# Kopaon PHP Language Binding

***Release 7.2.1***

**Kopano BV**

**Oct 09, 2017**

**Edition 8.0.0 - The Kopano Team**

This document, the Kopano PHP MAPI Language Interface document, describes how to fully access Kopano's messaging api (MAPI) via the interface provided by php-mapi.

CHAPTER 1

# Introduction

The PHP language binding (php-mapi) provides a PHP interface to the messaging API used by Kopano, MAPI.

Although not all MAPI functions and interfaces are supported, most functions have a php counterpart in this extension. Using php-mapi, users can create php-based e-mail and calendaring systems and interfaces with existing php projects, using the MAPI functions like a normal MAPI program.

## 1.1 Other documentation

The complete PHP-MAPI extension was based on the MAPI specification by Microsoft, which can be downloaded from MSDN at no charge. The functions defined in this document, are usually directly related to their MAPI counterpart and most information provided by the Microsoft MAPI specification is also applicable to the functions defined in the reference.

## 1.2 Target audience

This document is mainly targeted at web developers wanting to MAPI scripting through PHP, and PHP developers who want to start doing MAPI development.

Also, general MAPI practices, hierarchies and properties apply to the php-mapi extension. You are assumed to have basic knowledge of MAPI and its object structure before reading this reference manual.

# Conventions

This chapter will describe the PHP-MAPI API and how to use it. It will describe also some tricks that can be applied to make programming easier.

## 2.1 Properties

Every object in MAPI has several properties set. The properties contains information about the object like the name, the unique id, etc... A property tag consists of a type and a unique id, the property type and the property id. A property tag is made as in the following example:

**Property definitions**

```
define ('PR_SUBJECT', mapi_prop_tag(PT_TSTRING, 0x0037);
// This will result in a constant PR_SUBJECT with the value 0x001E0037
```

Within the PHP-code all of the properties are defined with a PR_ prefix to indicatie it is a property. All type codes are prefixed with PT_ (Property Type).

The function `mapi_prop_tag` accepts the property type and the property id and returns a unique code that indicates the property tag. This code is a simple long where the first 16 bits are the type and the last 16 bits are the ID.

Possible property types are:

Special properties:

- PT_NULL
- PT_ERROR

Single properties:

- PT_I2
- PT_LONG
- PT_FLOAT
- PT_DOUBLE
- PT_APPTIME
- PT_BOOLEAN
- PT_STRING8
- PT_SYSTIME
- PT_BINARY
- PT_CLSID

Multi value properties:

- PT_MV_I2
- PT_MV_LONG
- PT_MV_R4
- PT_MV_DOUBLE
- PT_MV_APPTIME
- PT_MV_SYSTIME
- PT_MV_STRING8
- PT_MV_BINARY
- PT_MV_CLSID

Not supported:

- PT_CURRENCY
- PT_UNICODE
- PT_OBJECT
- PT_I8
- PT_MV_CURRENCY
- PT_MV_UNICODE
- PT_MV_I8

Special properties for rules table:

- PT_ACTIONS
- PT_SRESTRICTION

## 2.2 Checking errors

When an array of properties is retrieved with the function `mapi_getprops` by example there could some problems with the properties. Because of optimalization MAPI has a restricition on the size of a property. When by example a PR_BODY is too big the error MAPI_E_NOT_ENOUGH_MEMORY will be returned instead of the actual data.

The type of this property will be set to PT_ERROR instead of PT_TSTRING. The value of the property will be the code of the error that occured. The numeric value of the property tag will be 0x000A0037 which is different from 0x001E0037. This requires a special way to check if there is an error and find out what the error code is.

The following program listing gives an example how to check the property on an error and how to get the error from the property:

**Checking property errors in an object**

```
// get properties
$msg_props = mapi_getprops($message);

if (array_key_exists($msg_props, PR_BODY) {
// there is a body with no error, display body here
} else {
        $pr_body_error = mapi_prop_tag(PT_ERROR, mapi_prop_id(PR_BODY));
        if (array_key_exists($msg_props, $pr_body_error)) {
                // found an error, now check which
                switch($msg_props[$pr_body_error]) {
                        case MAPI_E_NOT_ENOUGH_MEMORY:
```

```
                        // found but too big, use mapi_openproperty the get the␣
↪data.
                        break;
                default:
                        // other error.
                        break;
                }
        } else {
        // No error found, the body doesn't exist in this object.
        }
}
```

Also, when checking error values (such as 0x80040107, MAPI_E_NOT_FOUND), you must use the mapi_is_error function to check whether the valus is actually an error. This is due to the fact that 0x800xxxxx numbers are not handed as unsigned numbers within PHP.

**Checking for errors or warnings**

```
if(mapi_is_error($msg_props[$pr_body_error])) {
        echo "error!\n";
}
```

When a function returns the value 'false', it means the function has failed. To retrieve the MAPI error, you can use the mapi_last_hresult() function, to get the last returned HRESULT value of a MAPI function. Example:

**Checking last MAPI returned error code**

```
$inbox = mapi_msgstore_getreceivefolder($store);
if ($inbox == false) {
        print "no inbox available!\n";
        printf("last result: %x\n",mapi_last_hresult());
}
```

## 2.3 Restrictions

Restrictions limit the rows queried using a set of conditions. This way messages can be selected on specific properties, such as the existance of a certain property, a property's value larger than a certain value, etc..

A restriction is specified with an array with one value with the restriction type and one value with an array with the restriction specific properties.

Restriction types:

| Restriction type | Description |
|---|---|
| RES_AND | Takes an array with restrictions that must all be valid. |
| RES_OR | Expects an array with restrictions of which either one can be valid. |
| RES_NOT | Wants an array with a single restriction that must not be valid. |
| RES_CONTENT | Fuzzy search a property. The array must contain: FUZZYLEVEL, ULPROPTAG and VALUE. VALUE must be a string or binary. VALUE => array (tag => value) |
| RES_PROPERTY | Takes an array with properties of a property which must be valid. This array must contain: RELOP, ULPROPTAG and VALUE. The property and VALUE can have different types. VALUE => array (tag => value) |
| RES_COMPAREPROPS | Compares two props. Expects an array with the indexes: RELOP, ULPROPTAG1 and ULPROPTAG2. |
| RES_BITMASK | Applies a bitmask. The given array must contain: ULPROPTAG, ULTYPE and ULMASK. |
| RES_SIZE | Measures the size of a property. The array must have: ULPROPTAG, RELOP and CB. |
| RES_EXIST | Checks whether a property exists. Takes only ULPROPTAG in its array. |
| RES_SUBRESTRICTION | Places a restriction on a property. This is used on PR_MESSAGE_RECIPIENT properties with an RES_COMMENT restriction. |
| RES_COMMENT | A comment restriction can be used to add comments to the restriction structure, or in a special case with the RES_SUBRESTRICTION. The RES_COMMENT structure holds a RES_PROPERTY restriction, which applies on the given properties list in the structure. |

Retriction operations types:

| Restriction operation | Description |
|---|---|
| RELOP_LT | Less than. |
| RELOP_LE | Less than or equal to. |
| RELOP_GT | Greater than. |
| RELOP_GE | Greater than or equal to. |
| RELOP_EQ | Equal to. |
| RELOP_NE | Not equal to. |
| RELOP_RE | The attachment has just been created. |

FUZZYLEVEL can be one or bitwise more of these:

| Fuzzy level | Description |
|---|---|
| FL_FULLSTRING | The property must be equal to this exact string. |
| FL_SUBSTRING | Full text search for substring. |
| FL_PREFIX | The value must start with this string. |
| FL_IGNORECASE | Case should be ignored. |
| FL_IGNORENONSPACE | Non-spaces should be ignored. |
| FL_LOOSE | The string is allowed to match loosely. |

**Simple restriction example**

```
// Example for restrictions
$testrestriction = Array(RES_OR,
                         Array(
                               Array(RES_PROPERTY,
                                     Array(RELOP => RELOP_LE, ULPROPTAG => PR_BODY,
                                           VALUE => array(PR_BODY =>"test"))),
                               Array(RES_PROPERTY,
                                     Array(RELOP => RELOP_LE, ULPROPTAG => PR_
↪SUBJECT,
                                           VALUE => array(PR_SUBJECT => "bericht
↪")))));

$contents = mapi_table_queryallrows($table, $props, $sizerestriction);

foreach ($contents as $row)
{
        echo $row[PR_SUBJECT];
```

```
        echo "\n";
}
```

**RES_SUBRESTRICTION + RES_COMMENT example**

```
$restriction =
array(RES_SUBRESTRICTION,
      array(ULPROPTAG => PR_MESSAGE_RECIPIENTS, // this is the subobject
         RESTRICTION => array(RES_COMMENT,
            array(PROPS => array(),  // probably need to add outlook data
               RESTRICTION => array(RES_PROPERTY,
                  array(RELOP => RELOP_EQ,
                     ULPROPTAG => PR_SEARCH_KEY,
                        VALUE => array(PR_SEARCH_KEY => "one-off entryid") )
               )
            )
         )
      )
);
```

## 2.4 Attachments

Attachments are used to send a binary file with a email message. There are a few different attachments types:

| Property | Description |
|---|---|
| ATTACH_BY_VALUE | The attachment data is stored within the message object. |
| ATTACH_EMBEDDED_MSG | The attachment is stored as a message object within the message object. |
| ATTACH_BY_REFERENCE | These are regerences to a file by name. Since the PHP code runs on the server, these types are unsupp |
| ATTACH_BY_REF_RESOLVE | |
| ATTACH_BY_REF_ONLY | |

All of these types use different ways to store the name and to store the attachement data. with `mapi_message_getattachmenttable()` the table with all the attachments can be read from the message. Every row in this table has a property PR_ATTACH_METHOD which defines with which method the attachment was stored (see the values above). The property to read from an attachment or the function to call depends on the method an attachment has:

| Method | Description |
|---|---|
| NO_ATTACHMENT | The attachment has just been created. |
| ATTACH_BY_VALUE | The PR_ATTACH_FILENAME and PR_ATTACH_LONG_FILENAME contains the name of the attachment. When the attachment is opened with mapi_message_openattach() the data can be read with `mapi_attach_openbin()` |
| ATTACH_EMBEDDED_MSG | The PR_DISPLAY_NAME contains the name of the attachment (This is mostly the PR_SUBJECT of the message that has been attached). With `mapi_attach_openobj()` the attachment can be read from the message. This function returns a message object. |
| ATTACH_BY_REFERENCE | These types are unsupported and should never be used. |
| ATTACH_BY_REF_RESOLVE | |
| ATTACH_BY_REF_ONLY | |

When created, all attachment objects have an initial PR_ATTACH_METHOD value of NO_ATTACHMENT. Client applications and service providers are only required to support the attachment method represented by the

ATTACH_BY_VALUE value. The other attachment methods are optional.

The message store does not enforce any consistency between the value of PR_ATTACH_METHOD and the values of the other attachment properties.

## 2.5 Messages

The reading of the contents from a message can be done in a generic way. All messages contain all types of bodies. The different body types are kept in sync automatically, and the version which is written last wins, and updates the other versions. The following body types are available:

There are 3 ways in which the body of a message can be stored:

| Type | Description |
|---|---|
| RTF | The content of the message is stored in the PR_RTF_COMPRESSED property. |
| Embedded HTML in RTF | The content of the message is stored in the PR_RTF_COMPRESSED property. This is a special type of RTF that has an embedded HTML content. This version has been depricated by normal HTML. |
| HTML text | The content of the message is stored in the PR_HTML property. is normal HTML formatted text. |
| Plain text | The content of the body is stored in the PR_BODY property, and contains the plain text. |

The PR_RTF_COMPRESSED property can hold RTF of HTML embedded text. If you use this property, you can convert it to HTML with the mapi_rtf2html function. Therefore, this property is only useful to detect if the original message was sent in TRF or HTML.

### 2.5.1 RTF

The RTF-text may be used when messages are stored within Outlook. RTF text can only be sent through the TNEF data in a winmail.dat attachment.

### 2.5.2 Embedded HTML in RTF

The messages that are sent in HTML will be stored as RTF in the PR_RTF_COMPRESSED property. The HTML is converted from plain HTML to Embedded HTML. With the function `mapi_rtf2html` this type of RTF can be converted back to HTML to be shown in a browser. The function will return false when the RTF is not of the correct type.

While this is still true, this method of reading the body is depricated. If you want the HTML version of the body, use the normal HTML property.

### 2.5.3 HTML text

The normal HTML property is stored in the PR_HTML property. This is the preferred property if you want to read the HTML body.

### 2.5.4 Plain text

The PR_BODY will contain the message body in Plain text.

## 2.6 Recipients

Recipients are used to specify the receiving entities for a specific message.

The structure of a recipient array in PHP is as follows. When recipients are set, you should use mapi_createoneoff() to create the right PR_ENTRYIDs:

**Recipient table**

```
// Create array of recipients usable by the PHP MAPI functions
$recipientarray = Array(Array(PR_ENTRYID => mapi_createoneoff("John Doe", "SMTP",
↪"john@example.com"
                              PR_DISPLAY_NAME => "John Doe",
                              PR_ADDRTYPE => "SMTP",
                              PR_EMAIL_ADDRESS => "john@example.com"
                              PR_RECIPIENT_TYPE => MAPI_TO),
                        Array(PR_ENTRYID => mapi_createoneoff("Robert Roe", "SMTP",
↪ "robert@example.com"
                              PR_DISPLAY_NAME => "Robert Doe",
                              PR_ADDRTYPE => "SMTP",
                              PR_EMAIL_ADDRESS => "[email protected]"
                              PR_RECIPIENT_TYPE => MAPI_CC),
                        Array(PR_ENTRYID => mapi_createoneoff("Richard Miles",
↪"SMTP", "richard@example.com"
                              PR_DISPLAY_NAME => "Richard Miles",
                              PR_ADDRTYPE => "SMTP",
                              PR_EMAIL_ADDRESS => "richard@example.com"
                              PR_RECIPIENT_TYPE => MAPI_BCC));
```

To set or get recipients, use the mapi_message_modifyrecipients and mapi_message_getrecipienttable methods.

CHAPTER 3

# Getting Started

The PHP-MAPI extension provides access to MS MAPI function from php. Although not all MAPI functions and interfaces are supported, most functions have a php counterpart in this extension. Using PHP-MAPI, users can create webbased e-mail and calendaring systems and interfaces with existing php projects, using the mapi functions like a normal mapi program.

## 3.1 General MAPI Functions

If you work with mapi there are a few operations which will be generally used, these functions are used to interact with MAPI Objects.

The userstore (mapimsgstore) contains the Calendar, Mail, Notes etc. - These items are stored in a mapifolder and a mapifolder contains a mapitable which has mapimessages.

A MAPI message contains a list of properties, for example the subject and two tables: recipienttable and attachmenttable.

```
+> userstore
|
+-+> Inbox (mapifolder)
   |
   +--+> mapitable
   |  |
   |  +--+> mapimessage
   |     |
   |     +--+> properties
   |     |
   |     +--+> attachments
   |     |
   |     +--+> recipients
   |
   +> Calendar (mapifolder)
   |
   +--+> mapitable
   |
   +--+> mapimessage
```

There are a few functions which apply to these different items

### 3.1.1 mapi_getprops

```
mapi_getprops ( $obj, $properties)
```

This function returns an associative array of the requested properties and their values. If the value is not available, the value is not added to the associative array. Returns an array on success, FALSE on failure.

$obj The object which you are requesting the properties of. $properties (optional). The properties that you want to read from the object.

## 3.2 Handling MAPI Items

## 3.3 Opening a mail

Opening a message in PHP-MAPI requires a few operations, which are needed to get the default store, open the inbox folder and reading the properties of a MAPI message.

```
# PHP-MAPI provides these files
include('/usr/share/php/mapi/mapi.util.php');
include('/usr/share/php/mapi/mapidefs.php');
include('/usr/share/php/mapi/mapicode.php');
include('/usr/share/php/mapi/mapitags.php');
include('/usr/share/php/mapi/mapiguid.php');

$session = mapi_logon_zarafa('user','password');

$msgstorestable = mapi_getmsgstorestable($session);
$msgstores = mapi_table_queryallrows($msgstorestable, array(PR_DEFAULT_STORE, PR_
↪ENTRYID));
foreach ($msgstores as $row) {
        if($row[PR_DEFAULT_STORE]) {
                $storeentryid = $row[PR_ENTRYID];
        }
}

$userstore = mapi_openmsgstore($session, $storeentryid);
$inbox = mapi_msgstore_getreceivefolder($userstore);
$table = mapi_folder_getcontentstable($inbox);
$items = mapi_table_queryallrows($table,array(PR_SUBJECT, PR_ENTRYID));

foreach($items as $item) {
        print($item[PR_SUBJECT] . "\n");
}
```

The code above opens a user session, then looks for the defaultstore and opens it. Now we can use the mapi_msgstore_getreceivefolder() to open the inbox folder. Now that we have obtained the mapifolder we need to obtain mapitable so that we can iterate over it, when using mapi_table_queryallrows() we always needs to define which MAPI properties we want to have for this example I'm only interested in the entryid and the subject.

