the retrieved data.

get_attachment_filename

Retrieves the filename that is associated with an attachment.

```
virtual char * get_attachment_filename(DbFieldHandle field_handle)
```

Parameters

`field_handle`	Handle to an attachment field.

Returns A null-terminated string identifying the attachment file name.

set_address

Sets an address for the message.

```
virtual DbFieldHandle set_address(
    DbFieldHandle field_handle,
    char const * address,
    AddressFieldType address_type,
    char const * name = "",
    int address_length = DB_TEXT,
    int name_length = DB_TEXT)
```

Parameters

`field_handle`	Handle to an existing address field if the method was created using `create_address`.
`address`	A valid address.
`address_type`	The type of address.
name	The descriptive name for display to the user.
`address_length`	Length of the address data, or `DB_TEXT` if null terminated.
`name_length`	The length of the name data, or `DB_TEXT` if null terminated.

Returns The field handle for the address.

set_delivery_status_string

Sets the status string.

```
virtual bool set_delivery_status_string(
    char const * delivery_status_string,
    int delivery_status_string_length = DB_TEXT,
    DbFieldHandle field_handle = DB_INVALID_FIELD_HANDLE)
```

Parameters

`delivery_status_string`	The string to set.
`delivery_status_string_length`	The length of the string, or `DB_TEXT` to indicate a null-terminated string.
`field_handle`	Optional; the field handle to the status field.

Returns True if successful; false otherwise.

set_name

Sets the descriptive name for a particular address field.

```
virtual bool set_name(
    DbFieldHandle field_handle,
    char const * name,
    int name_length = DB_TEXT)
```

Parameters

`field_handle`	Handle to the address field.
name	The string to set for the field name.
name_length	The length of the name string, or `DB_TEXT` if the name parameter is null terminated.

Returns True if successful; false otherwise.

Chapter 3
Field tags

The following field tags are defined in `MessageFieldType.h`.

Type	Field Tag
Text fields	SUBJECT_FIELD_TAG MESSAGE_FIELD_TAG MESSAGE_ID_FIELD_TAG ORIGINAL_MESSAGE_ID_TAG DELIVERY_DATA_TAG TRANSPORT_ID_FIELD_TAG; GLOBAL_DELIVERY_DATA_TAG ORIGINAL_MESSAGE_FIELD_TAG GLOBAL_ERROR_MESSAGE_FIELD_TAG
Attachment fields	ATTACHMENT_FIELD_TAG NUM_ATTACHMENTS_FIELD_TAG HIDDEN_ATTACHMENT_FIELD_TAG
Address fields	TO_FIELD_TAG CC_FIELD_TAG BCC_FIELD_TAG SENDER_FIELD_TAG FROM_FIELD_TAG REPLY_TO_FIELD_TAG DISPLAY_TO_FIELD_TAG MHP_FORWARD_TO_FIELD_TAG DISPLAY_CC_FIELD_TAG DISPLAY_BCC_FIELD_TAG
Message view fields	LINK_FIELD_TAG DYNAMIC_MESSAGE_FIELD_TAG STATUS_FIELD_TAG FOLDER_FIELD_TAG PRIORITY_FIELD_TAG ERROR_FIELD_TAG DYNAMIC_SEPARATOR_FIELD_TAG DELIVERY_STATUS_STRING_TAG MESSAGE_ICON_TAG

Index of functions