



ADDONMAIL

MaXware
MailmaX.400 for Windows
Programmer's Guide

Edition 4.5
April 2011

This document is designed for MailmaX 400 V4.9 and newer

MaXware UA-FI, MaXware MailmaX.400 are trademarks of AddOnMail.

All other products or services mentioned in this document are identified by the trademarks or service marks of their respective companies or organizations, and AddOnMail disclaims any responsibility for specifying which marks are owned by which companies or organizations.

The information in this guide is subject to change without notice. The information does not represent a commitment on the part of AddOnMail. AddOnMail is not responsible for any errors that may appear in this manual. It is against the law to copy the documentation except when specifically permitted by in the license or non-disclosure agreement. The manual, or parts of the manual, may not be reproduced or transmitted in any form or by any means, electronically or mechanically, including photocopying and recording, for any purpose, without the express written permission of AddOnMail. No part of this publication may be transcribed, stored in a retrieval system or translated into any language without the prior written consent of AddOnMail.

Copyright © 1996-2003 MaXware AS

Copyright © 2004-2011 AddOnMail

Enquiries and orders to:

AddOnMail

16, rue Jean Jacques Rousseau

92138 Issy les Moulineaux

France

Telephone: +33 (0)1 40 83 80 90

Fax: +33 (0)1 40 83 80 99

Sales: infos@addonmail.com

Support: support@addonmail.com

Web: <http://www.addonmail.com>

Table of contents

1.	Introduction.....	i
1.1	Simple MAPI-compliant API.....	i
1.2	The Mail Spooler	i
1.3	DDE calls to MailmaX.400	i
1.4	MailmaX.400 for Windows application startup.....	i
1.5	Integration of customized Extension DLL	ii
2.	MaXware Simple MAPI functions	iii
2.1	Introduction	iii
2.2	Return values	iii
2.3	Prerequisites	iii
2.4	How to use the MaXware Simple MAPI.....	iii
2.5	MaXware Simple MAPI sessions	iv
2.6	General C data types	iv
2.7	MAPIFileDesc	v
2.8	MAPIFileTagExt	vii
2.9	MAPIMessage	ix
2.10	MAPIReceip	xi
2.11	MAPILogon.....	xiii
2.12	MAPILogoff	xv
2.13	MAPISendDocuments	xvi
2.14	MAPISendMail.....	xvii
2.15	MAPISaveMail.....	xix
2.16	MAPIFindNext	xxi
2.17	MAPIReadMail	xxiii
2.18	MAPIFreeBuffer	xxvi
2.19	MAPIDeleteMail.....	xxvii
2.20	MAPIAddress	xxviii
2.21	MAPIResolveName.....	xxx
2.22	MAPIDetails.....	xxx i
2.23	MAPIMoveMail.....	xxxii
2.24	MAPIExportArchive	xxxiii
2.25	MAPIEmptyWasteBasket.....	xxxv
2.26	GetRecipientInformation	xxxvi
2.26.1	<i>RecipientInformationDesc</i>	xxxvii
2.26.2	<i>RecipientDeliveryDesc</i>	xxxvii
2.27	GetMessageInformation	xxxix
2.27.1	<i>MessageInformationDesc</i>	xl
2.28	MAPIFreeRecipientInformation.....	xl iii
2.29	MAPIFreeMessageInformation.....	xliv
2.30	Return values and error codes.....	xl v
2.31	Sending a file using C.....	xl viii

2.32	Reading a message with C	xlix
2.33	Read unread messages with C	1
2.34	Delete messages with a certain subject	li
3.	DDE programming with MailmaX.400	lii
3.1	Introduction	lii
3.2	DDE Commands	liii
3.3	Starting MailmaX.400 for Windows	lv
3.4	C example.....	lvi
4.	Controlling the communication process	67
4.1	Introduction to the MaXware Mail Spooler	67
4.2	Architecture	67
4.3	Basic features	68
4.4	Configuring the Mailbox Commands	68
4.5	Start the Mail Spooler	69
4.6	DDE commands to the Mail Spooler.....	70
4.6.1	<i>Logon (User Name)</i>	70
4.6.2	<i>Connect(MailboxCommand)</i>	70
4.6.3	<i>IniFileChanged()</i>	70
4.6.4	<i>OUTTRAYChanged()</i>	70
4.6.5	<i>LogOff(LogOffFlag)</i>	70
4.7	DDE Services from the Mail Spooler.....	71
4.7.1	<i>State</i>	71
4.7.2	<i>Mail</i>	71
4.7.3	<i>ErrorText</i>	71
4.8	DDE System topics	72
5.	Hooks for calling an Extension DLL library	73
5.1	General	73
5.2	Routines and data structures in the Extension DLL library	74
5.3	Initialize.....	75
5.4	OnMessageOpen	77
5.5	OnMessageSave.....	79
5.6	OnSubmit	81
5.7	OnConnect	82
5.8	OnDisconnect.....	84
5.9	OnCommunicationStart	85
5.10	OnCommunicationEnd	86
5.10.1	<i>Terminate</i>	86
6.	Start program from MailmaX.400 command.....	87
7.	File viewers in MailmaX.400	88
8.	File format recognition	89
9.	UA-FI based applications with MailmaX.400	92
10.	Address syntax	95
11.	The MAXWARE.INI file	97

12.	Address and message templates	98
13.	Appendix B: MailFile - A MailmaX.400 extension.....	99
13.1	Introduction	99
13.2	Installation	100
13.3	MailFile – How it works.....	101
13.3.1	<i>Some command-line examples:</i>	101
13.3.2	<i>Syntax of the address-list file: (/File=).....</i>	102
13.4	Hints.....	103
13.4.1	<i>MailmaX.400 start-up:</i>	103
13.4.2	<i>Mail Spooler set-up</i>	103
13.4.3	<i>Run minimized.....</i>	103
13.4.4	<i>Automatic recognition of SMTP addresses.....</i>	103
13.5	Known problems	104
13.5.1	<i>Same message sent twice.....</i>	104
13.5.2	<i>Original file-name not (always) maintained.....</i>	104

1. Introduction

This guide describes the various aspects and options you have available if you wish to use MailmaX.400 for Windows to integrate X.400-based messaging into your application.

The Guide also describes how to extend MailmaX.400 with customized functionality.

The main options are described below:

1.1 Simple MAPI-compliant API

MailmaX.400 for Windows supports the MaXware Simple MAPI programming interface. This is an API compliant with Microsoft Simple MAPI that allows applications to access inbound and outbound messages in MailmaX.400. This document contains information about all implemented Simple MAPI calls, and several examples showing the use of these calls.

For more detailed information on what MAPI and Simple MAPI are, you can read the Microsoft documents on MAPI and Simple MAPI. These are distributed with Microsoft Mail, Microsoft Exchange, Microsoft Office Development Kit, and with the Microsoft Developers Network.

Note that Simple MAPI is an API that functions on a local set of messages and folders. To make the system connect to the remote messaging system, you need to program in relation to the Mail Spooler.