The variable $items contains an array which can contains multiple associative arrays which have a array(Property - Value) mapping. A few of these properties are explained in the table below.

Example of MAPI Properties of an email message.

| Property | Explanation |
|---|---|
| PR_SUBJECT | Message subject |
| PR_PRIORITY | Priority of E-Mail |
| PR_IMPORTANCE | Importance of E-Mail |
| PR_BODY | Body of E-Mail |

## 3.4 Reading recipients

Recipients of an email are stored in a separate MAPITable of a MAPI Object. To list the recipients of a message, we call **mapi_message_getrecipienttable** on a MAPI Object and **mapi_table_queryallrows** to fetch all the entries in the table.

```
$message = mapi_msgstore_openentry($store, $item[PR_ENTRYID]);
$table = mapi_message_getrecipienttable($message);
$recips = mapi_table_queryallrows($table, array(PR_DISPLAY_NAME, PR_ADDRTYPE, PR_
↪EMAIL_ADDRESS));
foreach($recips as $recip) {
        print $recip[PR_ADDRTYPE] . "\n";
        print $recip[PR_DISPLAY_NAME] . "\n";
        print $recip[PR_EMAIL_ADDRESS] . "\n";
}
```

## 3.5 Reading attachments

Attachments are also stored in a separate table of a MAPI Object. The example is similar to reading recipients, it also saves the attachments to disk.

```
$message = mapi_msgstore_openentry($store, $entry[PR_ENTRYID]);
$table = mapi_message_getattachmenttable($message);
$attachments = mapi_table_queryallrows($table, array(PR_ATTACH_NUM, PR_ATTACH_SIZE,
↪ PR_ATTACH_LONG_FILENAME));
foreach($attachments as $attach) {
        $filename = $attach[PR_ATTACH_LONG_FILENAME];
        print 'Attach Name ' . $filename  .' Attach number: ' . $attach[PR_ATTACH_
↪NUM] . ' | Size ' .  $attach[PR_ATTACH_SIZE] . "\n";

        $attach = mapi_message_openattach ($message, $attach[PR_ATTACH_NUM]);
        $stream = mapi_openpropertytostream($attach, PR_ATTACH_DATA_BIN);
        $stat = mapi_stream_stat($stream);

        $fhandle = fopen("$filename", 'w');
        $buffer = null;

        for($i = 0; $i < $stat["cb"]; $i += 1024) {
                // Write stream
                $buffer = mapi_stream_read($stream, 1024);
                fwrite($fhandle, $buffer, strlen($buffer));
        }
        fclose($fhandle);
}
```

# Function Reference

## 4.1 General functions

General functions - functions that are not directly MAPI related, but are required for many standard operations.

### 4.1.1 php_mapi_error

**mapi_is_error** (*$hresult*)

Checks if the high bit of the passed error is set.

**Parameters**

- **$hresult** (*long*) – The error value to check

**Returns** TRUE if the specified value has the high bit set (0x80000000), indicating a error

**Return type** bool

```
if (mapi_is_error(mapi_last_hresult())) {
        echo "error!\n";
}
```

### 4.1.2 mapi_last_hresult

**mapi_last_hresult** ()

Returns 0 if last call succeeded. Otherwise, this value contains the MAPI error code. MAPI error codes are defined in include/php/mapicode.php.

**Returns** The last occurred HRESULT from a MAPI function

**Return type** long

```
$inbox = mapi_msgstore_getreceivefolder($store);
if($inbox == false) {
        if (mapi_last_hresult() == MAPI_E_NO_ACCESS) {
                print "Access was denied to the inbox!\n";
        }
}
```

### 4.1.3 mapi_make_scode

**mapi_make_scode** (*$severity*, *$code*)

This function returns the SCODE value of the specified error. Refer to MSDN for further information on SCODEs.

**Parameters**

- **$severity** (*long*) – The severity value, 0 or 1
- **$code** (*long*) – The actual error value

**Returns** TRUE if error found, FALSE if not

**Return type** long

```
$error_not_found = mapi_make_scode(1, 0x010f);
```

### 4.1.4 mapi_prop_id

**mapi_prop_id** (*$proptag*)
Returns the ID part of the property tag (the high 16 bits).

**Parameters**

- **$proptag** (*long*) – The propertytag

**Returns** ID part of property tag

**Return type** long

```
// Get the ID
$id = mapi_prop_id(PR_SUBJECT); // $id is now 0x0037
```

See also `mapi_prop_tag()` `mapi_prop_type()`

### 4.1.5 mapi_prop_type

**mapi_prop_type** (*$proptag*)
Returns the type part of the property tag (the low 16 bits).

**Parameters**

- **$proptag** (*long*) – The propertytag

**Returns** Type part of property tag

**Return type** long

```
// Get the type
$id = mapi_prop_type(PR_SUBJECT); // $id is now equal to PT_STRING8
```

See also `mapi_prop_tag()` `mapi_prop_id()`

### 4.1.6 mapi_prop_tag

**mapi_prop_tag** (*$proptype*, *$propid*)
A propertytag is a unique identifier of a property. The property ID and property type are stores into the propertytag. Returns the property tag.

**Parameters**

- **$proptype** (*long*) – The type of the propertytag that must be made. Should be one of the PT_* constants.
- **$propid** (*long*) – The unique ID for the propertytag.

**Returns** The property tag

**Return type** long

```
// Make a PR_SUBJECT
define('PR_SUBJECT', mapi_prop_tag(PT_STRING8,   0x0037));
```

See also `mapi_prop_type()` `mapi_prop_id()`

**Note:** Because PHP can't handle unsigned longs some of the propertytags will look like this: -2129461248.

### 4.1.7 mapi_getidsfromnames

**mapi_getidsfromnames** (*$store*, *$names*[, *$guids* ])

Returns an array of mapped properties for the names specified. If $guids is not specified, then the default GUID {00062002-0000-0000-C000-000000000046} is used for all names. The returned properties are all of type PT_UNSPECIFIED so they have to be converted to the correct property types before being passed to mapi_setprops or mapi_getprops. Returns an array on success, FALSE on failure.

**Parameters**

- **$store** (*mapimsgstore*) – The store that the ID's are being requested for
- **$names** (*array*) – The names being requested.
- **$guids** (*array*) – The guids of the names being requested

**Returns** Mapped properties for specified names

**Return type** array

```
// Get the ID's of the named properties for 'Start' and 'End' of calendar items
$ids = mapi_getidsfromnames($store, Array(0x820d, 0x820e), Array("{00062002-
↪0000-0000-C000-000000000046}","{00062002-0000-0000-C000-000000000046}"));

// Create actual proptags by adding the type of the properties
$ids[0] = mapi_prop_tag(PT_SYSTIME, mapi_prop_id($ids[0]);
$ids[1] = mapi_prop_tag(PT_SYSTIME, mapi_prop_id($ids[1]);

$startend = mapi_message_getprops($message, $ids);
```

See also `mapi_getprops()` `mapi_setprops()` `mapi_prop_tag()` `mapi_prop_id()`

### 4.1.8 mapi_getnamesfromids

**mapi_getnamesfromids** (*$store*, *$proptags*)

Returns an array of mapped properties for the IDs specified. Returns an array on success, FALSE on failure. The array contains a name or id and a guid for each found property.

**Parameters**

- **$store** (*mapimsgstore*) – The store that the IDs are being requested for
- **$proptags** (*array*) – A list of named proptags for which names should be looked up. The prop_type is not important, and is mostly PT_NULL.

**Returns** Mapped properties for specified IDs

**Return type** array

```
// Get the ID's of the named properties for 'Start' and 'End' of calendar items
$ids = mapi_getidsfromnames($store, Array(0x820d, 0x820e));
// Re-read the names from the ids we just got.
$names = mapi_getnamesfromids($store, $ids);
print_r($names);
```

```
// or create a new array list of ids
$names = mapi_getnamesfromids($store, array(mapi_prop_tag(PT_NULL, 0x8503),␣
↪mapi_prop_tag(PT_NULL, 0x8001)));
print_r($names);
```

Output:

```
Array
(
[-2146631680] => Array
    (
        [guid] => guid...
        [id] => 33293
    )

[-2146566144] => Array
    (
        [guid] => guid...
        [id] => 33294
    )

[-2146959360] => Array
    (
        [guid] => guid...
        [id] => 33288
    )

[-2146238464] => Array
    (
        [guid] => guid...
        [id] => 33299
    )

[-2144010240] => Array
    (
        [guid] => guid...
        [id] => 33333
    )

)
Array
(
[-2063400959] => Array
    (
        [guid] => guid...
        [name] => ...
    )

[-2147418111] => Array
    (
        [guid] => guid...
        [id] => 33281
    )

)
```

See also `mapi_getprops()` `mapi_setprops()` `mapi_prop_tag()` `mapi_prop_id()`

### 4.1.9 mapi_getprops

**mapi_getprops** (*$obj*[, *$properties* ])

This function returns an associative array of the requested properties and their values. If the value is not available, the value is not added to the associative array. Returns an array on success, FALSE on failure.

**Parameters**

- **$obj** (*mapiobject*) – The object which you are requesting the properties of
- **$properties** (*array*) – The properties that you want to read from the object

**Returns** Requested properties and corresponding values

**Return type** array

```
// Get the properties of the message store
$storeprops = mapi_getprops($userstore, array(PR_DISPLAY_NAME));

print "Store: " . $storeprops[PR_DISPLAY_NAME];
```

### 4.1.10 mapi_setprops

**mapi_setprops** (*$obj*, *$properties*)

This function will set the properties for a MAPI object. It will work for all objects that inherit from IMAPIProp.

**Parameters**

- **$obj** (*mapiobject*) – The object which you are about to set properties of
- **$properties** (*array*) – Array of properties you want to set

**Returns** TRUE on success, FALSE on failure

**Return type** bool

```
// Create a message and set some props.
$message = mapi_message_createmessage($inbox);

// Create an array with some props
$props = Array(PR_SUBJECT => "Testsubject",
PR_BODY => "Hello world!");

// Set props
mapi_setprops($message, $props);

// Save changes
mapi_savechanges($message);
```

See also `mapi_getprops()` `mapi_savechanges()`

### 4.1.11 mapi_copyto

**mapi_copyto** (*$srcobj*, *$excludeiids*, *$excludeprops*, *$dstobj*, *$flags*)

This function will copy the properties for a MAPI object. It will work for all objects that inherit from IMAPIProp.

**Parameters**

- **$srcobj** (*mapiobject*) – The object you want to copy the properties of
- **$excludeiids** (*array*) – Array of interface identifiers (IIDs) indicating interfaces that should not be used

- **$excludeprops** (*array*) – Array of property tags that should be excluded
- **$dstobj** (*mapiobject*) – The object you want to copy the properties to
- **$flags** (*long*) – The MAPI flags to use copying. Most useful flags are MAPI_MOVE and MAPI_NOREPLACE

**Returns** TRUE on success, FALSE on failure

**Return type** bool

```
// Create a message
$newmessage = mapi_message_createmessage($inbox);

// Copy message
mapi_copyto($message, array(), array(), $newmessage);

// Save changes
mapi_savechanges($newmessage);
```

See also `mapi_savechanges()`

### 4.1.12 mapi_savechanges

**mapi_savechanges**(*$object*)

Save changes on a MAPI Object. This is mostly used on a message or attachment to push the property changes to the server.

Changes to a MAPI resource are not actually sent to the server until a mapi_savechanges is called. When creating new messages, the message will be deleted if a mapi_savechanges call is not sent to the newly created message object.

**Parameters**

- **$object** (*mapiobj*) – The object you wish to save. Can be a message, attachment, folder or store

**Returns** TRUE on success, FALSE on failure

**Return type** bool

```
// Set the subject
mapi_setprops($message, Array(PR_SUBJECT) => "New subject"));
mapi_savechanges($message);
```

See also `mapi_setprops()`

### 4.1.13 mapi_deleteprops

**mapi_deleteprops**(*$obj*[, *$properties* ])

Delete properties from a MAPI object.

This function deletes the properties given from a object. This works for all object that are derived from IMAPIProp.

Returns true when the deletion has been successful or false when something got wrong.

**Parameters**

- **$obj** (*mapiobject*) – The object which you are deleting the properties of
- **$properties** (*array*) – The properties that you want to delete from the object

**Returns** TRUE on success, FALSE on failure

**Return type** bool

```
// Deleting a property of a message
$message = mapi_msgstore_openentry($userstore, $message_entryid);

// remove subject from a message
mapi_deleteprops($message, array(PR_SUBJECT));
mapi_savechanges($message);
```

### 4.1.14 mapi_openproperty

**mapi_openproperty** (*$mapipropObj*, *$propertytag*, *$interface*, *$interfaceflags*, *$flags*)

Opens a property from an object.

Some properties are not shown when a mapi_getprops is executed, and return MAPI_E_NOT_ENOUGH_MEMORY. With mapi_openproperty these properties can be read, and are returned as a string.

**Parameters**

- **$mapipropObj** (*mapipropObj*) – The object from which the property should be opened, this object can be one of the following (Mapimessage, Mapifolder, Mapiattachment, Messagestore)
- **$propertytag** (*long*) – The specific tag that should be opened
- **$interface** (*string*) – The interface that should be used to open the property. This is normally any of the IID_I* interface definitions that are available.
- **$interfaceflags** (*long*) – Flags that should be passed to the interface which are specific to that interface. For example, you could specify STGM_TRANSACTED when opening an IID_IStream interface. Normally just 0.
- **$flags** (*long*) – The MAPI flags to use when opening this property. Most useful flags are MAPI_CREATE and MAPI_MODIFY, which must both be specified (ORed together) when creating a new property using mapi_openproperty

**Returns** string on success, FALSE on failure

**Return type** string

```
// Open a property from a folder
$viewliststream = mapi_openproperty($inbox, PR_FOLDER_VIEWLIST, IID_IStream, 0,
↪ 0);
```

See also `mapi_getprops()`

### 4.1.15 mapi_createoneoff

**mapi_createoneoff** (*$displayname*, *$type*, *$emailaddress*[, *$flags* ])

This function creates a One-Off entry identifier. These are used in recipient tables (for PR_ENTRYID) and in message properties. These are also used in flatentrylists. Returned is a binary string with the entry identifier.

**Parameters**

- **$displayname** (*string*) – The display name of the recipient to create a One-Off entryid for
- **$type** (*string*) – The type of the recipient's address (mostly "SMTP")
- **$emailaddress** (*string*) – The e-mail address of the recipient
- **$flags** (*long*) – Bitmask of flags that affects the one-off recipient

**Returns** Binary string with the entry identifier

**Return type** string

| Flag | Description |
|---|---|
| MAPI_UNICODE | The display name, address type, and address are in Unicode format. |
| MAPI_SEND_NO_RICH_INFO | The recipient cannot handle formatted message content. If MAPI_SEND_NO_RICH_INFO is set, MAPI sets the recipient's PR_SEND_RICH_INFO property to FALSE. If MAPI_SEND_NO_RICH_INFO is not set, MAPI sets this property to TRUE unless the recipient's messaging address pointed to by lpszAddress is interpreted to be an Internet address. In this case, MAPI sets PR_SEND_RICH_INFO to FALSE. |

```
// Creating a one-off entry identifier
$oneoffentryid = mapi_createoneoff("John Doe", "SMTP", "john@example.com");
```

See also `mapi_message_modifyrecipients()`

### 4.1.16 mapi_parseoneoff

**mapi_parseoneoff**(*$oneoff*)

Parses a one-off entryid to an array with meaningful values.

This function takes a one-off entryid and parses it. Given back is an array with indexes 'name', 'type' and 'value'.

**Parameters**

- **$oneoff** (*string*) – The input string with the one-off entryid to parse.

**Returns** Array with indexes 'name', 'type' and 'value'

**Return type** array

```
// Listing Kopano users
$props = mapi_getprops($message, Array(PR_ENTRYID));
$result = mapi_parseoneoff($props[PR_ENTRYID]);

echo "Display name: ".$result['name'];
echo "Type: ".$result['type'];
echo "E-mail address: ".$result['address'];

array mapi_parseoneoff(string $oneoff)
```

See also `mapi_createoneoff()`

### 4.1.17 mapi_sink_create

**mapi_sink_create**()

This function creates a mapi sink used for MAPI notifications.

**Returns** MAPISink

**Return type** MAPIObject

```
$sink = mapi_sink_create();
```

## 4.2 Logon functions

Logon functions - This section describes the various logon functions.

### 4.2.1 mapi_logon

**mapi_logon** (*$profile*, *$password*)

Logon to a MAPI profile

mapi_logon uses the MAPI calls MAPILogonEx().

**Parameters**

- **$profile** (*string*) – The profile name to log on to. Profiles must be pre-created to be used - If $profile is empty, the default profile is used
- **$password** (*string*) – The password to open the profile.

**Returns** mapisession object on success, FALSE on failure

**Return type** mapisession

---

**Note:** This is not the password to open a specific store or folder, just the password on the actual profile

---

See also `mapi_logon_zarafa()`

```
// Log on to the default profile with no password
$session = mapi_logon('','');
```

### 4.2.2 mapi_logon_zarafa

**mapi_logon_zarafa** (*$username*, *$password*[, *$server*, *$sslcert*, *$sslpass*, *$notifications*, *$client_version*, *$misc_version* ])

Logon to a Kopano server

mapi_logon_zarafa logs a user on to the server. A MAPI Session object is returned, which is required for further calls. This is the preferred method to logon to Kopano.

**Parameters**

- **$username** (*string*) – The name of the user to log with
- **$password** (*string*) – The password of the user
- **$server** (*string*) – The location of the server. If not given, this will default to 'http://localhost:236/kopano'. To connect over the unix socket (fastest method), use 'file:///var/run/kopano/server.sock' here.
- **$sslcert** (*string*) – The ssl certificate file
- **$sslpass** (*string*) – The ssl certificate password
- **$notifications** (*string*) – Allow notifications to be received from the server (possible values S_OK or S_FALSE)
- **$client_version** (*string*) – the send client version for statistics
- **$misc_version** (*string*) – misc. version

**Returns** mapisession object on success, FALSE on failure

**Return type** mapisession

```
// Log on to Kopano server using the unix socket
$session = mapi_logon_zarafa('user','password','file:///var/run/kopano/server.
↪sock');
if ($session == false) {
        print "logon failed with error ".mapi_last_hresult()."\n";
}
```

See also `mapi_getmsgstorestable()`

### 4.2.3 mapi_openmsgstore

**mapi_openmsgstore** (*$session*, *$entryID*)

Opens a messagestore

The session is used to open a messagestore with the entryID given. Returns a mapimsgstore object on success, FALSE on failure.

**Parameters**

- **$session** (*mapisession*) – The session that is opened before
- **$entryID** (*string*) – The unique identifier that is used to identify the messagestore

**Returns** mapimsgstore object on success, FALSE on failure

**Return type** mapimsgstore

```
// Opening the stores of a Kopano session
// $serverlocation is optional, default http://localhost:236/zarafa
$session = mapi_logon_zarafa($username, $password, $serverlocation);
$stores = mapi_getmsgstorestable($session);
$storeslist = mapi_table_queryallrows($stores);
$userstore = mapi_openmsgstore($session, $storeslist[0][PR_ENTRYID]);
$publicstore = mapi_openmsgstore($session, $storeslist[1][PR_ENTRYID]);
```

**Note:** When the entryid is a hexadecimal string, the function hex2bin must be used to convert the string to binary. Also the function bin2hex can be used to convert from binary to hexadecimal.

See also `mapi_getmsgstoretable()`

### 4.2.4 mapi_openentry

**mapi_openentry** (*$session*, *$entryID*, *$flags*)

Opens an entry from a messagestore or addressbook.

This function works as mapi_msgstore_openentry but automatically detects from which store the message should be opened. It can also be used for EntryIDs in the addressbook and one-off EntryIDs. Because this function must open a store to get the item in that store, it is more efficient to use the mapi_msgstore_openentry function.

**Parameters**

- **$session** (*mapisession*) – The session that is opened before.
- **$entryID** (*string*) – A unique identifier that is used to identify an item.
- **$flags** (*long*) – Usually 0, can be: MAPI_MODIFY (?).

**Returns** resource object on success, FALSE on failure

**Return type** resource

See also `mapi_msgstore_openentry()`

### 4.2.5 mapi_getmsgstorestable

**mapi_getmsgstorestable** (*$session*)

Gets a table with the messagestores within the session.

Gets a table with the messagestores defined in the profile. The profile should be opened first with the function mapi_logon_zarafa. The table can be read with a mapi_table_* functions. Internally, this function calls the MAPI function GetMsgStoreTable().

**Parameters**

- **`$session`** (*mapisession*) – The opened session

**Returns** mapitable object on success, FALSE on failure

**Return type** mapitable

```
// Getting the table with the stores
$session = mapi_logon_zarafa($username, $password, $serverlocation);
$table = mapi_getmsgstorestable($session);
```

See also `mapi_table_queryallrows()` `mapi_table_queryrows()`

### 4.2.6 mapi_openprofilesection

**`mapi_openprofilesection`** (*$session*, *pbGlobalProfileSectionGuid*)

Open the profile section of given guid, and returns the php-usable IMAPIProp object.

This is used to setup the contact provider for the addressbook. The addressbook can show more folders via the contact provider by adding a store, entryid and display name to the PR_ZC_CONTACT_STORE_ENTRYIDS, PR_ZC_CONTACT_FOLDER_ENTRYIDS and PR_ZC_CONTACT_FOLDER_NAMES.

**Parameters**

- **`$session`** (*mapisession*) – The opened session
- **`$pbGlobalProfileSectionGuid`** (*string*) – An hardcoded guid in mapi/mapidefs.php

**Returns** mapiobject object on which extra folders can be set using mapi_setprops

**Return type** mapiobject

```
$profsect = mapi_openprofilesection($session, pbGlobalProfileSectionGuid);
mapi_setprops($profset, array(
        PR_ZC_CONTACT_STORE_ENTRYIDS => Array($contact_store_entryid),
        PR_ZC_CONTACT_FOLDER_ENTRYIDS => Array($contact_entryid),
        PR_ZC_CONTACT_FOLDER_NAMES => Array("contacts"),
));
```

## 4.3 Store functions

Store functions - This section describes functions which are performed on MAPI Store objects.

### 4.3.1 mapi_msgstore_getreceivefolder

**`mapi_msgstore_getreceivefolder`** (*$store*)

Open the inbox for the specified store.

This function requests the EntryID for the inbox of the given store and opens the folder.

**Parameters**

- **`$store`** (*mapimsgstore*) – The store of which the inbox is required

**Returns** mapifolder object on success, FALSE on failure

**Return type** mapifolder

```
// Opening the inbox of a store

$store = mapi_logon_zarafa($username, $password, $serverlocation);
$stores = mapi_getmsgstorestable($session);
$storeslist = mapi_table_queryallrows($stores);
$userstore = mapi_openmsgstore($session, $storeslist[0][PR_ENTRYID]);
$inbox = mapi_msgstore_getreceivefolder($userstore);
```

See also `mapi_logon_zarafa()`

### 4.3.2 mapi_msgstore_openentry

**mapi_msgstore_openentry**(*$messagestore*, *$entryID*[, *$flags* ])

Opens an entry from the messagestore.

This function opens an entry from the messagestore. The function will return a mapifolder or a mapimessage, depending on the entryID.

**Parameters**

- **$messagestore** (*mapimsgstore*) – The current messagestore used
- **$entryID** (*string*) – Entry ID to be opened
- **$flags** (*long*) – Bitmask of flags that controls how information is returned in the table

**Returns** mapifolder object on success, FALSE on failure

**Return type** mapifolder

| Flag | Description |
|---|---|
| MAPI_BEST_ACCESS | Requests that the object be opened with the maximum network permissions allowed for the user and the maximum client application access. For example, if the client has read/write access, the object should be opened with read/write access; if the client has read-only access, the object should be opened with read-only access. |
| MAPI_MODIFY | Requests read/write access. By default, objects are opened with read-only access, and clients should not work on the assumption that read/write access has been granted. |
| SHOW_SOFT_DELETES | Open a Soft Deleted Message or Folder. |

```
// Opening an entry from a messagestore

// Open a folder from a store
$folder = mapi_msgstore_openentry($store, $folder_entryid);

// Open a message from a store
$message = mapi_msgstore_openentry($store, $message_entryid);
```

### 4.3.3 mapi_msgstore_createentryid

**mapi_msgstore_createentryid**(*$store*, *$username*)

Returns an entryid string to an arbitrary user store which exists on the Kopano server.

Creates an EntryID to use with mapi_openmsgstore() function.

**Parameters**

- **$store** (*resource*) – The store of the currently logged in user, which was opened with mapi_openmsgstore()
- **$username** (*string*) – The username of the store that needs to be opened.

**Returns** valid entry id on success, FALSE on failure

**Return type** string

```
// Opening a Kopano message store
$user2_entryid = mapi_msgstore_createentryid($userstore, "user2");
$user2_store = mapi_openmsgstore($session, $user2_entryid);
```

See also `mapi_openmsgstore()`

### 4.3.4 mapi_msgstore_openmultistoretable

**mapi_msgstore_openmultistoretable** (*$store*, *$entryids*[, *$flags* ])

Open a special table that may contain items from different stores. The items you wish the table to contain is passed to this function.

This function opens a special table which contains items the user sets. The return value can be treated a normal table, using the mapi_table_* functions to handle it.

**Parameters**

- **$store** (*mapimsgstore*) – The store of which the inbox is required.
- **$entryids** (*array*) – An array of message entry id's. This array may contain entry id's from any store.
- **$flags** (*long*) – Flags which is currently unsused, and is 0 by default.

**Returns** mapitable object on success, FALSE on failure

**Return type** mapitable

```
// Opening the special multistore table
// Open the inbox for a user
$store = mapi_logon_zarafa($username, $password, $serverlocation);
$stores = mapi_getmsgstorestable($session);
$storeslist = mapi_table_queryallrows($stores);
$userstore = mapi_openmsgstore($session, $storeslist[0][PR_ENTRYID]);
// array() should be replaced with an array of entry id's of messages
$mst = mapi_msgstore_openmultistoretable($userstore, array());
```

See also `mapi_table_queryrows()`

### 4.3.5 mapi_msgstore_entryidfromsourcekey

**mapi_msgstore_entryidfromsourcekey** (*$store*, *$sourcekey*)

Convert an PR_SOURCE_KEY to an PR_ENTRYID, for example to be able to open a folder if you only have a PR_SOURCE_KEY.

**Parameters**

- **$store** (*mapimsgstore*) – The store where $sourcekey is from.
- **$sourcekey** (*string*) – The folder sourcekey.

**Returns** string entryid on success, FALSE on failure

```
$entryid = mapi_msgstore_entryidfromsourcekey($store, $props[PR_SOURCE_KEY]);
$mapifolder = mapi_msgstore_openentry($store, $entryid);
```

### 4.3.6 mapi_msgstore_advise

**mapi_msgstore_advise** (*$store*, *$entryid*, *$flags*, *$sink*)

Setup function for receiving notifications when something changes in a store, it's possible to listen to the whole store for events or a specific folder. The $flag argument can filter on the types of notifications such as: fnevNewMail, fnevObjectCreated, fnevObjectDeleted, fnevObjectModified, fnevObjectCopied, fnevSearchComplete.

**Parameters**

- **$store** (*mapimsgstore*) – The store to watch for notifications.
- **$entryid** (*string|null*) – The folder entryid or null for the whole store
- **$flags** (*number*) – The events you want to listen for.
- **$sink** (*mapisink*) – the MAPISink cretaed with mapi_sink_create();

**Returns** mapiobject connection

```
$mapifolder = mapi_msgstore_openentry($store, $entryid);
$sink = mapi_sink_create();
$connection = mapi_msgstore_advise($store, null, fnevNewMail, $sink);
$notifs = mapi_sink_timedwait($sink, 10000);
mapi_msgstore_unadvise($store, $connection);
```

### 4.3.7 mapi_msgstore_unadvise

**mapi_msgstore_unadvise** (*$store*, *$connection*)

Cleanup the created $connection by mapi_msgstore_advise.

**Parameters**

- **$store** (*mapimsgstore*) – The store to watch for notifications.
- **$connection** (*mapiobj*) – The connection created by mapi_msgstore_advise

```
mapi_msgstore_unadvise($store, $connection);
```

## 4.4 Folder functions

Folder functions - This section describes functions which are performed on MAPI Folder objects.

### 4.4.1 mapi_folder_gethierarchytable

**mapi_folder_gethierarchytable** (*$folder*[, *$flags* ])

Gets the hierarchytable

Opens the hierarchytable of the folder. The hierarchytable contains the subfolders of the folder.

**Parameters**

- **$folder** (*mapifolder*) – The folder from which the hierarchytable should be opened.
- **$flags** (*long*) – Bitmask of flags that controls how information is returned in the table.

**Returns** mapitable object on success, FALSE on failure

**Return type** mapitable

| Flag | Description |
|---|---|
| SHOW_SOFT_DELETES | Get a list of Soft Deleted Folders of the subfolder pointed to by $entryid. |

```
// Get the hierarchytable when $inbox is a valid MAPI Folder
$table = mapi_folder_gethierarchytable($inbox);
```

See also `mapi_folder_getcontentstable()`

### 4.4.2 mapi_folder_getcontentstable

**mapi_folder_getcontentstable** (*$folder*[, *$flags* ])

Gets the contentstable

Opens the contentstable of the folder. The contentstable contains the messages of the folder.

**Parameters**

- **$folder** (*mapifolder*) – The folder from which the contentstable should be opened
- **$flags** (*long*) – Bitmask of flags that controls how information is returned in the table

**Returns** mapitable object on success, FALSE on failure

**Return type** mapitable

| Flag | Description |
|---|---|
| MAPI_ASSOCIATED | Get a list of associated messages of the subfolder pointed to by $folder. |
| SHOW_SOFT_DELETES | Get a list of Soft Deleted messages of the subfolder pointed to by $folder. |

```
// Get the messages of $inbox
$table = mapi_folder_getcontentstable($inbox);
```

See also `mapi_folder_gethierarchytable()`

### 4.4.3 mapi_folder_createmessage

**mapi_folder_createmessage** (*$folder*[, *$flags* ])

Creates a new message in the folder

This function makes a new message in the folder. It returns the newly created message and with mapi_setprops properties can be stored into the message. The message is not visible to other users until the mapi_savechanges function is called.

**Parameters**

- **$folder** (*mapifolder*) – The folder from in which the message should be created.
- **$flags** (*long*) – A bitmask of flags that control the creation of a folder.

**Returns** mapimessage object on success, FALSE on failure

**Return type** mapimessage

| Flag | Description |
|---|---|
| MAPI_DEFERRED_ERRORS | Allow CreateMessage to return successfully, possibly before the new folder is fully accessible to the calling client. If the new message is not accessible, making a subsequent call to it can result in an error. |
| MAPI_ASSOCIATED | The message to be created should be included in the associated contents table rather than the standard contents table. Associated messages typically contain invisible data, such as view descriptors. |

```
// Create a new message into the inbox
$viewlist = mapi_folder_createmessage($inbox);
```

See also `mapi_setprops()` `mapi_savechanges()` `mapi_message_submitmessage()`

### 4.4.4 mapi_folder_deletemessages

**mapi_folder_deletemessages** (*$folder*, *$entryids*[, *$flags* ])

Deletes messages from a folder

This function removes a message with the given entryid's from the specified folder. Note that normally in Outlook, messages are simply moved to the 'Deleted items' folder when a 'delete' is requested, and messages are only really deleted when they are removed from the 'Deleted items' folder.

**Parameters**

- **$folder** (*mapifolder*) – The folder containing the messages to be deleted
- **$entryids** (*array*) – An array of binary entryID's to be removed from the folder
- **$flags** (*long*) – Bitmask of flags that controls how information is returned in the table.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

| Flag | Description |
|---|---|
| DELETE_HARD_DELETE | Hard Delete the messages given by $entryid. |

```
// Create a new message into the inbox
$msg = mapi_folder_createmessage($inbox);
mapi_savechanges($msg);

// Get the EntryID
$props = mapi_message_getprops($msg);

// Remove it
mapi_folder_deletemessages(array($props[PR_ENTRYID]));
```

See also `mapi_folder_createmessage()`

### 4.4.5 mapi_folder_copymessages

**mapi_folder_copymessages** (*$srcFolder*, *$messageArray*, *$destFolder*[, *$Flags* ])

Copies one or more messages from one folder to another.

This function moves or copies the one or more messages given with the messageArray.

The messagearray must be in following structure:

```
$messageArray = array (
        [0] => {entryid}
        [1] => {entryid}
        ...
);
```

The optional flag can be MESSAGE_MOVE when the messages must be moved from the sourcefolder to the destination folder, when no flag is provided the messages will be copied.