1.2 The Mail Spooler

The Mail Spooler is a separate module in MailmaX.400 for Windows that controls all communication (connects to the server and downloads, lists or uploads mail). The Mail Spooler can be controlled either by the end user through the MailmaX.400 Mailbox commands or by DDE calls directly from the application.

1.3 DDE calls to MailmaX.400

It is possible to submit messages and/or start connections by issuing DDE calls to MailmaX.400 for Windows.

1.4 MailmaX.400 for Windows application startup

MailmaX.400 for Windows can be configured to show the right icon and start your application automatically when a message containing your file is opened.

1.5 Integration of customized Extension DLL

It possible to include custom functionality in MailmaX.400 by creating an Extension DLL with subroutines that are called by MailmaX.400 on certain events (like the saving of a message and communication startup).

2. MaXware Simple MAPI functions

2.1 Introduction

The MaXware S-MAPI interface supports applications written in C/C++.

To avoid conflict between MaXware's MAPI.DLL module and any other MAPI.DLL from another vendor, it is recommended that C, C++ programs call the module MXMAPI32.DLL instead of MAPI.DLL. This module is always installed in the Mailmax directory, whereas any vendor may replace the MAPI.DLL that is installed in the Windows System Directory. The two modules are functionally equivalent, but have different file and module names.

2.2 Return values

All implemented calls return **SUCCESS_SUCCESS (0)** on success. On error, function-dependent values are returned. The possible return values are documented with the procedure calls that can return them.

2.3 Prerequisites

MailmaX.400 should be properly installed, and you should be able to send and receive mail using it.

For MailmaX.400, the %TEMP% environment variable has to be defined. MaXware Simple MAPI uses the %TEMP% directory as temporary storage when composing messages and extracting files.

2.4 How to use the MaXware Simple MAPI

MailmaX.400 for Windows includes two copies of the Simple-MAPI .DLL module, one named MAPI.DLL and the other named MXMAPI32.DLL. MailmaX.400 installs MAPI.DLL to the Windows System directory, and MXMAPI.DLL/MXMAPI32.DLL to the MailmaX.400 directory. The user can choose not to install MAPI.DLL. Many E-mail and communication programs install their own MAPI.DLL to the Windows System directory. These programs include Microsoft Mail, Microsoft Windows and other applications supporting MAPI or Simple MAPI. Therefore you should use the MXMAPI.DLL/MXMAPI32.DLL in the MailmaX.400 directory to avoid the problems that arise if other programs overwrite the MaXware MAPI.DLL. If you have definition files from MaXware, MXMAPI.H defines how to access MXMAPI.DLL/MXMAPI32.DLL and also defines MaXware extensions to MAPI, whereas MAPI.H defines how to access Microsoft compliant routines in MAPI.DLL.

2.5 MaXware Simple MAPI sessions

All Simple MAPI calls are done in the context of a *session*, and the following section explains how a session is obtained.

Before the first MailmaX.400 or MAPI logon there is no existing valid MaXware Simple MAPI session. As result of a successful logon, the user's LMS (Local Message Store) is opened and a valid session is obtained. Any subsequent logon process can do either of the following with respect to logon credentials (user name and password):

- Not supply logon credentials (try to access the default session). In this case the MAPI module will return a handle to an existing session if this is possible (MailmaX.400 or another program based on Simple MAPI is already logged on).
- Supply logon credentials (explicit logon). In this case the messaging subsystem validates the name and password given, and logs on to a new user's LMS. This behavior is not dependent on flag settings - in MaXware Simple MAPI, all sessions behave like SHARED.

2.6 General C data types

The following definitions are used in all C struct definitions, proc definitions and examples through this document:

```
typedef unsigned long          ULONG;  
typedef unsigned long FAR *   LPULONG;  
typedef unsigned long         FLAGS;  
typedef unsigned long         LHANDLE  
typedef unsigned long FAR *   LPLHANDLE;  
typedef unsigned long FAR *   LPSTR;  
typedef unsigned void FAR *   LPVOID;  
typedef unsigned long         ULRESERVED;
```

2.7 MAPIFileDesc

MAPIFileDesc contains information about a file to be attached to a message.

C and C++

```
typedef struct {
    ULONG          ulReserved;      // Reserved for future use (must be 0)
    ULONG          flFlags;         // Flags
    ULONG          nPosition;       // Attachment position in text
    LPSTR          lpszPathName;    // Full path name of attachment file
    LPSTR          lpszFileName;    // Original file name (optional)
    LPVOID         lpFileType;      // Attachment file type (optional)
} MapiFileDesc, far *lpMapiFileDesc;
```

Note: MaXware Simple MAPI does not support OLE attachments.

Attributes:

Reserved

Reserved. Must be zero.

Flags

Bit map of flags. Unused flags are reserved and must be zero.

Can be set to `MAPI_ATTACH_BINARY (&H8000000)` to force binary type for attachment when submitting or saving messages using **MAPISendMail** or **MAPISaveMail**.

Position

Position of attachment in text note.

Note: In MaXware Simple MAPI, this value is ignored and will always be set to -1 by the MAPI routines.

PathName

The full path and file name of the attachment file. This name should include disk volume and directory names.

The file must be closed by your application before it is handed to Simple MAPI.

FileName

The file name of the attachment as used by the end user.

With the MaXware Simple MAPI it is possible to create messages containing a Message BodyPart (forwarding). The following MaXware specific syntax must be used for the value of the *FileName*:

```
%FORWARDEDBP=<MessageID for the message to be forwarded>
```

The message to be forwarded must exist in the Local Message Store.

To get the Message ID of any forwarded message when reading a message, the MAPI_RETURN_FORWARD_ID flag must be set in the Flags parameter to MAPIReadMail.

If the flag is not set, MaXware Simple MAPI will process all forwarded messages recursively, and list all found BodyParts as if they were BodyParts to the outermost message.

FileType

Pointer to a MAPIFileTagExt structure describing the file format (type) of the attachment and the X.400 BodyPart encoding to be used. This structure is described in the next section.

2.8 MAPIFileTagExt

The **MAPIFileTagExt** structure specifies the format and type of an attached file, i.e. if it is a MS Word file, an Excel file, etc. It also specifies how to encode the BodyPart (the attachment) in an X.400 message (Text, FTAM, Bilaterally Defined etc.).