**Parameters**

- **$srcfolder** (*mapifolder*) – The folder which you want to copy the messages from.
- **$messageArray** (*array*) – An array with the entryid's from the message you want to copy.

- **`$destfolder`** (*mapifolder*) – The folder which you want to copy the messages to.
- **`$flags`** (*long*) – Some flags you can set. (Only MESSAGE_MOVE will work)

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Moving messages to the wastebasket
$storeprops = mapi_msgstore_getprops($userstore);
$wastebasket = mapi_msgstore_openentry($userstore, $storeprops[PR_IPM_
↪WASTEBASKET_ENTRYID]);

$inbox = mapi_msgstore_getreceivefolder($userstore);
$contentstable = mapi_folder_getcontentstable($inbox);
$contents = mapi_table_queryallrows($contentstable);

foreach($contents as $row) {
        $msg[] = $row[PR_ENTRYID];
}

mapi_folder_copymessages($inbox, $msg, $wastebasket, MESSAGE_MOVE);
```

### 4.4.6 mapi_folder_emptyfolder

**`mapi_folder_emptyfolder`**(*$folder*[, *$flags*])

Deletes all the messages in a folder.

This function deletes all the messages and subfolders in the folder. But without deleting itself.

**Parameters**

- **`$folder`** (*mapifolder*) – The folder which you want to copy the messages from
- **`$flags`** (*long*) – Bitmask of flags that controls how the folder is emptied

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

| Flag | Description |
|---|---|
| DEL_ASSOCIATED | Deletes all subfolders, including subfolders containing messages with associated content. The DEL_ASSOCIATED flag only has meaning for the top-level folder the call acts on. |
| DELETE_HARD_DELETE | Hard Delete the messages and/or folders |

```
// Emptying a folder
$storeprops = mapi_msgstore_getprops($userstore);
$wastebasket = mapi_msgstore_openentry($userstore, $storeprops[PR_IPM_
↪WASTEBASKET_ENTRYID]);

mapi_folder_emptyfolder($wastebasket);
// wastebasket is empty now
```

### 4.4.7 mapi_folder_copyfolder

**`mapi_folder_copyfolder`**(*$srcfolder*, *$entryid*, *$destfolder*, *$new_foldername*[, *$flags*])

Copies a folder from one parent folder to another.

This function moves or copies the folder given with entryid. The function will return true when the copy/move was successful.

**Parameters**

- **$srcfolder** (*mapifolder*) – The parent folder which you want to copy the messages from
- **$entryid** (*string*) – An array with the entryid's from the message you want to copy
- **$destfolder** (*mapifolder*) – The folder which you want to copy the messages to
- **$new_foldername** (*string*) – The new foldername of the copy of the source folder in the destination folder
- **$flags** (*long*) – A bitmask of flags that control the setting of a message's read flag

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

| Flag | Description |
|---|---|
| COPY_SUBFOLDERS | All of the subfolders in the folder to be copied should also be copied. When COPY_SUBFOLDERS is not set for a copy operation, only the folder identified by $entryid is copied. With a move operation, the COPY_SUBFOLDERS behavior is the default regardless of whether the flag is set. |
| FOLDER_MOVE | The folder is to be moved rather than copied. If FOLDER_MOVE is not set, the folder is copied. |

```
// Move a complete folder, with subtree, to a new destination

$res = mapi_folder_copyfolder($inbox, $tocopy_folder_entryid, $dest_folder,
↪"New folder name", COPY_SUBFOLDERS | FOLDER_MOVE);
if ($res == false) {
        print "mapi_folder_copyfolder() failed.";
}
```

See also `mapi_msgstore_openentry()`

### 4.4.8 mapi_folder_deletefolder

**mapi_folder_deletefolder** (*$srcfolder*, *$entryid*[, *$flags* ])

Deletes a folder.

This function deletes the folder given with the entryid.

**Parameters**

- **$srcfolder** (*mapifolder*) – The folder which you want to delete the folder from.
- **$entryid** (*string*) – The entryid of the folder you want to delete.
- **$flags** (*long*) – A bitmask of flags that control the setting of a message's read flag.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

| Flag | Description |
|---|---|
| DEL_FOLDERS | All subfolders of the subfolder pointed to by $entryid should be soft deleted. |
| DEL_MESSAGES | All messages in the subfolder pointed to by $entryid should be soft deleted. |
| DELETE_HARD_DELETE | All messages in the subfolder pointed to by $entryid should be hard deleted. |

```
// Deleting a folder
$result = mapi_folder_deletefolder($parent, $folderid);
```

See also `mapi_msgstore_openentry()`

### 4.4.9 mapi_folder_createfolder

**mapi_folder_createfolder** (*$parentfolder*, *$name*, *$description*[, *$flags* ][, *$foldertype* ])

Creates a folder.

This function creates a folder within the given folder; with the given name and description.

**Parameters**

- **$parentfolder** (*mapifolder*) – The folder in which you want to create a new folder.
- **$name** (*string*) – The name of the folder to create.
- **$description** (*string*) – A description of the folder.
- **$flags** (*long*) – A bitmask of flags that control the creation of a folder.
- **$foldertype** (*long*) – Specify the type of the folder to create.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

| Flag | Description |
|---|---|
| MAPI_DEFERRED_ERRORS | Allow CreateFolder to return successfully, possibly before the new folder is fully accessible to the calling client. If the new folder is not accessible, making a subsequent call to it can result in an error. |
| MAPI_UNICODE | The passed-in strings are in Unicode format. If the MAPI_UNICODE flag is not set, the strings are in ANSI format. |
| OPEN_IF_EXISTS | Allows the method to succeed even if the folder named in the lpszFolderName parameter already exists by opening the existing folder with that name. Note that message store providers that allow sibling folders to have the same name might fail to open an existing folder if more than one exists with the supplied name. |

| Folder type | Description |
|---|---|
| FOLDER_GENERIC | Create a generic folder. |
| FOLDER_SEARCH | Create a search folder. |

```
// Creating a folder
$result = mapi_folder_createfolder($folder, "New folder", "Description for new folder");
```

See also `mapi_msgstore_openentry()`

### 4.4.10 mapi_folder_setreadflags

**mapi_folder_setreadflags** (*$folder*, *$entryids*[, *$flags* ])

Mark a list of messages in a folder as read.

This function markes selected messages in a given folder. All the messages can be set read or unread in the folder.

**Parameters**

- **$folder** (*mapifolder*) – The folder in which you want mark a number of messages read.
- **$entryids** (*array*) – A list of entryids of messages that are in the given folder. When this array is empty, all the messages in the folder will be marked.
- **$flags** (*long*) – A bitmask of flags that control the creation of a folder.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

| Flag | Description |
|---|---|
| CLEAR_READ_FLAG | Instead of marking the messages read, the messages are marked unread. |

```
// Mark all messages read in the inbox
$result = mapi_folder_createfolder($inbox, array());
```

See also `mapi_message_setreadflag()`

### 4.4.11 mapi_folder_openmodifytable

**mapi_folder_openmodifytable** (*$inbox*)

Returns the IExchangeModifyTable interface.

The IExchangeModifyTable interface is used for the rules table. Use this interface to read and write rules which are applied on incoming e-mails.

**Parameters**

- **$inbox** (*mapifolder*) – The rules table is only available on the default delivery folder.

**Returns** Returns IExchangeModifyTable on success, FALSE on failure

**Return type** resource

```
// Requesting the IExchangeModifyTable interface
$inbox = mapi_msgstore_getreceivefolder($store);
$emt = mapi_folder_openmodifytable($inbox);
```

See also `mapi_rules_gettable()` `mapi_rules_modifytable()`

### 4.4.12 mapi_folder_setsearchcriteria

**mapi_folder_setsearchcriteria** (*$folder*, *$restriction*, *$folderlist*[, *$flags* ])

Set/reset search criteria of a search folder.

mapi_folder_setsearchcriteria can be used to setup a search folder. Once setsearchcriteria has been successfully called, the server will start running the search. A contents table for this folder with show only those items that match the search and will be dynamically built in the background.

Note that after setsearchcriteria has been called, the contents table will not directly show all matches, but will start to build them slowly. You can see if the search is still running by calling mapi_folder_getsearchcriteria.

**Parameters**

- **$folder** (*mapifolder*) – The folder to set the search criteria on. This must be a search folder, ie it must have been created with type FOLDER_SEARCH
- **$restriction** (*array*) – The restriction that items in the search folder must match.
- **$folderlist** (*array*) – EntryID based array of folders of which folders are involved in search operation.
- **$flags** (*long*) – Valid values are BACKGROUND_SEARCH, FOREGROUND_SEARCH, RECURSIVE_SEARCH, RESTART_SEARCH, SHALLOW_SEARCH and STOP_SEARCH. Background and foreground searching is currently not implemented in Kopano.

**Returns** TRUE on success, FALSE on failure

**Return type** bool

```
// Create a search folder, with all objects smaller or equal 100 bytes
$folder = mapi_folder_createfolder($root, "mysearchfolder", "", OPEN_IF_EXISTS,
↪ FOLDER_SEARCH);
mapi_savechanges($folder);
$res = Array(RES_PROPERTY, Array(RELOP => RELOP_LE, ULPROPTAG => PR_MESSAGE_
↪SIZE, VALUE => array (PR_MESSAGE_SIZE => 100) ));

mapi_folder_setsearchcriteria($folder, $res, array($storeProps[PR_IPM_SUBTREE_
↪ENTRYID]), RECURSIVE_SEARCH);
```

See also `mapi_folder_getsearchcriteria()`

### 4.4.13 mapi_folder_getsearchcriteria

**mapi_folder_getsearchcriteria**(*$folder*[, *$flags* ])

Get search criteria and status of a search folder.

mapi_folder_getsearchcriteria returns an associative array with three values: the restriction and folder list from setsearchcriteria, and a searchstate flag. These are stored respectively in the 'restriction', 'folderlist' and 'searchstate' keys in the returned associative array. The searchstate value may be any combination of SEARCH_REBUILD, SEARCH_RUNNING, SEARCH_FOREGROUND and SEARCH_RECURSIVE.

When a folder first starts the search process (after mapi_folder_setsearchcriteria), it will return SEARCH_REBUILD | SEARCH_RUNNING until the search has completed. After this, it will return SEARCH_RUNNING, until the search is restarted or stopped. If the search has stopped, neither SEARCH_REBUILD or SEARCH_RUNNING will be set.

**Parameters**

- **$folder** (*mapifolder*) – The search folder to query.
- **$flags** (*long*) – These flags are currently unused.

**Returns** Associative array with restriction, folder list and searchstate flag

**Return type** array

```
$start = time();
$table = mapi_folder_getcontentstable($searchFolder);

while(time() - $start < 10) {
        $count = mapi_table_getrowcount($table);
        $result = mapi_folder_getsearchcriteria($searchFolder);

        // Stop looping if we have data or the search is finished
        if($count > 0)
                break;

        if(($result["searchstate"] & SEARCH_REBUILD) == 0)
                break; // Search is done

        sleep(1);
}
```

See also `mapi_folder_setsearchcriteria()`

## 4.5 Table functions

Table functions - This section describes functions which are performed on MAPI Table objects.

### 4.5.1 mapi_table_queryallrows

**mapi_table_queryallrows** (*$table*[, *$tagarray* ][, *array $restriction* ])
Queries all the rows in a table and returns a PHP Array

This function reads all the rows of a table and returns the properties specified with the second parameter. An array with properties that should be queried can be given. Also, an array with restrictions can be given as third argument.

**Parameters**

- **$table** (*mapitable*) – The table resource.
- **$tagarray** (*array*) – An array with the propertytags that should be returned
- **$restriction** (*array*) – An array with the restriction that should be applied before querying all rows of the table

**Returns** Array on succes, FALSE on failure

**Return type** array

```
// Reading the rows from a table
// Read the rows with properties PR_ENTRYID and PR_SUBJECT
// Restriction which should restrict messages smaller than or equal to 100␣
↪bytes
$restriction = Array(RES_PROPERTY, Array(RELOP => RELOP_LE, ULPROPTAG => PR_
↪MESSAGE_SIZE, VALUE => array (PR_MESSAGE_SIZE => 100) ));
$array = mapi_table_queryallrows($table, array(PR_ENTRYID, PR_SUBJECT),
↪$restriction);
```

See also `mapi_table_queryrows()` `mapi_table_sort()`

### 4.5.2 mapi_table_queryrows

**mapi_table_queryrows** (*$table*[, *$tagarray* ][, *$start* ][, *$limit* ])
Queries the rows in a table and returns a PHP Array

This function reads at most $limit rows the rows of a table, starting at row $start and returns the properties specified with the second parameter.

**Parameters**

- **$table** (*mapitable*) – The table resource.
- **$tagarray** (*array*) – An array with the propertytags that should be returned
- **$start** (*long*) – The starting row number to read
- **$limit** (*long*) – The maximum amount of rows to read

**Returns** Array on succes, FALSE on failure

**Return type** array

```
// Read the rows with properties PR_ENTRYID and PR_SUBJECT
$array = mapi_table_queryrows($table, array(PR_ENTRYID, PR_SUBJECT), 10, 10);
```

See also `mapi_table_queryallrows()` `mapi_table_sort()`

### 4.5.3 mapi_table_getrowcount

**mapi_table_getrowcount** (*$table*)
Get the amount of rows in a table

Returns the total amount of rows in the table, however, the table may change while processing the rows in the table after retrieving the amount of rows, so you cannot rely on mapi_table_queryrows() to return data for a row which should have existed according to mapi_table_getrowcount().

**Parameters**

- **$table** (*mapitable*) – The table resource.

**Returns** Amount of rows on success, FALSE on failure

**Return type** long

```
// Getting the the rows one by one from a table
// Read the rows with properties PR_ENTRYID and PR_SUBJECT
for ($i = 0; $i < mapi_table_getrowcount($table); $i++) {
        $array = mapi_table_queryrows($table, array(PR_ENTRYID, PR_SUBJECT),
↪$i, 1);
}
```

See also `mapi_table_queryallrows()` `mapi_table_sort()` `mapi_table_queryrows()`

### 4.5.4 mapi_table_sort

**mapi_table_sort** (*$table*, *$sortcolumns*)

Sort the given table according to columns.

This function sets the sort order of a table to the given sorting columns. The MAPI provider may or may not actually perform the sort operation at the time of the call, but usually the actual sort is performed when the data is retrieved from the table from mapi_table_queryrows or mapi_table_queryallrows.

**Parameters**

- **$table** (*mapitable*) – The table resource.
- **$array_sortcolumns** (*array*) – An array of columns that should be sorted. A property is the key, how it should be sorted is the value. Eg. array(PR_SUBJECT => TABLE_SORT_ASCEND, PR_RECEIVED_DATE => TABLE_SORT_DESCEND). Other sort method is TABLE_SORT_COMBINE.

**Returns** TRUE on succes, FALSE on failure

**Return type** boolean

```
// Sorting and retrieving the rows of a table by subject
// Read the rows with properties PR_ENTRYID and PR_SUBJECT
mapi_table_sort($table,array(PR_SUBJECT => TABLE_SORT_ASCEND));

for ($i = 0; $i < mapi_table_getrowcount($table); $i++) {
        $array = mapi_table_queryrows($table, array(PR_ENTRYID, PR_SUBJECT),
↪$i, 1);
}
```

See also `mapi_table_queryallrows()` `mapi_table_queryrows()`

### 4.5.5 mapi_table_restrict

**mapi_table_restrict** (*$table*, *$restriction*)

Restrict the data in a table to certain rules.

This function sets a restriction on the table, narrowing the data in the table.

**Parameters**

- **$table** (*mapitable*) – The table resource.

- **$restriction** (*array*) – The array holding the complete restriction.

**Returns** TRUE on succes, FALSE on failure

**Return type** boolean

```
// Restriction on a mapitable
$restriction = array(RES_CONTENT,
array( FUZZYLEVEL => FL_SUBSTRING | FL_IGNORECASE,
       ULPROPTAG => PR_SUBJECT,
       VALUE => array(PR_SUBJECT=>'spam'))
);
mapi_table_restrict($table, $restriction);

// all mapi_table_queryrows() calls here after only return rows that match the␣
↪restriction
// if the result was false, mapi_last_hresult() will probably be set to MAPI_E_
↪TOO_COMPLEX
mapi_table_restrict($table, array());
```

See also `mapi_table_queryallrows()` `mapi_folder_openmodifytable()` `mapi_rules_gettable()`

## 4.6 Rules functions

Rules functions - This section describes functions which are performed on the rules table.

### 4.6.1 mapi_rules_gettable

**mapi_rules_gettable**(*$emt*)

Returns the rules table from an IExchangeModifyTable interface.

This function opens a table view on the rules table from the IExchangeModifyTable interface.

**Parameters**

- **$emt** (*resource*) – The IExchangeModifyTable interface, retrieved from mapi_folder_openmodifytable ()

**Returns** MAPI table object on success, FALSE on failure

**Return type** array

```
// Retrieving the rules table
$emt = mapi_folder_openmodifytable($inbox);
$rtable = mapi_rules_gettable($emt);
$rules = mapi_table_queryallrows($rtable);
print_r($rules);
```

See also `mapi_table_queryallrows()` `mapi_folder_openmodifytable()` `mapi_rules_modifytable()`

### 4.6.2 mapi_rules_modifytable

**mapi_rules_modifytable**(*$emt*, *$rows*[, *$flags* ])

Modifies the rules table using the IExchangeModifyTable interface.

This function alters the rules table with the current row set. The previous table can be cleared when the ROWLIST_REPLACE is passed in the flags parameter.

**Parameters**

- **$emt** (*resource*) – The IExchangeModifyTable interface, retrieved from mapi_folder_openmodifytable ()
- **$rows** – The rows that need to be added, modified or deleted.
- **$flags** – Set this flag to ROWLIST_REPLACE to empty the current table before altering the table.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Modifying the rules table

// Get the interface
$emt = mapi_folder_openmodifytable($inbox);

$restriction = array(RES_CONTENT,
array( FUZZYLEVEL => FL_SUBSTRING | FL_IGNORECASE,
        ULPROPTAG => PR_SUBJECT,
        VALUE => array(PR_SUBJECT=>'spam'))
);
$action = array(
        array (
                'action' => OP_MOVE,
                'storeentryid' => $publicstore_entryid,
                'folderentryid' => $public_folder_entryid
        ),
);
$rule = array(
        PR_RULE_ACTIONS => $action,
        PR_RULE_CONDITION => $restriction,
        PR_RULE_PROVIDER => 'RuleOrganizer',
        PR_RULE_STATE => ST_ENABLED,
        PR_RULE_NAME => 'move "spam" mails to "important" folder in the public store',
);

$rows = array( 0 => array(
        'rowflags' => ROW_ADD,
        'properties' => $rule
        )
);

$result = mapi_rules_modifytable($emt, $rows);

$rtable = mapi_rules_gettable($emt);
$rules = mapi_table_queryallrows($rtable);
print_r($rules);
```

See also `mapi_table_queryallrows()` `mapi_folder_openmodifytable()` `mapi_rules_gettable()`

## 4.7 Message functions

Message functions - This section describes functions which are performed on MAPI Message objects.

### 4.7.1 mapi_message_setreadflag

**mapi_message_setreadflag** (*$message*, *$flags*)
Sets the readflag of a specific message.

This function changes or sets the PR_MESSAGE_FLAGS property and controls the processing of read reports.

**Parameters**

- **$message** (*mapimessage*) – The message you are requesting the properties of.
- **$flag** (*long*) – A bitmask of flags that control the setting of a message's read flag.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

| Flag | Description |
|---|---|
| CLEAR_READ_FLAG | The MSGFLAG_READ flag should be cleared in PR_MESSAGE_FLAGS and no read report should be sent. |
| CLEAR_NRN_PENDING | The MSGFLAG_NRN_PENDING flag should be cleared in PR_MESSAGE_FLAGS and a nonread report should not be sent. |
| CLEAR_RN_PENDING | The MSGFLAG_RN_PENDING flag should be cleared in PR_MESSAGE_FLAGS and no read report should be sent. |
| GENERATE_RECEIPT_ONLY | A read report should be sent if one is pending, but there should be no change in the state of the MSGFLAG_READ flag. |
| SUPPRESS_RECEIPT | A pending read report should be canceled if a read report had been requested and this call changes the state of the message from unread to read. If this call does not change the state of the message, the message store provider can ignore this flag. |

```
// Marking all messages as read in the inbox
$inbox = mapi_msgstore_getreceivefolder($userstore);
$contents = mapi_folder_getcontentstable($inbox);

// Loop through all the messages in the inbox and open them one by one,
// each time, marking the message read
for ($i = 0; $i < mapi_table_getrowcount($contents); $i++) {
        $props = mapi_table_queryrows($contents, array(PR_ENTRYID), $i, 1);
        $message = mapi_msgstore_openentry($userstore, $props[PR_ENTRYID]);
        mapi_message_setreadflag($message);
}

// Note: you should actually do this with mapi_folder_setreadflags()
```

See also `mapi_getprops()` `mapi_folder_setreadflags()`

### 4.7.2 mapi_message_getattachmenttable

**mapi_message_getattachmenttable**(*mapimessage $message*)

Retrieve attachment table of a message

Returns a table of the attachments of a message. This table contains information about each attachment, most notably PR_ATTACH_NUM which can be passed to mapi_message_openattach to open the attachment object. The table is automatically updated when a new attachment is added or an attachment is removed (even if the attachment table is still open).

**Parameters**

- **$message** (*mapimessage*) – The message you are requesting the attachment table of.

**Returns** mapitable object on success, FALSE on failure

**Return type** mapitable

See also `mapi_message_openattach()`

```
// Opening the attachmenttable of a message
$inbox = mapi_msgstore_getreceivefolder($userstore);
$contents = mapi_folder_getcontentstable($inbox);

// Loop through all the messages in the inbox and open them one by one,
// each time retrieving their attachment table
for ($i = 0; $i < mapi_table_getrowcount($contents); $i++) {
        $props = mapi_table_queryrows($contents, array(PR_ENTRYID), $i, 1);
        $message = mapi_msgstore_openentry($store, $props[PR_ENTRYID]);
        $attachtable = mapi_message_getattachmenttable($message);
        // the attachement table can be processed with the mapi_table_*␣
↪functions aswell
}
```

### 4.7.3 mapi_message_getrecipienttable

**mapi_message_getrecipienttable**(*$message*)

Retrieve recipient table of a message

Each message has a number of recipients (including 0). This function returns a table of recipients for the message. Each row in the table represents one recipient, usually with an e-mail adress in the PR_EMAIL_ADDRESS and 'SMTP' in the PR_ADDRTYPE.

**Parameters**

- **$message** (*mapimessage*) – The message you are requesting the recipient table of.

**Returns** mapitable object on success, FALSE on failure

**Return type** mapitable

```
// Opening a recipienttable
$recipienttable = mapi_message_getrecipienttable ($message);
print "Number of recipients: " . mapi_table_getrowcount($recipienttable);
```

See also `mapi_message_modifyrecipients()`

### 4.7.4 mapi_message_modifyrecipients

**mapi_message_modifyrecipients**(*$message*, *$flags*, *$recipients*)

Add or remove recipients to a message

With this function the recipients of a message can be added, modified or deleted. With the flag the needed action is selected.

The recipient array should be formed like the following example:

```
$recipient[] = array {
        PR_DISPLAY_NAME=>"John Doe",
        PR_EMAIL_ADDRESS=>"john@example.com",
        PR_ADDRTYPE => "SMTP",
        PR_RECIPIENT_TYPE=> MAPI_BCC
}
```

**Parameters**

- **$message** (*mapimessage*) – The message the recipients should be modified.
- **$flags** (*long*) – A flag that can be set to specify what action should be performed with the recipients.

- **$recipients** (*array*) – An array with the properties of the recipients of the message.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

| Flag | Description |
|---|---|
| MOD-RECIP_ADD | The recipients in the recipients array are added to the recipients in the message. |
| MOD-RECIP_MODIFY | The recipients in the recipients array should replace existing recipients. |
| MOD-RECIP_REMOVE | The recipients in the recipients array should be removed from the recipient list. The PR_ROWID should be used as a index for the rows to remove. |

```
// Modifying recipients

$message = mapi_folder_createmessage($folder);

mapi_message_modifyrecipients($message, MODRECIP_ADD,
        array(
                array(
                        PR_DISPLAY_NAME=>"Pete",
                        PR_EMAIL_ADDRESS=>"pete@example.com",
                        PR_ADDRTYPE => "SMTP",
                        PR_RECIPIENT_TYPE=> MAPI_TO
                )
        )
);
```

See also `mapi_folder_createmessage()`

## 4.7.5 mapi_message_openattach

**mapi_message_openattach**(*$message*, *$attachnum*)

Open an attachment from a message

This function returns a mapiattach object by opening an existing attachment in a message. You must specify which attachment to open with the $attachnum parameter. This parameter can be retrieved by looking at the attachment table retrieved with mapi_message_getattachmenttable().

**Parameters**

- **$message** (*mapimessage*) – The message you are requesting the recipient table of.
- **$attachnum** (*long*) – The attachment number of the attachment you wish to open.

**Returns** mapiattach object on success, FALSE on failure

**Return type** mapiattach

```
// Opening an attachment

// Open attachment number 0
$attach = mapi_message_openattach ($message, 0);

// Dump properties of the attachment (filename, etc)
print_r(mapi_attach_getprops($attach));
```

See also `mapi_message_getattachmenttable()` `mapi_message_deleteattach()`

### 4.7.6 mapi_message_createattach

**mapi_message_createattach** (*$message*)

Open an attachment from a message

This function returns a mapiattach object by creating a new attachment in a message.

**Parameters**

- **$message** (*mapimessage*) – The message you are createing an attachment on.

**Returns** mapiattach object on success, FALSE on failure

**Return type** mapiattach

```
// Creating an attachment

// Open attachment number 0
$attach = mapi_message_createattach ($message);

// Set properties of the attachment
$props = Array(PR_DISPLAY_NAME => "testname.txt",
        PR_FILENAME => "testname.txt",
        PR_ATTACH_METHOD => ATTACH_BY_VALUE,
        PR_ATTACH_DATA_BIN => "Hello world!"
);

mapi_setprops($attach, $props);
mapi_savechanges($attach);

// Dump properties of the attachment (filename, etc)
print_r(mapi_attach_getprops($attach));
```

See also `mapi_message_getattachmenttable()` `mapi_message_openattach()` `mapi_message_deleteattach()`

### 4.7.7 mapi_message_deleteattach

**mapi_message_deleteattach** (*$message*, *$attachnum*)

Open an attachment from a message

This function deletes an existing attachment in a message. You must specify which attachment to delete with the $attachnum parameter. This parameter can be retrieved by looking at the attachment table retrieved mapi_message_getattachmenttable ()

**Parameters**

- **$message** (*mapimessage*) – The message you are deleting the attachment from.
- **$attachnum** (*long*) – The attachment number of the attachment you wish to delete.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Delete attachment number 0
if (mapi_message_deleteattach($message, 0) == true) {
        print("successfully removed attachment");
} else {
        print("unable to remove attachment");
}
```

See also `mapi_message_getattachmenttable()` `mapi_message_createattach()` `mapi_message_openattach()`

### 4.7.8 mapi_message_submitmessage

**mapi_message_submitmessage** (*$message*)

Submits a message for sending

The message is marked ready for sending when this function is called. MAPI passes messages to the underlying messaging system in the order in which they are marked for sending. Because of this functionality, a message might stay in a message store for some time before the underlying messaging system can take responsibility for it.

**Parameters**

- **$message** (*mapimessage*) – The message you wish to submit

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Submitting a message

// Set the subject
mapi_setprops($message,Array(PR_SUBJECT) => "New subject"));
mapi_savechanges($message);
mapi_message_submitmessage($message);
```

See also `mapi_savechanges()`

## 4.8 Attachment functions

Attachment functions - This section describes functions which are performed on MAPI Attachment objects.

### 4.8.1 mapi_attach_openbin

**mapi_attach_openbin** (*$attach*, *$proptag*)

Read binary contents of attachment

Retrieves the actual attachment data in the attachment when the attachment has PR_ATTACH_METHOD is 1 (ATTACH_BY_VALUE).

**Parameters**

- **$attach** (*mapiattach*) – The attachment of which to retrieve binary attachment data.
- **$proptag** (*long*) – The specific tag that should be opened

**Returns** Binary string on success, FALSE on failure

**Return type** string

```
// Getting attachment data

// Open attachment number 0
$attach = mapi_message_openattach ($message, 0);

// Output attachment data
print(mapi_attach_openbin($attach, PR_ATTACH_DATA_BIN));
```

See also `mapi_message_openattach()` `mapi_message_getattachmenttable()`

### 4.8.2 mapi_attach_openobj

**mapi_attach_openobj** (*$attach*[, *$flags* ])

Read message contents of attachment

Returns a message object which is embedded in an attachment. This can be done for attachment which have PR_ATTACH_METHOD 5 (ATTACH_EMBEDDED_MSG).

**Parameters**

- **$attach** (*mapiattach*) – The attachment of which to retrieve message attachment data.
- **$flags** (*long*) – Bitmask of flags that controls access to the property.

**Returns** mapimessage object on success, FALSE on failure

**Return type** mapimessage

| Flag | Description |
|---|---|
| MAPI_CREATE | If the property does not exist, it should be created. If the property does exist, the current value of the property should be discarded. When a caller sets the MAPI_CREATE flag, it should also set the MAPI_MODIFY flag. |
| MAPI_MODIFY | Requests read/write access to the property. The default access is read-only. MAPI_MODIFY must be set when MAPI_CREATE is set. |

```
// Getting the attachment message

// Open attachment number 0
$attach = mapi_message_openattach($message, 0);

// Open embedded message
$message = mapi_attach_openobj($attach);

// Show message contents
print_r(mapi_message_getprops($message);
```

See also `mapi_message_openattach()` `mapi_message_getattachmenttable()`

## 4.9 Stream functions

Stream functions - This section describes functions to create streams to objects and how to use these streams.

### 4.9.1 mapi_openpropertytostream

**mapi_openpropertytostream** (*$mapiProp*, *$propertytag*[, *$flags* ])

Opens a property to a stream object

This functions opens a stream on a specific property. It returns a stream that can be used with other stream functions to read and write the data.

> **Warning:** DEPRECATED: use mapi_openproperty instead

**Parameters**

- **$mapiProp** (*mapipropobj*) – A resource that is derived from MAPIProp.
- **$propertytag** (*long*) – The property that should be read.
- **$flags** (*int*) – The default access is read-only.

**Returns** stream on success, FALSE on failure

**Return type** stream

| Flag | Description |
|---|---|
| MAPI_CREATE | If the property does not exist, it should be created. If the property does exist, the current value of the property should be discarded. When a caller sets the MAPI_CREATE flag, it should also set the MAPI_MODIFY flag. |
| MAPI_MODIFY | Requests read/write access to the property. The default access is read-only. MAPI_MODIFY must be set when MAPI_CREATE is set. |

```
// Reading a stream
$bodyStream = mapi_openpropertytostream($message, PR_BODY);
$stat = mapi_stream_stat($bodyStream);
```

See also `mapi_stream_stat()` `mapi_stream_write()` `mapi_stream_setsize()` `mapi_stream_seek()` `mapi_openproperty()`

### 4.9.2 mapi_stream_create

**mapi_stream_create**()

Create a new in-memory stream object

This function can be used to create an in-memory stream object. Any data written to this object can subsequently be read from the same stream mapi_stream_read()

**Returns** resource on success, FALSE on failure

**Return type** resource

See also `mapi_stream_read()` `mapi_stream_write()` `mapi_stream_setsize()` `mapi_stream_seek()`

```
// Creating a new stream

$stream = mapi_stream_create();
mapi_stream_write($stream, "Hello, world!");
```

### 4.9.3 mapi_stream_read

**mapi_stream_read**(*$stream*, *$lbytes*)

Reads data from a stream

This functions read a specific ammount of data from a stream.

**Parameters**

- **$stream** (*stream*) – A resource that contains the stream. This resource is given back by mapi_openproperty()
- **$lbytes** (*int*) – The ammount of bytes that should be read.

**Returns** string on success, FALSE on failure

**Return type** string

```
// Reading from a stream

$body = mapi_stream_read($stream, 100);
echo $body; // will echo the first 100 bytes of the body
```

See also `mapi_openpropertytostream()` `mapi_stream_stat()` `mapi_stream_write()` `mapi_stream_setsize()` `mapi_stream_seek()`

### 4.9.4 mapi_stream_seek

**mapi_stream_seek** (*$stream*, *$bytes*[, *$flags* ])

Moves the internal pointer the given bytes.

This functions seek in a stream. It can move the pointer with the given bytes. The function will return true when the seeking was succesful or false when it wasn't. If no flag is given, STREAM_SEEK_CUR will be assumed.