The syntax of the **MapiFileTagExt** structure is as follows:

```
typedef struct {
    ULONG          ulReserved;
    ULONG          cbTag;
    LPBYTE         lpTag;
    ULONG          cbEncoding;
    LPBYTE         lpEncoding;
} MapiFileTagExt, Far *lpMapiFileTagExt;
```

Attributes:

Reserved

Reserved. Must be zero.

cbTag

Size, in bytes, of the value of the Tag Attribute

lpTag

Pointer to an X.400 Object Identifier defining the format and type of the BodyPart/attachment in its original form, such as MS Word or MS Excel. If the attachment is encoded as an FTAM BodyPart, the Object Identifier is sent in the FTAM parameter "Application Reference". Possible values for the Tag attribute appear in the OID column of the file DOCMAGIC. in the MailmaX.400 directory.

cbEncoding

Size, in bytes, of the value of the Encoding Attribute

IpEncoding

Pointer to an Object Identifier defining the X.400 BodyPart encoding of the file. The Object identifiers are defined in X.420 and are listed in the table below:

MaXware BodyPart Type name	X.400 BodyPart Type	The extended BodyPart type names as defined in the X.420 standard	IpEncoding ObjectID
Data	Bilaterally defined	id-et-bilaterally-defined	2.6.1.4.9
File	FTAM	id-et-file-transfer Note: This causes the message to be encoded as P22.	2.6.1.4.12
IA5	IA5	id-et-ia5-text	2.6.1.4.0
Teletex	Teletext (T.61)	id-et-teletex	2.6.1.4.4
GeneralText (ISO 8859-x character sets)	GeneralText	id-et-general-text Note: This causes the message to be encoded as P22.	2.6.1.4.11
G3Fax (incoming only)	G3Facsimile	id-et-g3-facsimile	2.6.1.4.2
IPM	Forwarded message	id-et-message	2.6.1.4.7

If an unknown ObjectID is specified by the application, S-MAPI will store and handle the message as type DATA. Incoming messages with unknown or undefined X.400 BodyPart encoding are represented and handled as DATA.

2.9 MAPIMessage

MAPIMessage contains information about a message.

C and C++

```
typedef struct {
    ULONG                ulReserved;
    LPSTR                lpszSubject;
    LPSTR                lpszNoteText;
    LPSTR                lpszMessageType;
    LPSTR                lpszDateReceived;
    LPSTR                lpszConversationID;
    FLAGS                flFlags;
    IpMapiRecipDesc      lpOriginator;
    ULONG                nRecipCount;
    IpMapiRecipDesc      lpRecips;
    ULONG                nFileCount;
    IpMapiFileDesc       lpFiles;
} MapiMessage, far *MapiMessage;

#define MAPI_UNREAD                0x00000001
#define MAPI_RECEIPT_REQUESTED    0x00000002
#define MAPI_SENT                  0x00000004
#define MAPI_MSG_FAILED            0x08000000
```

Attributes:

Reserved

Reserved. Must be zero.

Subject

Pointer to the text string describing the message subject, max. 256 characters. A pointer value of NULL or empty string indicates no subject text.

NoteText

Pointer to a string containing the message text (the text note which is the first BodyPart in the X.400 message). A pointer value of NULL or empty string indicates no text (either no BodyParts, or that the first BodyPart is binary).

MessageType

Pointer to a string indicating the message class. The type is for use by applications other than interpersonal mail.

MaXware Simple MAPI simulates the MAPI Message type concept by putting “<<message type>>” at the end of the subject when a message is created, and detecting the message type in the subject when a message is received. This means that the Message Type concept will work when both the sender and the recipient use MaXware Simple MAPI, if both parties use Microsoft Simple MAPI, but not when one of the applications uses Microsoft Simple MAPI and the other uses the MaXware Simple MAPI implementation.

DateReceived

Pointer to a string indicating the date the message was received. The format is “YYYY/MM/DD HH:MM” using a 24-hour clock.

Note that the MailmaX.400 folder shows the submission date for messages, and not the date received.

Note also that the date and time format does not follow the (local) Windows date and time format.

ConversationID

Ignored in this version of MaXware Simple MAPI.

Flags

Bitmap of message flags. Unused flags are reserved and must be zero for outbound messages, and should be ignored for inbound messages.

MAPI_UNREAD on an inbound message means that the message has not been read.

MAPI_UNREAD is set on an outbound message when saved, and removed when one receipt notification is received for each of the recipients of the message (the message status in the MailmaX.400 OUTTRAY).

MAPI_SENT is set on an outbound message as soon as the message has been submitted (transferred) to the X.400 message store.

MAPI_MSG_FAILED is a MaXware extension that will be set on an outbound message if the message has the state "FAILED" in the MailmaX.400 OUTTRAY.

MAPI_RECEIPT_REQUESTED can be set on an outbound message to request receipt notifications from all recipients.

Originator

Pointer to a MAPIRecep structure describing the originator of the message.

RecipCount

The number of message recipient structures that the lpRecips Attribute points to. The number of recipient descriptors in the message's envelope is stored in this location. A value of 0 indicates that no recipients have been defined.

Recips

Pointer to an array of MAPIRecep structures describing the recipients of the message.

FileCount

The number of file attachment descriptors that lpFiles points to. A value of 0 indicates that no file attachments have been defined.

Files

Pointer to an array of MAPIFileDesc structures, each holding information about a file attachment.

2.10 MAPIRecip

A **MAPIRecip** structure holds information about a message sender or recipient.

C and C++

```
typedef struct {
    ULONG          ulReserved
    ULONG          ulRecipClass
    PSTR           lpszName
    LPSTR          lpszAddress
    ULONG          ulEIDSize
    LPVOID         lpEntryID
} MapiRecipDesc, far lpMapiRecipDesc;

#define MAPI_ORIG 0           // Recipient is message originator
#define MAPI_TO 1           // Recipient is a primary recipient
#define MAPI_CC 2           // Recipient is a copy recipient
#define MAPI_BCC 3         // Recipient is blind copy recipient
```

Attributes:

Reserved

Reserved. Must be zero.

RecipClass

Numeric value specifying the type of recipient (0:Originator, 1:To, 2:CC, 3:BCC). Also used to describe the originator of a received message.

Name

The display (“friendly”) name of the recipient. This may be the FreeForm name in the X.400 address for an inbound message. When creating a recipient, you can fill in the recipient’s name in this field. If the address field is empty, the MaXware MAPI will look for a legal address (see below) in the Name field..

Address

The address of a recipient. The address must be in one of the following formats:

MailmaX.400 short name (for list or person in the Address Book)

- X.400 O/R-name in the form: g=John;s=Jones... as defined in F.400 (1992) and Chapter 10 in this manual.
- "X.400:g=John;s=Jones..."
- The O/R name may be prefixed by the string "X.400:."; this string is ignored.
- Internet format in the form: xxxx@yyyy.zzz
- <Template name>;parameter 1/parameter 2...
where the template name is the name of an address template defined in the MailmaX.400 file <Profile>.FRM (fax, telex, pager, Internet, etc.)
- A comma-delimited list of the above.

EIDSize

The size in bytes of the entry identifier value of the lpEntryID Attribute.

EntryID

Binary data used by the messaging system to specify and handle the recipient efficiently. Unlike the lpszAddress Attribute, this data is opaque and may not be printable. The messaging system returns valid lpEntryIDs for recipients and originators that are defined in the Address Book.

2.11 MAPILogon

MAPILogon starts and initializes a Simple-MAPI session.

C and C++

```

ULONG MAPILogon (
    ULONG          UIParam,
    LPSTR          ProfileName,
    LPSTR          Password,
    ULONG          Flags,
    ULONG          Reserved,
    LPLHANDLE      Session)

#define MAPI_LOGON_UI          0x00000001
#define MAPI_NEW_SESSION      0x00000002
#define MAPI_ALLOW_OTHERS     0x00000008
#define MAPI_EXPLICIT_PROFILE 0x00000010
#define MAPI_USE_DEFAULT      0x00000040
// MaXware extension:
#define MAPI_MAILMAX          0x10000000
#define MAPI_DELETE_TO_WASTEBASKET 0x08000000

```

UIParam

If **MAPI_LOGON_UI** is set in the flFlags word, this parameter must contain the handle for the parent window of type HWND.

ProfileName

User's logon name. If given, logon applies to the LMS of this user. If NULL, the behavior is controlled by flFlags. See below.

Password

The user's password. This value is required for successful logon if the **MAPI_LOGON_UI** is not set.

Flags

If **MAPI_LOGON_UI** is set, the dialog box UI is presented when necessary.

MAPI_ALLOW_OTHERS makes it possible for other applications to perform implicit logon (a MAPILogon call where lpszProfileName and lpszPassword are NULL) and get a valid session handle to this session.

If **MAPI_NEW_SESSION** is set, a new session will always be opened.

If **MAPI_EXPLICIT_PROFILE** is not set, the messaging subsystem supplies logon credentials by itself.

To enable MaXware extensions to Simple MAPI, the flag **MAPI_MAILMAX** must be set. The extensions enabled by this flag are documented with the calls and structs they extend.

Reserved

Should always be 0.

Session

A handle to a MAPI Session structure. MAPILogon initializes this handle and the corresponding structure, and the handle must be given as input to all subsequent Simple MAPI calls.

Return values

- MAPI_E_FAILURE** If ulReserved was not 0.
- MAPI_E_LOGON_FAILURE:** If the session handle was not valid, or if ProfileName is empty and the combination of flags in the Flags parameter is not correct or if some other error occurs during the logon process.
- MAPI_E_INSUFFICIENT_MEMORY**
if an error occurs while opening LMS.
- MAPI_USER_ABORT** if the user presses Cancel in the logon dialog.

2.12 MAPILogoff

MAPILogoff terminates a Simple-MAPI session.

C and C++

```

ULONG MAPILogoff (
    LHANDLE      Session,
    ULONG        UIParams,
    FLAGS        Flags,
    ULONG        Reserved)

#define MAPI_LOGOFF_SHARED      0x00000001
#define MAPI_LOGOFF_UI         0x00000002

```

Session

A handle for the current session, obtained by calling **MAPILogon**.

UIParam

Handle of parent window. This parameter is not used because no dialog box is displayed during logoff.

Flags

MAPI_LOGOFF_UI is not used by the MaXware Simple MAPI module.

To terminate the session completely, **MAPI_LOGOFF_SHARED** must be set.

Reserved

Should always be 0.

Return values

MAPI_E_FAILURE if Reserved != 0.

or Session == NULL or error while logging out. False, return.

MAPI_E_INVALID_SESSION if Session == 0.

NOTE: The routine may fail (general protection fault) if the Session parameter is larger than 0 and does not point to a valid session.

SUCCESS_SUCCESS if the logoff operation was successful.

2.13 MAPISendDocuments

MAPISendDocuments is used to compose a message with one or more files attached and a cover note. The MailmaX.400 Compose window is displayed to let the user add recipients and other information to the message.

Note that this call has no complex parameters - it is particularly useful when working with macro languages and other languages without complex data types.

C and C++

```
ULONG MAPISendDocuments(  
    ULONG          UIParam,  
    LPSTR          DelimChar,  
    LPSTR          FilePaths,  
    LPSTR          FileNames,  
    ULONG          Reserved)
```

UIParam

Parent window handle; 0 is always valid.

DelimChar

Delimiter character used between file names in the FilePaths and FileNames strings. The strings should be terminated by \0.

FilePaths

A set of path strings for the files to be attached (including file names).

FileNames

Names of the files to be attached (no paths, used for display only).
Use 8.3 format (filename.txt).

Reserved

Always 0.

2.14 MAPISendMail

MAPISendMail is used to send mail using MailmaX.400. Note that the message sent is not transferred to the mail server; it is just saved in the OUTTRAY folder with the status "Ready". Place all information about the message to be sent in the **Message** parameter (**MAPIMessage** structure/type).

To get the MailmaX.400 Mail Spooler to connect and send documents that are in the OUTTRAY, the application must issue DDE commands to the spooler.

C and C++

```

ULONG MAPISendMail (
    LHANDLE          Session,
    ULONG           UIParam,
    lpMapiMessage   Message,
    FLAGS           Flags,
    ULONG           Reserved)

#define MAPI_DIALOG          0x00000008
#define MAPI_LOGON_UI       0x00000001

```

Session

Session struct, obtained by MAPILogon. If this is set to 0, implicit logon will be performed. This will only succeed if a valid session with MAPI_ALLOW_SHARED exists, or the MAPI_LOGON_UI is set.

UIParam

ULONG, Handle to the parent window.

Message

Pointer to message struct containing the message to be sent.

Recipient()

An array of recipients. When RecipCount is 0, this parameter is ignored. The recipient string can include either the recipient's name or the recipient's name-address pair. If just a name is specified, the name is resolved to an address using implementation-defined address book search rules. If an address is also specified, no search for the name is performed. The address is in an implementation-defined format, and is assumed to have been obtained from the implementation in some other way. When the address is specified, the name is used for display to the user and the address is used for delivery. When EntryID is used, no search is performed and the display-name and address are ignored. (A name and address are associated with the EntryID within Mail.) EntryIDs are returned by the **MAPIReadMail** function.

File()

An array of attachment files written when the message is read. The number of attachments per message may be limited in some systems. If the limit is exceeded, the error MAPI_E_TOO_MANY_FILES is returned. When FileCount is 0, this parameter is ignored.

Attachment files are read and attached to the message before the call returns. Do not attempt to display attachments outside the range of the message body.

Flags

MAPI_DIALOG causes MAPI to display a Compose window with information on the message, and lets the user edit and add recipients, attachments, etc..

MAPI_LOGON_UI must be set if the function should display a dialog box to prompt the user for user name and password.

Reserved

Should always be 0.

Return values

MAPI_E_FAILURE if ulReserved != 0, and in all error situations except those listed below.

MAPI_E_INVALID_SESSION if hSession < 0L (this call allows hSession == 0, i.e. implicit logon is allowed)

2.15 MAPISaveMail

MAPISaveMail is used to save a new message or to replace an existing message. The message is saved in the OUTTRAY folder with the status “Draft”.

Note that the Microsoft Simple MAPI module saves draft messages in the INTRAY, where they may be found later by using the **MAPIFindNext** call. This is not possible with MaXware Simple MAPI, but a saved message may later be fetched using the message identifier returned by **MAPISaveMail** as a parameter to **MAPIReadMail**.