**Parameters**

- **$stream** (*stream*) – A resource that contains the stream. This resource is given back by mapi_openproperty()
- **$bytes** (*int*) – The ammount of bytes the pointer should be moved.
- **$flags** (*int*) – special flag for seek positioning

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

| Flag | Description |
|---|---|
| STREAM_SEEK_SET | The new seek pointer is an offset relative to the beginning of the stream. |
| STREAM_SEEK_CUR | The new seek pointer is an offset relative to the current seek pointer location. |
| STREAM_SEEK_END | The new seek pointer is an offset relative to the end of the stream. |

```
// Seeking in a stream
mapi_stream_seek($stream, 100);
```

See also `mapi_openpropertytostream()` `mapi_stream_stat()` `mapi_stream_write()` `mapi_stream_setsize()` `mapi_stream_read()`

### 4.9.5 mapi_stream_setsize

**mapi_stream_setsize** (*$stream*, *$newsize*)

Resizes a stream

This functions resizes the stream. With this function it is possible to preallocate space to use with the stream.

**Parameters**

- **$stream** (*stream*) – A resource that contains the stream. This resource is given back by mapi_openproperty()
- **$newsize** (*int*) – The new size of the stream in bytes.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Setting the size of a stream
mapi_stream_setsize($stream, 100);
```

See also `mapi_openpropertytostream()` `mapi_stream_stat()` `mapi_stream_write()` `mapi_stream_commit()` `mapi_stream_read()`

### 4.9.6 mapi_stream_commit

**mapi_stream_commit** (*$stream*)

Commits the stream

This functions commits the stream. Now all changes are saved.

**Parameters**

- **$stream** (*stream*) – A resource that contains the stream. This resource is given back by mapi_openproperty()

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Committing a stream
mapi_stream_commit($stream);
```

See also `mapi_openpropertytostream()` `mapi_stream_stat()` `mapi_stream_write()` `mapi_stream_setsize()` `mapi_stream_read()`

### 4.9.7 mapi_stream_write

**mapi_stream_write** (*$stream, $data*)

Write data to a stream

This functions writes data to the stream. The pointer will also moved the places that is needed. Use mapi_stream_commit to save the stream.

**Parameters**

- **$stream** (*stream*) – A resource that contains the stream. This resource is given back by mapi_openproperty()
- **$data** (*string*) – The bytes that should be written into the stream.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Writing data into a stream
mapi_stream_write($stream, "data");
```

See also `mapi_openpropertytostream()` `mapi_stream_stat()` `mapi_stream_setsize()` `mapi_stream_read()`

### 4.9.8 mapi_stream_stat

**mapi_stream_stat** (*stream $stream*)

Gets the statistics of the stream.

This functions reads the statistics from a stream and returns this in an assoc array. For now only the element 'cb' is supported, this gives the size of the stream.

**Parameters**

- **$stream** – A resource that contains the stream. This resource is given back by mapi_openproperty()

**Returns** Associated array on success, FALSE on failure

**Return type** array

```
// Getting the statistics from a stream
$stat = mapi_stream_stat($stream);
echo "Size :" . $stat['cb']; // will read the stats from the stream.
```

See also `mapi_openpropertytostream()` `mapi_stream_commit()` `mapi_stream_seek()` `mapi_stream_read()`

## 4.10 Addressbook functions

Addressbook functions - This section describes functions to use the addressbook for user and e-mail address lookups.

### 4.10.1 mapi_openaddressbook

**mapi_openaddressbook** (*$session*)

Opens the MAPI Addressbook. On Kopano, this only is the Global Addressbook, containing the Kopano users.

**Parameters**

- **$session** (*mapisession*) – The MAPI Session that is opened before.

**Returns** resource on success, FALSE on failure

**Return type** resource

```
// Open the Addressbook

// First, logon to zarafa to get a session
$session = mapi_logon_zarafa($username, $password, $server);

// Now we can open the addressbook
$addrbook = mapi_openaddressbook($session);
```

See also `mapi_ab_openentry()` `mapi_ab_resolvename()` `mapi_ab_getdefaultdir()`

### 4.10.2 mapi_ab_openentry

**mapi_ab_openentry** (*$addrbook*, *$entryid*[, *$flags* ])

Calls the OpenEntry() function on the Addressbook object.

This function performs the OpenEntry() call on the AddressBook interface.

**Parameters**

- **$addrbook** (*resource*) – The Addressbook resource, retrieved with mapi_openaddressbook().
- **$entryid** (*string*) – This is the EntryID to request from the Addressbook. This can result in opening a MailUser, DistList or an Addressbook Container, which will be returned. When the EntryID is NULL, the top-level Addressbook Container is returned.
- **$flags** (*long*) – MAPI_BEST_ACCESS (default), MAPI_DEFERRED_ERRORS and/or MAPI_MODIFY.

**Returns** A MailUser, DistList or ABContainer is returned as resource on success, FALSE on failure

**Return type** resource

```
// Requesting the top-level AddressBook container

$addrbook = mapi_openaddressbook($session);
$abcontainer = mapi_ab_openentry($addrbook);
```

See also `mapi_openaddressbook()`

### 4.10.3 mapi_ab_resolvename

**mapi_ab_resolvename** (*$addrbook*, *$name_entry*)

Calls the ResolveName() function on the Addressbook object.

This function performs the ResolveName() call on the AddressBook interface.

**Parameters**

- **$addrbook** (*resource*) – The Addressbook resource, retrieved with mapi_openaddressbook().
- **$array** (*array*) – This is an array of arrays of properties, and will be converted to an AdrList structure. This structure will be used to find the given name of a user.

**Returns** MAPI_E_AMBIGUOUS_RECEIP or MAPI_E_NOT_FOUND when these errors occurred, or FALSE on any other error. When the user is found, the array returned has the full information of the user requested

**Return type** array

```
// Resolving a user to e-mail address

$addrbook = mapi_openaddressbook($session);
$user = array( array( PR_DISPLAY_NAME => 'steve' ), array( PR_DISPLAY_NAME =>
↪'milo' ) );
$user = mapi_ab_resolvename($addrbook, $user);
print_r($user);
```

Output:

```
Array
(
  [0] => Array
    (
        [805437470] => SMTP
        [805371934] => John Doe
        [956301315] => 0
        [805503006] => john@example.com

        [268370178] => ...
        [267780354] => ...
        [268304387] => 6
        [267976962] => ...
        [806027522] => SMTP:john@example.com
    )
  [1] => Array
    (
        [805437470] => SMTP
        [805371934] => Richard Miles
        [956301315] => 0
        [805503006] => richard@example.com

        [268370178] => ...
        [267780354] => ...
        [268304387] => 6
        [267976962] => ...
        [806027522] => SMTP:richard@example.com
    )
)
```

See also `mapi_openaddressbook()`

### 4.10.4 mapi_ab_getdefaultdir

**mapi_ab_getdefaultdir** (*$addrbook*)

Calls the GetDefaultDir() function on the Addressbook object.

This function performs the GetDefaultDir() call on the AddressBook interface.

**Parameters**

- **$addrbook** (*resource*) – The Addressbook resource, retrieved with mapi_openaddressbook().

**Returns** EntryID on success, FALSE on failure

**Return type** string

```
// Opening the default Addressbook Directory
$addrbook = mapi_openaddressbook($session);
#entryid = mapi_ab_getdefaultdir($addrbook);
```

See also `mapi_openaddressbook()` `mapi_ab_resolvename()`

## 4.11 Freebusy functions

Freebusy functions - This section describes freebusy specific functions to see the freebusy data of the users.

### 4.11.1 mapi_freebusysupport_open

**mapi_freebusysupport_open** (*$session*[, *$messagestore* ])

Retreive freebusy support object

**Parameters**

- **$session** (*resource*) – The resource id of a session of the user.
- **$messagestore** (*resource*) – The resource id of the messagestore of the user. Only when you update the freebusy data.

**Returns** resource object on success, FALSE on failure

**Return type** resource

```
// Open a freebusysupport object
$fbsupport = mapi_freebusysupport_open($session);
```

See also `mapi_freebusysupport_close()`

### 4.11.2 mapi_freebusysupport_close

**mapi_freebusysupport_close** (*$freebusysupport*)

Close the freebusy support object

**Parameters**

- **$freebusysupport** (*resource*) – The resource id of a freebusysupport object.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
$fbsupport = mapi_freebusysupport_open($session);
// Do something
$result = mapi_freebusysupport_close($fbsupport);
```

See also `mapi_freebusysupport_open()`

### 4.11.3 mapi_freebusysupport_loaddata

**mapi_freebusysupport_loaddata** (*$freebusysupport*, *$fbusers*)

Get a freebusydata object for one of more users.

**Parameters**

- **$freebusysupport** (*resource*) – The resource id of a freebusysupport object.
- **$fbusers** (*array*) – An array of user addressbook entryids

**Returns** Array on success, FALSE on failure

**Return type** array

```
// Get freebusydata object of one user

$fbsupport = mapi_freebusysupport_open($session);
$fbDataArray = mapi_freebusysupport_loaddata($fbsupport, array($userentrid1,
↪$userentrid2) );

if(isset($fbDataArray[0]))
{
        $rangeuser1 = mapi_freebusydata_getpublishrange($fbDataArray[0]);
        print_r($rangeuser1);
}else {
        echo "No freebusy for User 1\n";
}

if(isset($fbDataArray[1]))
{
        $rangeuser2 = mapi_freebusydata_getpublishrange($fbDataArray[1]);
        print_r($rangeuser2);
}else {
        echo "No freebusy for User 2\n";
}

mapi_freebusysupport_close($fbsupport);
```

See also `mapi_freebusysupport_open()` `mapi_freebusysupport_close()`

### 4.11.4 mapi_freebusysupport_loadupdate

**mapi_freebusysupport_loadupdate** (*$freebusysupport*, *$fbusers*)

Get freebusyupdate object of one or more users to write the new freebusydata.

**Parameters**

- **$freebusysupport** (*resource*) – The resource id of a freebusysupport object.
- **$fbusers** (*array*) – An array of user addressbook entryids

**Returns** Array on success, FALSE on failure

**Return type** array

```
// Get freebusyupdate object of one user

$fbupdate = mapi_freebusysupport_loadupdate($fbsupport, array($userentryid));
$result = mapi_freebusyupdate_reset($fbupdate[0]);
```

```
$fbblocks = array(
        array("start" => 1157439600, "end" => 1157468400, "status" => 2),
        array("start" => 1157526000, "end" => 1157554800, "status" => 2)
);

// Add fbblocks
$result = mapi_freebusyupdate_publish($fbupdate[0], $fbblocks);

// Save data
$result = mapi_freebusyupdate_savechanges($fbupdate[0], 1157061600,␣
↪1162335600);
```

See also `mapi_freebusysupport_open()` *mapi_freebusysupport_close*

### 4.11.5 mapi_freebusydata_enumblocks

**mapi_freebusydata_enumblocks** (*$freebusydata*, *$starttime*, *$endtime*)
Get an interface to supports accessing and enumerating free/busy blocks of data for a user within a time range.

**Parameters**

- **$freebusydata** (*resource*) – The resource id of a freebusydata object.
- **$starttime** (*long*) – starttime of the freebusy information in unixtimestamp
- **$endtime** (*long*) – endtime of the freebusy information in unixtimestamp

**Returns** resource on success, FALSE on failure

**Return type** resource

```
// Get freebusy enumblock object of one user and read some data

$enumblock = mapi_freebusydata_enumblocks($fbData, 0, 0xFFFFFFFF);

// Set the cursur on the begining
$result = mapi_freebusyenumblock_reset($enumblock);

//Read the free/busy blocks in a cycles of 2 blocks
while(true)
{
        $blocks = mapi_freebusyenumblock_next($enumblock, 2);
        if($blocks == false)
                break;
        print_r($blocks);
}
```

Output:

```
Array ( [0] => Array ( [start] => 1157439600, [end] => 1157468400, [status] =>␣
↪2 ),
        [1] => Array ( [start] => 1157526000, [end] => 1157554800, [status] =>␣
↪2 )
      )
```

See also `mapi_freebusysupport_open()` `mapi_freebusysupport_close()` `mapi_freebusysupport_loaddata()`

### 4.11.6 mapi_freebusydata_getpublishrange

**mapi_freebusydata_getpublishrange** (*$freebusydata*)
Gets a preset time range for an enumeration of free/busy blocks of data for a user.

Get free/busy publish range, start and end time in unixtimestamp format.

**Parameters**

- **$freebusydata** – The resource id of a freebusydata object.

**Returns** resource on success, FALSE on failure

**Return type** resource

```
// Get freebusy publish range
// For a meaningful example, please See mapi_freebusysupport_loaddata
```

See also `mapi_freebusysupport_open()` `mapi_freebusysupport_close()` `mapi_freebusysupport_loaddata()`

### 4.11.7 mapi_freebusydata_setrange

**mapi_freebusydata_setrange** (*$freebusydata*, *$starttime*, *$endtime*)
Sets the range of time for an enumeration of free/busy block of data for a user.

Sets the freebusy range of time.

**Parameters**

- **$freebusydata** (*resource*) – The resource id of a freebusydata object.
- **$starttime** (*long*) – starttime of the freebusy information in unixtimestamp
- **$endtime** (*long*) – endtime of the freebusy information in unixtimestamp

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Sets the freebusy range of time.
$result = mapi_freebusydata_setrange($freebusydata, 1157061600, 1162335600);
```

See also `mapi_freebusysupport_open()` `mapi_freebusysupport_close()` `mapi_freebusysupport_loaddata()`

### 4.11.8 mapi_freebusyenumblock_reset

**mapi_freebusyenumblock_reset** (*$freebusyenumblock*)
Resets the enumerator by setting the cursor to the beginning.

**Parameters**

- **$freebusyenumblock** (*resource*) – The resource id of a free/busy enumblock object.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Sets the cursor to the beginning
// For a meaningful example, please see mapi_freebusydata_enumblocks
```

See also `mapi_freebusysupport_open()` `mapi_freebusysupport_close()` `mapi_freebusysupport_loaddata()`,

### 4.11.9 mapi_freebusyenumblock_skip

**mapi_freebusyenumblock_skip** (*$freebusyenumblock*)
Skips a specified number of blocks of free/busy data.

**Parameters**

- **$freebusyenumblock** (*resource*) – The resource id of a free/busy enumblock object.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Skips 2 blocks of free/busy data.

$enumblock = mapi_freebusydata_enumblocks($fbData, 0, 0xFFFFFFFF);

// Set the cursur on the begining
$result = mapi_freebusyenumblock_reset($enumblock);

// Skip 2 free/busy blocks
$result = mapi_freebusyenumblock_skip($enumblock, 2);

// Read the 2 free/busy blocks
$blocks = mapi_freebusyenumblock_next($enumblock, 2);
```

See also `mapi_freebusysupport_open()` `mapi_freebusysupport_close()` `mapi_freebusysupport_loaddata()` `mapi_freebusydata_enumblocks()`

### 4.11.10 mapi_freebusyenumblock_restrict

**mapi_freebusyenumblock_restrict** (*$freebusyenumblock*, *$starttime*, *$endtime*)
Restricts the enumeration to a specified time period.

**Parameters**

- **$freebusyenumblock** (*resource*) – The resource id of a free/busy enumblock object.
- **$starttime** (*long*) – starttime of the free/busy information in unixtimestamp
- **$endtime** (*long*) – endtime of the free/busy information in unixtimestamp

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Restricts the enumeration to a specified time period.

$enumblock = mapi_freebusydata_enumblocks($fbData, 0, 0xFFFFFFFF);

// Set the cursur on the begining
$result = mapi_freebusyenumblock_reset($enumblock);

//Restrict the free/busy data
$result = mapi_freebusyenumblock_restrict($enumblock, 0x12345578, 0xFFFF0000);

//Read the 2 free/busy blocks
$blocks = mapi_freebusyenumblock_next($enumblock, 2);
```

See also `mapi_freebusysupport_open()` `mapi_freebusysupport_close()` `mapi_freebusysupport_loaddata()` `mapi_freebusydata_enumblocks()`

### 4.11.11 mapi_freebusyenumblock_next

**mapi_freebusyenumblock_next** (*$freebusyenumblock*, *$celt*, *$endtime*)

Gets the next specified number of blocks of free/busy data in an enumeration.

**Parameters**

- **$freebusyenumblock** (*resource*) – The resource id of a free/busy enumblock object.
- **$celt** (*long*) – items to retreive
- **$endtime** (*long*) – endtime of the free/busy information in unixtimestamp

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Gets the next 10 blocks of free/busy data.
// For a meaningful example, please see mapi_freebusydata_enumblocks
```

See also `mapi_freebusysupport_open()` `mapi_freebusysupport_close()` `mapi_freebusysupport_loaddata()` `mapi_freebusydata_enumblocks()`

### 4.11.12 mapi_freebusyupdate_reset

**mapi_freebusyupdate_reset** (*$freebusyupdate*)

Remove all current free/busy data

**Parameters**

- **$freebusysupport** (*resource*) – The resource id of a freebusyupdate object.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

:: // Remove all current free/busy data // For a meaningful example, please see mapi_freebusysupport_loadupdate

See also `mapi_freebusysupport_open()` `mapi_freebusysupport_close()` `mapi_freebusysupport_loadupdate()`

### 4.11.13 mapi_freebusyupdate_publish

**mapi_freebusyupdate_publish** (*$freebusyupdate*, *$freebusyblocks*)

Publish a specified number of blocks of free/busy data. May be called more than once successively.

**Parameters**

- **$freebusysupport** (*resource*) – The resource id of a freebusyupdate object.
- **$freebusyblocks** (*array*) – An array of free/busy data blocks.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Publish 2 blocks of free/busy data
// For a meaningful example, please see mapi_freebusysupport_loadupdate
```

See also `mapi_freebusysupport_open()` `mapi_freebusysupport_close()` `mapi_freebusysupport_loadupdate()`

### 4.11.14 mapi_freebusyupdate_savechanges

**mapi_freebusyupdate_savechanges** (*$freebusyupdate*, *$starttime*, *$endtime*)
Save the free/busy data with time frame between the begintime and endtime.

**Parameters**

- **$freebusysupport** (*resource*) – The resource id of a freebusyupdate object.
- **$starttime** (*long*) – starttime of the freebusy information in unixtimestamp
- **$endtime** (*long*) – endtime of the freebusy information in unixtimestamp

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Save the free/busy data
// For a meaningful example, please see mapi_freebusysupport_loadupdate
```

See also mapi_freebusysupport_open() mapi_freebusysupport_close() mapi_freebusysupport_loadupdate()

## 4.12 Kopano specific text functions

Kopano specific text functions - This section describes special functions used to handle texts of a mailbody.

### 4.12.1 mapi_decompressrtf

**mapi_decompressrtf** (*$compressedRTF*)
Decompresses a compressed RTF stream from the PR_RTF_COMPRESSED property.

This function will decompress the RTF Body from a message and return the decompressed RTF.

**Parameters**

- **$compressedRTF** (*string*) – A string with the compressed RTF information. This information can be retrieved with mapi_openproperty($message, PR_RTF_COMPRESSED)

**Returns** string on success, FALSE on failure

**Return type** string

```
// Decompress a RTF body
$rtf = mapi_openproperty($inbox, PR_RTF_COMPRESSED);
mapi_decompressrtf($rtf);
```

## 4.13 Kopano specific functions

Kopano specific functions - This section describes Kopano specific functions to manage users and groups and get and set permissions on folders.

### 4.13.1 mapi_zarafa_createuser

**mapi_zarafa_createuser** (*$storeid*, *$username*, *$password*, *$fullname*, *$emailaddress*[, *$isnonactive* ][, *$isadmin* ])
Creates a new user on an Kopano server

Creates a new user on the Kopano server. This can only be done when a store is specified which is logged on to a server with administration rights. The user is then created, but is initially 'nonactive' after being created. If the username existed before the call to mapi_zarafa_createstore(), an error is generated. The returned value can be used in a call to mapi_zarafa_setuser to set user settings of the created user.

**Parameters**

- **$store** (*store*) – A store that is logged on to an Kopano server, with sufficient permissions to create users.
- **$username** (*string*) – The username of the user to be added. The user must not exist prior to calling mapi_zarafa_createuser.
- **$password** (*string*) – The login password for this user
- **$fullname** (*string*) – The full name of the user as to be displayed to human users (e.g. 'John Doe')
- **$emailaddress** (*string*) – The fully qualified e-mail address of the user (e.g. 'john@example.com')
- **$isnonactive** (*long*) – Specifies whether the account is non-active. Specify 0 here to activate an account.
- **$isadmin)** (*long*) – Specifies whether the account is an administator user. Specify 0 here for a normal account, 1 for an admin account, 2 for a system admin account (This level allows the user to manage companies).

**Returns** User ID on success, FALSE on failure

**Return type** long

```
// Creating a user

$session = mapi_logon_zarafa($admin_user, $password, $serverlocation);

$stores = mapi_getmsgstorestable($session);
$storeslist = mapi_table_queryallrows($stores);
$adminstore = mapi_openmsgstore($session, $storeslist[0][PR_ENTRYID]);

$userid = mapi_zarafa_createuser($adminstore, "newuser", "pass", "New User Name
↪", "john@zarafa.com");

if($userid == false) {
        print "Error creating user\n";
}
```

See also `mapi_zarafa_setuser()`

### 4.13.2 mapi_zarafa_setuser

**mapi_zarafa_setuser** (*$store*, *$userid*, *$username*, *$fullname*, *$emailaddress*, *$password*, *$isnonactive*, *$isadmin*)

Sets user information for a specific user on a Kopano server

This function sets the user information on a user previously created (with mapi_zarafa_createuser). This call can only be done by users with administration rights on the server.

**Parameters**

- **$store** (*store*) – A store that is logged on to a Kopano server, with sufficient permissions to create users.
- **$userid** (*long*) – The ID of the user to update

- **$username** (*string*) – The username of the user, this is the name which was used to login into the system.
- **$fullname** (*string*) – The full name of the user as to be displayed to human users (e.g. 'John Doe')
- **$emailaddress** (*string*) – The fully qualified e-mail address of the user (e.g. 'john@example.com')
- **$password** (*string*) – The login password for this user
- **$isnonactive** (*long*) – Specifies whether the account is non-active. Specify 0 here to activate an account.
- **$isadmin)** (*long*) – Specifies whether the account is an administator user. Specify 0 here for a normal account, 1 for an admin account, 2 for a system admin account (This level allows the user to manage companies).

**Returns** TRUE on success, FALSE on failure

**Return type** long

```
// Setting user information

$store = mapi_openmsgstore_zarafa("user","password");
$userid = mapi_zarafa_createuser($store, "newuser", "pass", "New User Name",
↪"john@example.com");

if($userid == false)
        print "Error creating user\n";
else
        mapi_zarafa_setuser($store, $userid, 'John Doe', 'john@example.com',
↪'password', 0);
```

See also `mapi_zarafa_createuser()`

### 4.13.3 mapi_zarafa_createstore

**mapi_zarafa_createstore** (*$storeid*, *$storetype*, *$userid*)

Creates a store on a Kopano server

Creates a complete store either for a specific user or creates a public store. When creating a user store, the standard folders 'Inbox', 'Outbox', etc. are all created and added to the store.

**Parameters**

- **$store** (*store*) – A store that is logged on to a Kopano server, with sufficient permissions to create users.
- **$storetype** (*long*) – The type of the store. (0 = private mailbox, 1 = public store)
- **$userid** (*long*) – The user ID to which the store should be coupled if this is a private mailbox

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Creating a store

$store = mapi_openmsgstore_zarafa("user","password");
$userid = mapi_zarafa_createuser($store, "newuser");

if($userid == false)
        print "Error creating user\n";
```

```
mapi_zarafa_setuser($store, $userid, 'John Doe', 'john@example.com', 'password
↪', 0);
mapi_zarafa_createstore ($store,0,$userid);
```

See also `mapi_zarafa_createuser()`

### 4.13.4 mapi_zarafa_deleteuser

**mapi_zarafa_deleteuser** (*$store*, *$username*)

Deletes a Kopano user.

This function deletes a user from the Kopano server. The deleted store will be moved to an Admin folder in the toplevel of the public store.

**Parameters**

- **$store** – The store resource of an admin user.
- **$username** – The loginname of the user to delete.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Deleting a user, deleting store with it.
$result = mapi_zarafa_deleteuser($store, "john");
```

See also `mapi_zarafa_createuser()`

### 4.13.5 mapi_zarafa_getuserlist

**mapi_zarafa_getuserlist** (*$messagestore*[, *$companyid* ])

Retreive a list of Kopano users.

Returns an associative array of all users, with the username as key, and the full name as value.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$companyid** (*int*) – This companyid for which we are requesting the members of.

**Returns** Associative array on success, FALSE on failure

**Return type** array

```
// Listing the Kopano users

$tempStoreslist = mapi_openmsgstore_zarafa($username, "password", "http://
↪localhost:236/zarafa");
$msgstore = $tempStoreslist[0];
$users = mapi_zarafa_getuserlist($msgstore);
print_r($users);
```

See also `mapi_zarafa_createuser()`

### 4.13.6 mapi_zarafa_getuser_by_id

**mapi_zarafa_getuser_by_id** (*$messagestore*, *$userid*)

Retreive user information for zarafa user.

Returns an associative array of information for the specified user, or FALSE of the user does not exist or an other error occurred. The associative array contains 'userid', 'username', 'fullname', 'emailaddress', 'nonactive' and 'admin'.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user
- **$userid** (*int*) – The id of the user whose information is te be returned

**Returns** Associative array on success, FALSE on failure

**Return type** array

```
// Getting user information

$info = mapi_zarafa_getuser_by_id($msgstore, 10);

print $info["userid"];
print $info["username"];
print $info["fullname"];
print $info["emailaddress"];
print $info["admin"];
print $info["nonactive"];
```

See also `mapi_zarafa_setuser()` `mapi_zarafa_getuser_by_name()`

### 4.13.7 mapi_zarafa_getuser_by_name

See also `mapi_zarafa_getuser()`

### 4.13.8 mapi_zarafa_getuser

**mapi_zarafa_getuser**(*$messagestore*, *$username*)

Retreive user information for zarafa user.

Returns an associative array of information for the specified user, or FALSE of the user does not exist or an other error occurred. The associative array contains 'userid', 'username', 'fullname', 'emailaddress', 'nonactive' and 'admin'.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user
- **$username** (*string*) – The username of the user whose information is te be returned

**Returns** Associative array on success, FALSE on failure

**Return type** array

```
// Getting user information

$info = mapi_zarafa_getuser($msgstore, "username");

print $info["userid"];
print $info["username"];
print $info["fullname"];
print $info["emailaddress"];
print $info["admin"];
print $info["nonactive"];
```

See also `mapi_zarafa_setuser()` `mapi_zarafa_getuser_by_id()`

### 4.13.9 mapi_zarafa_getgroup_by_id

**mapi_zarafa_getgroup_by_id**(*$messagestore*, *$groupid*)

Retrieve group information for zarafa group.

Returns an associative array of information for the specified group, or FALSE of the group does not exist or an other error occurred. The associative array contains 'groupid' and 'groupname'.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user
- **$groupid** (*int*) – The id of the group whose information is te be returned

**Returns** Associative array on success, FALSE on failure

**Return type** array

```
// Getting group information

$info = mapi_zarafa_getgroup_by_id($msgstore, 2);

print $info["groupid"];
print $info["groupname"];
```

See also `mapi_zarafa_setgroup()` `mapi_zarafa_getgroup_by_name()`

### 4.13.10 mapi_zarafa_getgroup_by_name

See also `mapi_zarafa_getgroup()`

### 4.13.11 mapi_zarafa_getgroup

**mapi_zarafa_getgroup**(*$messagestore*, *$group*)

Retrieve group information for zarafa group.

Returns an associative array of information for the specified group, or FALSE of the group does not exist or an other error occurred. The associative array contains 'groupid' and 'groupname'.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$groupname** (*string*) – The name of the group whose information is te be returned

**Returns** Associative array on success, FALSE on failure

**Return type** array

```
// Getting group information

$info = mapi_zarafa_getgroup($msgstore, "Everyone");

print $info["groupid"];
print $info["groupname"];
```

See also `mapi_zarafa_setgroup()` `mapi_zarafa_getgroup_by_id()`

### 4.13.12 mapi_zarafa_setgroup

**mapi_zarafa_setgroup** (*$messagestore*, *$groupid*, *$groupname*)
Set group information for a specific Kopano group.

Modifies the given group, setting the groupname. Basically, this is a 'rename' of a group, as groups currently have no other information apart from their name.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$groupid** (*long*) – The ID of the group to be changed
- **$groupname** (*string*) – The name of the group whose information is te be returned

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Changing group information
mapi_zarafa_setgroup($msgstore, 5, "NewName");
```

See also `mapi_zarafa_getgroup()`

### 4.13.13 mapi_zarafa_getgrouplist

**mapi_zarafa_getgrouplist** (*$messagestore*[, *$companyid* ])
Get list of all groups

Returns an array of all groups. Each list item is an associative array with the keys 'groupname' and 'groupid'. There are as many items in the returned array as there are groups that are viewable by the user

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$companyid** (*int*) – The companyid for which we are requesting the groups

**Returns** Array on success, FALSE on failure

**Return type** array

```
// Listing groups
$array = mapi_zarafa_getgrouplist($msgstore);
```

See also `mapi_zarafa_getgroup()`

### 4.13.14 mapi_zarafa_creategroup

**mapi_zarafa_creategroup** (*$messagestore*, *$groupname*)
Create a new group.

This function creates a new group with the specified name on the server. After creation, the group will contain no users.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$groupname** (*string*) – The name of the group to be created.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Creating a group
$groupid = mapi_zarafa_creategroup($msgstore, "newgroup");
```

See also `mapi_zarafa_deletegroup()`

### 4.13.15 mapi_zarafa_deletegroup

**mapi_zarafa_deletegroup** (*$messagestore*, *$groupname*)

Delete an existing group.

This function deletes a group on the server. Any users within this group are left untouched.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$groupname** (*string*) – The name of the group to be created.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Deleting a group
mapi_zarafa_deletegroup($msgstore,"newgroup");
```

See also `mapi_zarafa_creategroup()`

### 4.13.16 mapi_zarafa_addgroupmember

**mapi_zarafa_addgroupmember** (*$messagestore*, *$groupid*, *$userid*)

Add a user to a group.

This function adds a user to an existing group.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$groupid** (*int*) – The id of the group to add the user to.
- **$userid** (*int*) – The id of the user to add to the group.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Adding a user to a group
mapi_zarafa_addgroupmember($msgstore,$groupid,$userid);
```

See also `mapi_zarafa_deletegroupmember()`

### 4.13.17 mapi_zarafa_deletegroupmember

**mapi_zarafa_deletegroupmember** (*$messagestore*, *$groupid*, *$userid*)

Delete a user from a group.

This function deletes a user from an existing group.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$groupid** (*int*) – The id of the group to remove the user from.
- **$userid** (*int*) – The id of the user to remove from the group.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Deleting a user from a group
mapi_zarafa_deletegroupmember($msgstore,$groupid,$userid);
```

See also `mapi_zarafa_addgroupmember()`

### 4.13.18 mapi_zarafa_getgrouplistofuser

**mapi_zarafa_getgrouplistofuser**(*$messagestore*, *$userid*)

Get a list of groups for a specific user.

This function returns a list of groups in the same way as mapi_zarafa_getgrouplist, but only returns the groups of which the specified user is a member.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$username** (*long*) – The id of the user to be queried.

**Returns** Array on success, FALSE on error

**Return type** array

```
// Getting groups of a user
$groups = mapi_zarafa_getgrouplistofuser($msgstore,2);
```

See also `mapi_zarafa_getgrouplist()`

### 4.13.19 mapi_zarafa_getuserlistofgroup

**mapi_zarafa_getuserlistofgroup**(*$messagestore*, *$userid*)

Get a list of users for a specific group.

This function returns a list of users in the same way as mapi_zarafa_getuserlist, but only returns the users that are members of the specified group.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$groupid** (*long*) – The id of the group to be queried.

**Returns** Array on success, FALSE on error

**Return type** array

```
// Getting users of a group
$groups = mapi_zarafa_getuserlistofgroup($msgstore,1);
```

See also `mapi_zarafa_getuserlist()`

### 4.13.20 mapi_zarafa_createcompany

**mapi_zarafa_createcompany** (*$messagestore*, *$companyname*)

Create a new company.

This function creates a new company with the specified name on the server. After creation, the company will contain no users.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$companyname** (*string*) – The name of the company to be created.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Creating a company
$groupid = mapi_zarafa_createcompany($msgstore, "newcompany");
```

See also `mapi_zarafa_deletecompany()`

### 4.13.21 mapi_zarafa_deletecompany

**mapi_zarafa_deletecompany** (*$messagestore*, *$companyname*)

Delete an existing company.

This function deletes a company on the server. Any users within this company are left untouched.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$companyname** (*string*) – The name of the company to be deleted.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Deleting a company
mapi_zarafa_deletecompany($msgstore,"delcompany");
```

See also `mapi_zarafa_createcompany()`

### 4.13.22 mapi_zarafa_getcompany_by_id

**mapi_zarafa_getcompany_by_id** (*$messagestore*, *$companyid*)

Retrieve company information for zarafa company.