C and C++

```
ULONG MAPISaveMail (
    LHANDLE      Session,
    ULONG        UIParam,
    IpMapiMessage Message,
    FLAGS        Flags,
    ULONG        Reserved,
    LPSTR        MessageID)

#define MAPI_LOGON_UI      0x00000001
```

Session

Session struct, obtained by **MAPILogon**. If this is set to 0, implicit logon will be performed. This will only succeed if a valid session with **MAPI_ALLOW_SHARED** exists, or the **MAPI_LOGON_UI** is set.

UIParam

Handle of the parent window.

Message

Pointer to message struct containing the message to be saved.

Recipient()

An array of recipients. When **RecipCount** is 0, this parameter is ignored. The recipient string can include either the recipient’s name or the recipient’s name-address pair. If just a name is specified, the name is resolved to an address using implementation-defined address book search rules. If an address is also specified, no search for the name is performed. The address is in an implementation-defined format, and is assumed to have been obtained from the implementation in some other way. When the address is specified, the name is used for display to the user, and the address is used for delivery. When **EntryID** is used, no search is performed, and the display-name and address are ignored. (A name and address are associated with the **EntryID** within Mail.) **EntryIDs** are returned by the **MAPIReadMail** function.

File()

An array of attachment files written when the message is read. The number of attachments per message may be limited in some systems. If the limit is exceeded, the error **MAPI_E_TOO_MANY_FILES** is returned. When **FileCount** is 0, this parameter is ignored.

Attachment files are read and attached to the message before the call returns. Do not attempt to display attachments outside the range of the message body.

Flags

Controls this function's behavior.

MAPI_LOGON_UI must be set if the function should display a dialog box to prompt the user for user name and password.

Reserved

Should always be 0.

MessageID

If not an empty string, this parameter gives the ID of an existing message that is overwritten. If empty, the message is saved as a new message, and a new MessageID is returned in this parameter.

Return values

MAPI_E_FAILURE if ulReserved != 0, and in all other cases except those listed below.

MAPI_E_INVALID_SESSION if hSession < 0L

(this call allows hSession == 0, i.e. implicit logon allowed)

2.16 MAPIFindNext

To read mail from the MailmaX.400 INTRAY Folder with the MaXware Simple MAPI, you need to obtain the MessageID of the message that you want to read. To do this, you must use the **MAPIFindNext** call. After you have read the message and processed the information returned, you should free the space allocated by the messaging subsystem.

MAPIFindNext obtains a valid MessageID. The ID can later be given as parameter to other Simple MAPI calls for reading or deleting messages, or as a seed parameter to a new MAPIFindNext call.

You can choose messages from the pool of ALL messages in the INTRAY Folder, or just messages with status “Unread”. The flag setting determines the folder to be searched for messages.

Note: With MaXware Simple MAPI it is also possible to read messages from the MailmaX OUTTRAY Folder. To do this, set the flag MAPI_SEARCH_OUTTRAY.

Messages may also be filtered on the Message Type.

C and C++

```

ULONG MAPIFindNext(
    LHANDLE      Session,
    ILONG        UIParam,
    LPSTR        MessageType,
    LPSTR        SeedMessageID,
    FLAGS        Flags,
    ULONG        Reserved,
    LPSTR        MessageID)

#define MAPI_UNREAD_ONLY          0x00000020
#define MAPI_GUARANTEE_FIFO      0x00000100
#define MAPI_LOGON_UI             0x00000001
// MaXware extension:
#define MAPI_SEARCH_OUTTRAY      0x00000200

```

Session

Session handle, obtained by MAPILogon. No implicit logon.

UIParam

Handle of parent window

MessageType

MAPI Message type. If this is NULL or “”, all messages will be found. If this string is not “”, only messages mailed with this string in the MessageType field in the Message struct will be found. (See additional comments on the MaXware implementation of MessageTypes under the description of the Message struct).

SeedMessageID

If not an empty string, this parameter is the seed MessageID, i.e. the ID of the last fetched message. The ID of the returned message will be lower than this number. (In MaXware, this means that the message is newer than the “seed” message). If an empty string is given, the ID of the first (oldest) message that satisfies the given flag setting and message type is returned.

Flags

MAPI_UNREAD_ONLY forces the call to search among UNREAD messages only.

Note:

There is no direct way to specify read messages only. One way to implement this could be to use FindNext for all messages (without using **MAPI_UNREAD_ONLY**), read each message with MAPI_PEEK and MAPI_ENVELOPE_ONLY (see under MAPIReadMail for these flags), and test the message structure's Flags value against **MAPI_UNREAD**. If the message is unread, use the returned MessageID as seed for a new call to MAPIFindNext, until you get a read message or the whole folder is traversed.

MAPI_GUARANTEE_FIFO ensures that the message pool is traversed in the FIFO order. As MailmaX.400 always uses FIFO, this flag is not necessary (and ignored) for MaXware Simple MAPI.

MaXware Simple MAPI extension:

If the **MAPI_SEARCH_OUTTRAY** flag is set, **MAPIFindNext** will search the messages in the OUTTRAY Folder.

Reserved

Should always be 0.

MessageID

Returned message ID. Should be used in subsequent calls to MAPIFindNext / MAPIDeleteMail / MAPIReadMail, etc.

Message IDs may be invalidated at any time if another application deletes or moves a message.

Return values:

MAPI_E_FAILURE if Reserved != 0, and in all other error conditions not listed below.

MAPI_E_INVALID_SESSION if Session = 0.

MAPI_E_NO_MESSAGES if there are no more messages in the folder that match flag settings given in the Flags.

2.17 MAPIReadMail

To read mail from the MailmaX.400 INTRAY Folder with MaXware Simple MAPI, you need to obtain the MessageID of the message that you want to read. To do this, you must use the **MAPIFindNext** call. After you have read the message and processed the information returned, you should free the space allocated by the messaging subsystem (C / C++).

MAPIReadMail fetches the message with the given MessageID and returns the message as a **MAPIMessage** structure/type. A valid MessageID is obtained by the **MAPIFindNext** call.

Note that some flags in the returned Message struct's flag word have special meanings when MAPILogon is performed with the flag MAPI_MAILMAX set and the message is read from the MailmaX.400 OUTTRAY Folder.

C and C++

```

ULONG MAPIReadMail (
    LHANDLE          Session,
    ULONG            UIParam,
    LPSTR            MessageID,
    FLAGS            Flags,
    ULONG            Reserved,
    LpMapiMessage FAR * Message)

#define MAPI_ENVELOPE_ONLY 0x00000040
#define MAPI_PEEK          0x00000080
#define MAPI_BODY_AS_FILE 0x00000200
#define MAPI_SUPPRESS_ATTACH 0x00000800
#define MAPI_LOGON_UI      0x00000001
// MaXware extension:
#define MAPI_RETURN_FORWARD_ID 0x2000000

```

This function reads a mail message. You should use **MAPILogon** to establish a valid MAPI session, and then use **MAPIFindNext** before calling MAPIReadMail to verify that the message to be read is the one desired.

The call returns one message, breaking the message content into the same parameters and types used in the **MAPISendMail** function. MAPIReadMail fills a block of memory with the MapiMessage type containing message elements. File attachments are saved to temporary files, and the names are returned to the caller in the message type. Recipients, attachments, and contents are copied from the message before the function returns to the caller, so that later changes to the files do not affect the contents of the message.

There is a flag for specifying that only envelope information is to be returned from the call. Another flag (in the MapiMessage type) specifies whether the message is marked as sent or unsent.

All strings are null-terminated, and must be specified in the current character set or code page of the calling program's operating system process. In Microsoft Windows, the character set is ANSI.

Parameters:

Session

An opaque session handle whose value represents a session with the messaging subsystem. The session handle is returned by **MAPILogon** and invalidated by **MAPILogoff**.

UIParam

The parent window handle for the dialog box. A value of 0 specifies that any dialog box displayed is application modal.

MsgID

Mail's string identifier for this message returned by **MAPIFindNext** or **MAPISaveMail**.

Flags

A bitmask of flags. Unspecified flags should always be passed as 0. Undocumented flags are reserved. The following flags are defined.

When you set **MAPI_ENVELOPE_ONLY** (&H40), the function doesn't copy attachments to temporary files or return the note text. All other message information (except for temporary filenames) is returned. Setting this flag usually reduces the processing time required for the function.

When you set **MAPI_SUPPRESS_ATTACH** (&H800), **MAPIReadMail** doesn't copy attachments but returns note text. If **MAPI_ONLY_ENVELOPE** is set, this flag is ignored. The flag should reduce the time required by the **MAPIReadMail** function.

When you set **MAPI_BODY_AS_FILE** (&H200), the message body is written to a temporary file and added to the attachment list as the first attachment, instead of returning a pointer to the message body (the default behavior). The Position parameter of a body attachment is -1.

When you set **MAPI_PEEK** (&H80), **MAPIReadMail** does not mark the message as read. Any unsuccessful return leaves the message unread.

Reserved

Reserved for future use. This parameter must be 0.

Message

A type set by **MAPIReadMail** to a message containing the message contents.

Originator

The originator of the message.

Recipients()

An array of recipients. This array will be redimensioned as necessary to accommodate the number of recipients chosen by the user.

Files()

An array of attachment files written when the message is read. When `MAPIReadMail` is called, all message attachments are written to temporary files. It is the caller's responsibility to delete these files when they are no longer needed. When `MAPI_ENVELOPE_ONLY` or `MAPI_SUPPRESS_ATTACH` is set, no temporary files are written and no temporary names are filled into the file attachment descriptors. This array will be redimensioned as necessary to accommodate the number of files attached by the user.

Return values:

MAPI_E_FAILURE if `Reserved` \neq 0, and in all other error conditions not listed below.

MAPI_E_INVALID_SESSION if `Session` = 0.

MAPI_E_INVALID_MESSAGE if the message is encrypted and can't be decrypted

2.18 MAPIFreeBuffer

Frees memory allocated by the **MAPIReadMail**, **MAPIAddress** or **MAPIResolveName** calls.

C and C++

ULONG MAPIFreeBuffer (lpMapiMessage pMessage)

pMessage

A pointer to a message record to be freed. All recipient records, file attachment records, the originator record and all dynamically allocated space are freed. On success, pMessage is set to NULL.

2.19 MAPIDeleteMail

MAPIDeleteMail deletes the message with the given MessageID. To obtain a valid message ID, you can use the **MAPIFindNext** call.

C and C++

```

ULONG MAPIDeleteMail (
    LPHANDLE      Session,
    ULONG         UIParam,
    LPSTR         MessageID,
    FLAGS         Flags,
    ULONG         Reserved)

// MaXware extension:
#define MAPI_DELETE_TO_WASTEBASKET    0x08000000

```

Session

Session struct, obtained by **MAPILogon**.

UIParam

Parent window handle.

MessageID

ID of the message to be deleted.

Flags

If the MaXware extension flag **MAPI_DELETE_TO_WASTEBASKET** is set, the message is not totally deleted. It is only moved to the WASTEBASKET folder (as with all message deletion in MailmaX.400 for Windows). To avoid filling up the WASTEBASKET with old messages, the default action of **MAPIDeleteMail** is to delete the message completely.

Reserved

Should always be 0.

Return values:

Returns **MAPI_E_FAILURE** if `ulReserved != 0` and for all other error conditions except those listed below.

Returns **MAPI_E_INVALID_SESSION** if `hSession <= 0L`.

Returns **MAPI_E_INVALID_MESSAGE** if an error occurs during the deletion of a message (probably because the MessageID given is invalid).

2.20 MAPIAddress

MAPIAddress is to create or modify the set of recipients defined for a message. If a user interface is displayed, this is the same as for the Address Book functions of MailmaX.400.

C and C++

```

ULONG MAPIAddress(
    LHANDLE          Session,
    ULONG            UIParam,
    LPSTR            Caption,
    ULONG            nEditFields,
    LPSTR            Labels,
    ULONG            nRecips,
    IpMapiRecipDesc Recips
    FLAGS            Flags,
    ULONG            Reserved,
    LPULONG          nNewRecips
    IpMapiRecipDesc FAR *NewRecips)

#define MAPI_AB_NOMODIFY    0x00000400

```

Session

If this is set to 0, implicit logon will be performed. This will only succeed if a valid session with `MAPI_ALLOW_SHARED` exists, or the `MAPI_LOGON_UI` is set.

UIParameter

Parent windows handle.

Caption

Caption for the address book dialog.

nEditFields

The number of fields with recipient lists -

- 1: To (primary recipient) field only
- 2: To and CC (Carbon Copy) fields
- 3: To, CC and BCC (Blind Carbon Copy) fields.

Label

Label used on the 'To' field if nEditFields is 1 and Label is not empty.

nRecips

The number of recipients in Recips.

Recips

An array of recipients.

fFlags

MAPI_LOGON_UI: Show dialog if necessary to let the user log on.

MAPI_AB_NOMODIFY: The MaXware Simple MAPI does not support this flag - the address book(s) are always read-only.

Reserved

Should always be 0.

nNewRecips

The number of recipients added by this call to MAPIAddress.

NewRecips

An array of recipient records:

- in: currently defined recipients
- out: result after the user has added / deleted some recipients.

2.21 MAPIResolveName

MAPIResolveName converts a user-friendly name (short name in the Address Book) to a complete X.400 mail address, optionally by prompting the end user for information.

C and C++

```
ULONG MAPIResolveName(  
    LHANDLE          IhSession,  
    ULONG            UIPParameter,  
    LPSTR            Name,  
    FLAGS            Flags,  
    ULONG            Reserved,  
    IpMapiRecipDesc FAR * Recip)  
  
#define MAPI_AB_NOMODIFY    0x00000400
```

Session

If this is set to 0, implicit logon will be performed. This will only succeed if a valid session with **MAPI_ALLOW_SHARED** exists, or the **MAPI_LOGON_UI** is set.

UIParameter

Parent window handle.

Name

Name to be resolved.

Flags

MAPI_LOGON_UI: Show dialog if necessary to let the user log on.

MAPI_DIALOG: Let the user manually resolve names by displaying a list of recipients.