Returns an associative array of information for the specified company, or FALSE of the company does not exist or an other error occurred. The associative array contains 'companyid' and 'companyname'.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$companyid** (*int*) – The id of the company whose information is te be returned

**Returns** Associative array on success, FALSE on failure

**Return type** array

```
// Getting company information
$info = mapi_zarafa_getcompany_by_id($msgstore, 2);

print $info["companyid"];
print $info["companyname"];
```

See also `mapi_zarafa_getcompany_by_name()`

### 4.13.23 mapi_zarafa_getcompany_by_name

**mapi_zarafa_getcompany_by_name** (*$messagestore*, *$companyname*)

Retrieve company information for zarafa company.

Returns an associative array of information for the specified company, or FALSE of the company does not exist or an other error occurred. The associative array contains 'companyid' and 'companyname'.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$companyname** (*string*) – The name of the company whose information is te be returned

**Returns** Associative array on success, FALSE on failure

**Return type** array

```
// Getting company information
$info = mapi_zarafa_getcompany_by_name($msgstore, "default");

print $info["companyid"];
print $info["companyname"];
```

See also `mapi_zarafa_getcompany_by_id()`

### 4.13.24 mapi_zarafa_getcompanylist

**mapi_zarafa_getcompanylist** (*$messagestore*)

Get list of all companies

Returns an array of all companies on the server.Each list item is an associative array with the keys 'companyname' and 'companyid'. There are as many items in the returned array as there are companies

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.

**Returns** Associative array on success, FALSE on failure

**Return type** array

See also `mapi_zarafa_getcompany_by_name()` `mapi_zarafa_getcompany_by_id()`

```
// Listing companies
$array = mapi_zarafa_getcompanylist($msgstore);
```

### 4.13.25 mapi_zarafa_add_company_remote_viewlist

**mapi_zarafa_add_company_remote_viewlist** (*$messagestore*, *$setcompanyid*, *$companyid*)

Add a company to the remote view list of a different company

This will add a company (setcompanyid) to the remote view list of a different company (companyid). This will allow the members of company 'companyid' to view the members of company 'setcompanyid'.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$setcompanyid** (*int*) – The company to be added to the remote view list.
- **$companyid** (*int*) – The company which remote view list should be edited.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Add a company to the remote view list of a different company
mapi_zarafa_add_company_remote_viewlist($msgstore, $setcompanyid, $companyid);
```

See also `mapi_zarafa_del_company_remote_viewlist()`

### 4.13.26 mapi_zarafa_del_company_remote_viewlist

**mapi_zarafa_del_company_remote_viewlist** (*$messagestore*, *$setcompanyid*, *$companyid*)

Delete a company from the remote view list of a different company

This will delete a company (setcompanyid) from the remote view list of a different company (companyid). This will prevent the members of company 'companyid' to view the members of company 'setcompanyid'.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$setcompanyid** (*int*) – The company to be deleted from the remote view list.
- **$companyid** (*int*) – The company which remote view list should be edited.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Delete a company from the remote view list of a different company
mapi_zarafa_del_company_remote_viewlist($msgstore, $setcompanyid, $companyid);
```

See also `mapi_zarafa_add_company_remote_viewlist()`

### 4.13.27 mapi_zarafa_get_remote_viewlist

**mapi_zarafa_get_remote_viewlist** (*$messagestore*, *$companyid*)

List all companies in the remote view list

This will return the remote view list for the specified company.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$companyid** (*int*) – The company for which we are requesting the remote view list.

**Returns** Array on success, FALSE on failure

**Return type** array

```
// List all companies in the remote view list
$companies = mapi_zarafa_get_remote_viewlist($msgstore, $companyid);
print_r($companies);
```

See also `mapi_zarafa_add_company_remote_viewlist()` `mapi_zarafa_del_company_remote_viewl`

### 4.13.28 mapi_zarafa_add_user_remote_adminlist

**mapi_zarafa_add_user_remote_adminlist** (*$messagestore*, *$userid*, *$companyid*)

Add a user to the remote admin list of a company

This will add a user to the remote admin list of a company. This will allow the user to perform administrator tasks over the specified company.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$userid** (*int*) – The user to be added to the remote administrator list.
- **$companyid** (*int*) – The company for which the user will be set as administrator.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
//Add a user to the remote admin list of a different company
mapi_zarafa_add_user_remote_adminlist($msgstore, $userid, $companyid);
```

See also `mapi_zarafa_del_user_remote_adminlist()`

### 4.13.29 mapi_zarafa_del_user_remote_adminlist

**mapi_zarafa_add_user_remote_adminlist** (*$messagestore*, *$userid*, *$companyid*)

Delete a user from the remote admin list of a company

This will delete an user from the remote admin list of a company. This will strip the administrator rights for the user on the specified company.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$userid** (*int*) – The user to be deleted from the remote administrator list.
- **$companyid** (*int*) – The company for which the user will no longer be administrator.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Delete a user from the admin list of a company
mapi_zarafa_del_user_remote_adminlist($msgstore, $userid, $companyid);
```

See also `mapi_zarafa_add_user_remote_adminlist()`

### 4.13.30 mapi_zarafa_get_remote_adminlist

**mapi_zarafa_get_remote_adminlist** (*$messagestore*, *$companyid*)

List all users in the remote admin list

This will return the remote admin list for the specified company.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$companyid** (*int*) – The company for which we are requesting the remote admin list.

**Returns** Array on success, FALSE on failure

**Return type** array

```
// List all users in the remote admin list

$users = mapi_zarafa_get_remote_adminlist($msgstore, $companyid);
print_r($users);
```

See also `mapi_zarafa_add_user_remote_adminlist()` `mapi_zarafa_del_user_remote_adminlist()`

### 4.13.31 mapi_zarafa_add_quota_recipient

**mapi_zarafa_add_quota_recipient** (*$messagestore*, *$companyid*, *$recipientid*, *$type*)

Add a recipient to the recipient list of a company

When a user exceeds his quota, he will receive a warning email. With this function you can edit the list of additional recipients for this email. With the type argument you can set if the recipient list of userquota or companyquota warnings should be edited.

**Parameters**

- **$messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$companyid** (*int*) – The company for which to edit the recipientlist.
- **$recipientid** (*int*) – The user to be added to the recipientlist.
- **$type** (*int*) – The recipientlist type (USEROBJECT_TYPE_USER or USEROBJECT_TYPE_COMPANY).

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Add a recipient to the recipient list of a company
mapi_zarafa_add_quota_recipient($msgstore, $companyid, $recipientid,␣
↪USEROBJECT_TYPE_COMPANY);
```

See also `mapi_zarafa_del_quota_recipient()` `mapi_zarafa_get_quota_recipientlist()`

### 4.13.32 mapi_zarafa_del_quota_recipient

**mapi_zarafa_del_quota_recipient** (*$messagestore*, *$companyid*, *$recipientid*, *$type*)

Delete a recipient from the recipient list of a company

When a user exceeds his quota, he will receive a warning email. With this function you can edit the list of additional recipients for this email. With the type argument you can set if the recipient list of userquota or companyquota warnings should be edited.

**Parameters**

- **messagestore** (*resource*) – The resource id of the messagestore of the logged-on user.
- **$companyid** (*int*) – The company for which to edit the recipientlist.

- **`$recipientid`** (*int*) – The user to be removed from the recipientlist.
- **`$type`** (*int*) – The recipientlist type (USEROBJECT_TYPE_USER or USEROBJECT_TYPE_COMPANY).

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Delete a recipient from the recipient list of a company
mapi_zarafa_del_quota_recipient($msgstore, $companyid, $userid, USEROBJECT_
↪TYPE_COMPANY);
```

See also `mapi_zarafa_add_quota_recipient()` `mapi_zarafa_get_quota_recipientlist()`

### 4.13.33 mapi_zarafa_get_quota_recipientlist

**`mapi_zarafa_get_quota_recipientlist`** (*$messagestore*, *$userid*)

List all users in the recipient list

When a user exceeds his quota, he will receive a warning email. With this function you can request the list of additional recipients for this email. The userid as argument can either be a userid or companyid.

**Parameters**

- **`$messagestore`** (*resource*) – The resource id of the messagestore of the logged-on user.
- **`$userid`** (*int*) – The user for which we are requesting the recipientlist.

**Returns** Array on success, FALSE on failure

**Return type** array

```
// List all users in the recipient list

$users = mapi_zarafa_get_quota_recipientlist($msgstore, $userid);
print_r($users);
```

See also `mapi_zarafa_add_quota_recipient()` `mapi_zarafa_del_quota_recipient()`

### 4.13.34 mapi_zarafa_getpermissionrules

**`mapi_zarafa_getpermissionrules`** (*$mapiobject*, *$acl_type*)

Get a list of ACLs from an object.

This function returns an array of set ACLs of the requested type.

The type field is one of: ACCESS_TYPE_DENIED, ACCESS_TYPE_GRANT, ACCESS_TYPE_BOTH

The 'rights' field in the array is the access value. These values are defined in the ecRights* defines, found in include/mapidefs.php. The values are bitwise OR-ed. 1531 == 0x5FB == ecRightsAll.

The state field is one of: RIGHT_NORMAL, RIGHT_NEW, RIGHT_MODIFY, RIGHT_DELETED, RIGHT_AUTOUPDATE_DENIED. In the mapi_zarafa_getpermissionrules(), only RIGHT_NORMAL is returned.

**Parameters**

- **`$mapiobject`** (*resource*) – This is a MAPI object. It can be a store, folder, message or attachment.
- **`$acl_type`** (*long*) – The ACL type to query. This is one of: ACCESS_TYPE_DENIED, ACCESS_TYPE_GRANT, ACCESS_TYPE_BOTH

**Returns** Array on success, FALSE on failure

**Return type** array

**Note:** Only ACLs on the store and on a folder work.

```
// Getting the ACL list of the store.
$acls = mapi_zarafa_getpermissionrules($msgstore, ACCESS_TYPE_GRANT);
print_r($acls);
```

Output:

```
Array
(
[0] => Array
    (
        [userid] => 4
        [type] => 2
        [rights] => 1531
        [state] => 0
    )

)
```

See also `mapi_zarafa_setpermissionrules()` `mapi_zarafa_getuserlist()`

### 4.13.35 mapi_zarafa_setpermissionrules

**mapi_zarafa_getpermissionrules**(*$mapiobject*, *$acls*)

Set a list of ACLs on an object.

This function sets returns an array of set ACLs of the requested type.

Each entry contains the following fields: userid, type, rights, state.

| Returns | Description |
|---|---|
| Userid | This is an id from a user. Use mapi_getuserlist() to get all valid users. |
| Type | Type is one of ACCESS_TYPE_DENIED, ACCESS_TYPE_GRANT, ACCESS_TYPE_BOTH |
| Rights | Rights is an OR-ed value, by ecRights* defines, found in include/mapi/mapidefs.php |
| State | State is one of: RIGHT_NEW, RIGHT_MODIFY, RIGHT_DELETED. RIGHT_NORMAL is not used, since it is no update action. RIGHT_AUTOUPDATE_DENIED can be OR-ed to an action to automatically set the inverted DENY action. |

**Parameters**

- **$mapiobject** (*resource*) – This is a MAPI object. It can be a store, folder, message or attachment.
- **$acls** (*array*) – An Array containing ACL settings.

**Returns** Array on success, FALSE on failure

**Return type** array

```
// Setting the ACL list on the inbox.
$acls = array( 0 => array(
            'userid'    => 3,
            'type'      => ACCESS_TYPE_GRANT,
            'rights'    => ecRightsFullControl,
            'state'     => RIGHT_NEW | RIGHT_AUTOUPDATE_DENIED
        )
```

```
);

$ret = mapi_zarafa_setpermissionrules($inbox, $acls);
```

See also `mapi_zarafa_getpermissionrules()` `mapi_zarafa_getuserlist()`

### 4.13.36 mapi_inetmapi_imtoinet

**mapi_inetmapi_imtoinet** (*$session*, *$addrbook*, *$message*, *$flags*)
Converts a MAPI Message into an RFC822-formatted e-mail stream.

**Parameters**

- **$session** (*resource*) – Session on which the message was opened
- **$addrbook** (*resource*) – Addressbook to resolve email addresses
- **$message** (*resource*) – Message to convert to RFC822
- **$flags** (*array*) – Reserved.

**Returns** stream on success, FALSE on failure

**Return type** stream

```
// Read the message as RFC822-formatted e-mail stream.
$stream = mapi_inetmapi_imtoinet($session, $addrBook, $message, array());
```

### 4.13.37 mapi_inetmapi_imtomapi

**mapi_inetmapi_imtomapi** (*$session*, *$store*, *$addrbook*, *$message*, *$eml*, *$flags*)
Converts a RFC822-formatted e-mail into a MAPI Message.

**Parameters**

- **$session** (*resource*) – Session on which the message was opened
- **$store** (*resource*) – Store the users MAPIStore
- **$addrbook** (*resource*) – Addressbook to resolve email addresses
- **$message** (*resource*) – MAPI Message to convert the RFC822 message into
- **$eml** (*resource*) – RFC822 eml to convert to a MAPI Message
- **$flags** (*array*) – Reserved.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Convert an RFC822-formatted e-mail to a MAPI Message
$ok = mapi_inetmapi_imtomapi($session, $store, $ab, $msg, $eml, array());
```

### 4.13.38 mapi_mapitoical

**mapi_mapitoical** (*$session*, *$addrbook*, *$message*, *$options*)
Converts a MAPI Message into an ICS file.

**Parameters**

- **$session** (*resource*) – Session on which the message was opened
- **$addrbook** (*resource*) – Addressbook to resolve email addresses

- **$message** (*resource*) – Message to convert to ICS
- **$flags** (*array*) – Reserved.

**Returns** string on success, FALSE on failure

**Return type** string

```
// Read the message as ICS.
$ics = mapi_mapitoical($session, $addrBook, $message, array());
```

### 4.13.39 mapi_icaltomapi

**mapi_icaltomapi** (*$session*, *$store*, *$addrbook*, *$message*, *$ics*, *$no_recipients*)
Converts an ICS file into a MAPI Message.

**Parameters**

- **$session** (*resource*) – Session on which the message was opened
- **$store** (*resource*) – Store the users MAPIStore
- **$addrbook** (*resource*) – Addressbook to resolve email addresses
- **$message** (*resource*) – MAPI Message in which the ICS data is converted
- **$eml** (*resource*) – ICS to convert to a MAPI Message
- **$no_recipients** (*bool*) – import recipients from ICS file.

**Returns** TRUE on success, FALSE on failure

**Return type** boolean

```
// Convert an ICS to a MAPI Message
$ok = mapi_icaltomapi($session, $store, $ab, $msg, $ics, false);
```

CHAPTER 5

# Legal Notice



Adobe, Acrobat, Acrobat Reader and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Apache is a trademark of The Apache Software Foundation.

Apple, Mac, Macintosh, Mac OS, iOS, Safari and TrueType are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Blackberry is the trademark or registered trademark of BlackBerry Limited, the exclusive rights to which are expressly reserved. Kopano is not affiliated with, endorsed, sponsored, or otherwise authorized by BlackBerry Limited.

Collax is a trademark of Collax GmbH.

Debian is a registered trademark of Software in the Public Interest, Inc.

ECMAScript is the registered trademark of Ecma International.

Gentoo is a trademark of Gentoo Foundation, Inc.

Google, Android and Google Chrome are trademarks or registered trademarks of Google Inc.

IBM and PowerPC are trademarks of International Business Machines Corporation in the United States, other countries, or both.

MariaDB is a registered trademark of MariaDB Corporation AB.

Microsoft, Microsoft Internet Explorer, the Microsoft logo, the Microsoft Internet Explorer logo, Windows, Windows Phone, Office Outlook, Office 365, Exchange, Active Directory and the Microsoft Internet Explorer interfaces are trademarks or registered trademarks of Microsoft, Inc.

Mozilla, Firefox, Mozilla Firefox, the Mozilla logo, the Mozilla Firefox logo, and the Mozilla Firefox interfaces are trademarks or registered trademarks of Mozilla Corporation.

MySQL, InnoDB, JavaScript and Oracle are registered trademarks of Oracle Corporation Inc.

NDS and eDirectory are registered trademarks of Novell, Inc.

NGINX is a registered trademark of Nginx Inc. NGINX Plus is a registered trademark of Nginx Inc.

Opera and the Opera "O" are registered trademarks or trademarks of Opera Software AS in Norway, the European Union and other countries.

Postfix is a registered trademark of Wietse Zweitze Venema.

QMAIL is a trademark of Tencent Holdings Limited.

Red Hat, Red Hat Enterprise Linux, Fedora, RHCE and the Fedora Infinity Design logo are trademarks or registered trademarks of Red Hat, Inc. in the U.S. and other countries.

SUSE, SLES, SUSE Linux Enterprise Server, openSUSE, YaST and AppArmor are registered trademarks of SUSE LLC.

# Index

## Symbols