MAPI_AB_NOMODIFY: Not supported by the MaXware Simple MAPI (the address book is always read-only).

Reserved

Always 0.

Recip

List of currently defined recipients (in and out parameter).

2.22 MAPIDetails

MAPIDetails presents a dialog with the details of an Address Book entry.

C and C++

```
ULONG MAPIDetails(  
    ULONG          Session,  
    ULONG          UIParam,  
    IpMapiRecipDesc Recip,  
    FLAGS          Flags,  
    ULONG          Reserved)  
  
#define MAPI_AB_NOMODIFY    0x00000400
```

Session

If this is set to 0, implicit logon will be performed. This will only succeed if a valid session with MAPI_ALLOW_SHARED exists, or the MAPI_LOGON_UI is set.

UIParam

Parent window handle.

Recip

Recipient record with the information to be presented to the user.

Flags

MAPI_AB_NOMODIFY: Not supported by the MaXware Simple MAPI; the address book(s) are always read-only.

Reserved

Should always be 0.

2.23 MAPIMoveMail

MAPIMoveMail move a message from a folder to another folder.

C and C++

```

ULONG MAPIMoveMail(
    ULONG          Session,
    ULONG          UIParam,
    LPSTR          MessageID,
    LPSTR          IpszTarget,
    FLAGS          Flags,
    ULONG          Reserved)

#define MAPI_ LOGON_UI      0x00000001

```

Session

If this is set to 0, implicit logon will be performed. This will only succeed if a valid session with MAPI_ALLOW_SHARED exists, or the MAPI_LOGON_UI is set.

UIParam

Parent window handle.

MessageID

Give the ID of the message to move.

IpszTarget

Give the ID of the destination folder.

Flags

MAPI_LOGON_UI must be set if the function should display a dialog box to prompt the user for user name and password.

If the **MAPI_SEARCH_OUTTRAY** flag is set, MAPIMoveMail will search the message in the OUTTRAY folder.

Reserved

Should always be 0.

Return values:

Returns **MAPI_E_FAILURE** if ulReserved != 0 and for all other error conditions except those listed below.

Returns **MAPI_E_INVALID_SESSION** if hSession <= 0L (this call allows hSession==0, i.e. implicit logon is allowed)

Returns **MAPI_E_INVALID_MESSAGE** if an error occurs during the move of a message (probably because the MessageID given is invalid).

Returns **MAPI_E_INVALID_FOLDER** if an error occurs during the move of the message (probably because the IpszTarget given is invalid).

2.24 MAPIExportArchive

MAPIExportArchive copies all message in the “Archive” system folder to a specific directory with standard LMS format. Optional this LMS structure will be compressed in a ZIP file.

C and C++

```

ULONG MAPIExportArchive (
    ULONG          Session,
    ULONG          UIParam,
    LPSTR          IpszTarget,
    LPSTR          IpszArchiveName,
    ULONG          DeleteMessages,
    BOOL           bDecryptAttachments
    ULONG          UseCompression,
    INT4           *NumberExported,
    FLAGS          Flags,
    ULONG          Reserved)

```

```
#define MAPI_ LOGON_UI      0x00000001
```

Session

If this is set to 0, implicit logon will be performed. This will only succeed if a valid session with MAPI_ALLOW_SHARED exists, or the MAPI_LOGON_UI is set.

UIParam

Parent window handle.

IpszTarget

Give the path of the destination directory.

IpszArchiveName

Give the path of the archive directory. If this is set to 0, generate automatically the archive name according to the format.

DeleteMessages

Delete messages in the “Archive” system folder after export. If set to 1, delete messages after export.

bDecryptAttachments

Allow to archive decrypted attachments

If set to false, attachments are not decrypted (if they are encrypted)

If set to true, encrypted attachments are decrypted in the archive LMS.

UseCompression

Compress the archive directory after export. If set to 1, the compression is used.

NumberExported

Return the number of mails exported by the archive mechanism.

Flags

MAPI_LOGON_UI must be set if the function should display a dialog box to prompt the user for user name and password.

Reserved

Should always be 0.

Return values:

Returns **MAPI_E_FAILURE** if Reserved != 0 and for all other error conditions except those listed below.

Returns **MAPI_E_INVALID_SESSION** if hSession <= 0L (this call allows hSession==0, i.e. implicit logon is allowed)

Returns **MAPI_E_INVALID_DIRECTORY** if an error occurs during the move of the message (probably because the lpszTarget or lpszArchiveName given is invalid).

2.25 MAPIEmptyWasteBasket

MAPIEmptyWasteBasket empty the waste basket folder.

C and C++

```
ULONG MAPIEmptyWasteBasket (  
    ULONG Session,  
    ULONG UIParam,  
    FLAGS Flags,  
    ULONG Reserved)
```

Session

If this is set to 0, implicit logon will be performed. This will only succeed if a valid session with MAPI_ALLOW_SHARED exists, or the MAPI_LOGON_UI is set.

UIParam

Parent window handle.

Flags

MAPI_LOGON_UI must be set if the function should display a dialog box to prompt the user for user name and password.

Reserved

Should always be 0.

Return values:

Returns **MAPI_E_FAILURE** if Reserved != 0 and for all other error conditions except those listed below.

Returns **MAPI_E_INVALID_SESSION** if hSession <= 0L (this call allows hSession==0, i.e. implicit logon is allowed)

2.26 GetRecipientInformation

GetRecipientInformation retrieved a list of recipients with their details.

C and C++

```
ULONG GetRecipientInformation(  
    LPHANDLE Session,  
    LPSTR MessageID,  
    ULONG *nRecipientCount;  
    lpRecipientInformationDesc* RecipientInfo);
```

Session

An opaque session handles whose value represents a session with the messaging subsystem. The session handle is returned by MAPILogon and invalidated by MAPILogoff.

MessageID

Mail's string identifier for this message returned by MAPIFindNext or MAPISaveMail.

nRecipientCount

The number of recipient structures that the RecipientInfo attribute points to.

RecipientInfo

Point to an array of RecipientInformationDesc structures describing the recipients of the message.

Return values:

MAPI_E_FAILURE in all other error conditions not listed below.

MAPI_E_INVALID_SESSION if Session = 0

MAPI_E_INVALID_MESSAGE if MessageID is not valid in session

Note: RecipientInfo object must be freed with MAPIFreeRecipientInformation function

2.26.1 RecipientInformationDesc

RecipientInformationDesc contains a MAPIRecipDesc that describe a recipient and the RecipientDeliveryDesc delivery information associated to the recipient.

C and C++

```
typedef struct {
    lpMapiRecipDesc Recipient;
    lpRecipientDeliveryDesc Delivery;
} RecipientInformationDesc, far *lpRecipientInformationDesc;
```

Recipient

MAPIRecipDesc object that represst a recipient.

Delivery

RecipientInformationDesc object that defines recipient details.

2.26.2 RecipientDeliveryDesc

RecipientDeliveryDesc contains information about requested information of a recipient.

Each time field has the following format: YYYY/MM/DD HH:MM:SS. If time is not set, value is an empty string.

C and C++

```
typedef struct {
    // Delivery Notification
    BOOL          DNRequested;
    BOOL          DNReceived;
    LPSTR         DNReceivedTime;
    // Non Delivery Notification
    BOOL          NDNRequested;
    BOOL          NDNReceived;
    LPSTR         NDNReceivedTime;
    LPSTR         NDNReason;
    LPSTR         NDNDiagnostic;
    // Receipt Notification
    BOOL          RNRequested;
    BOOL          RNReceived;
    LPSTR         RNReceivedTime;
    // Non Receipt Notification
```

```
        BOOL          NRNRequested;  
    BOOL  NRNReceived;  
    LPSTR          NRNReceivedTime;  
        LPSTR          NRNReason;  
    LPSTR  NRNDiagnostic;  
        LPSTR          Answered;  
        LPSTR          AnsweredTime;  
} RecipientInformationDesc, far *lpRecipientInformationDesc;
```

DNReceivedTime is available only if DNReceived is TRUE

NDNReceivedTime, NDNReason and NDNDiagnostic are available only if NDNReceived is TRUE

RNReceivedTime is available only if RNReceived is TRUE

NRNReceivedTime, NRNReason and NRNDiagnostic are available only if NRNReceived is TRUE

AnsweredTime is available only if Answered is TRUE

2.27 GetMessageInformation

GetMessageInformation retrieved details of a message.

C and C++

```
ULONG GetMessageInformation(  
    LPHANDLE Session,  
    LPSTR MessageID,  
    LpMessageInformationDesc* MessageInfo);
```

Session

An opaque session handle whose value represents a session with the messaging subsystem. The session handle is returned by MAPILogon and invalidated by MAPILogoff.

MessageID

Mail's string identifier for this message returned by MAPIFindNext or MAPISaveMail.

MessageInfo

Point to a MessageInformationDesc structure.

Return values:

MAPI_E_FAILURE in all other error conditions not listed below.

MAPI_E_INVALID_SESSION if Session = 0

MAPI_E_INVALID_MESSAGE if MessageID is not valid in session

Note: MessageInfo object must be freed with MAPIFreeMessageInformation function

2.27.1 MessageInformationDesc

MessageInformationDesc contains information about requested information of a message.

Each time has the following format: YYYY/MM/DD HH:MM:SS. If time is not set, value is an empty string.

C and C++

```
typedef struct {
    LPSTR Status;
    LPSTR MessageID;
    LPSTR MTSID;
    INT Importance;
    INT Priority;
    INT Sensitivity;
    LPSTR InReplyTo;
    LPSTR SubmissionTime
    LPSTR DeliveryTime
    LPSTR FetchedTime
    BOOL IsForwarded
    BOOL IsReply
    BOOL ReadReceiptRequested;
    BOOL ReplyRequested;
    LPSTR ExpiredTime;
    INT Integrity;
    INT Encrypted;
} MessageInformationDesc, far *lpMessageInformationDesc;
```

Status

Indicate the status of the message.

MessageID

Id of the message

MTSID

Server message Id

Importance

Importance of the message. Values:

- 0 = Low
- 1 = Normal
- 2 = High

Priority

Priority of the message. Values:

- 0 = Normal
- 1 = Non urgent
- 2 = Urgent

Sensitivity

Sensitivity of the message. Values:

- 0 = None
- 1 = Personal
- 2 = Private
- 3 = Confidential

InReplyTo

Message id of original message

SubmissionTime

Time when the message has been submitted

DeliveryTime

Sent message: Time when the message has been delivered

Received message: This parameter is unavailable

FetchTime

Sent message: Time when the message has been fetched

Received message: Time when the last report has been fetched

IsForwarded

Is the message a forwarded message

IsReply

Is the message a reply message

ReadReceiptRequested

Read receipt notification requested

ReplyRequested

Reply notification requested

ExpiredTime

Message expiration time

Integrity

Integrity information. Values:

- 1 = No integrity information
- 0 = Integrity check failed
- 1 = Integrity check succeed

Encrypted

Encryption information Values:

- 1 = Information unknown
- 0 = Bodies non ecrypted
- 1 = Bodies encrypted

2.28 MAPIFreeRecipientInformation

Free memory allocated by **GetRecipientInformation** calls.

C and C++

ULONG MAPIFreeBuffer (LpRecipientInformationDesc pRecipientInfo, ULONG RecipientCount)

pRecipientInfo

A pointer to a RecipientInformationDesc object to be freed.

RecipientCount

Number of recipient

2.29 MAPIFreeMessageInformation

Free memory allocated by **GetMessageInformation** calls.

C and C++

ULONG MAPIFreeBuffer (LpMessageInformationDesc pMessageInfo)

pMessageInfo

A pointer to a MessageInformationDesc object to be freed.

2.30 Return values and error codes

A brief list of other possible error codes:

SUCCESS_SUCCESS

#define SUCCESS_SUCCESS 0
No failure.

MAPI_USER_ABORT

#define MAPI_USER_ABORT 1
Returned if the user presses Cancel in a dialog.

MAPI_E_FAILURE

#define MAPI_E_FAILURE 2
General (unspecified) error

MAPI_E_LOGIN_FAILURE

#define MAPI_E_LOGIN_FAILURE 3
Login went wrong for an unspecified reason.

MAPI_E_DISK_FULL

#define MAPI_E_DISK_FULL 4
The disk became full while MAPI was trying to save something.

MAPI_E_INSUFFICIENT_MEMORY

#define MAPI_E_INSUFFICIENT_MEMORY 5
Lack of free memory or Windows resources.

MAPI_E_ACCESS_DENIED

#define MAPI_E_ACCESS_DENIED 6
Wrong password

MAPI_TOO_MANY_SESSIONS

#define MAPI_E_TOO_MANY_SESSIONS 8
Attempt to log in too many applications with separate sessions, or the same application too many times.

MAPI_E_TOO_MANY_FILES

#define MAPI_E_TOO_MANY_FILES 9
No more free file handles in MS Windows.

MAPI_E_TOO_MANY_RECIPIENTS

#define MAPI_E_TOO_MANY_RECIPIENTS 10
Attempt to save / send a message with too many recipients defined.

MAPI_E_ATTACHMENT_NOT_FOUND

#define MAPI_E_ATTACHMENT_NOT_FOUND 11
The path or file name in a MAPIFileDesc record is not pointing to a file.

MAPI_E_ATTACHMENT_OPEN_FAILURE

```
#define MAPI_E_ATTACHMENT_OPEN_FAILURE 12
```

The file may be write protected or opened by another (or your) application.

MAPI_E_ATTACHMENT_WRITE_FAILURE

```
#define MAPI_E_ATTACHMENT_WRITE_FAILURE 13
```

The file may be write protected or opened by another (or your) application.

MAPI_E_UNKNOWN_RECIPIENT

```
#define MAPI_E_UNKNOWN_RECIPIENT 14
```

No dialog was allowed on sendmail or resolvename, and a RecipDesc contained a name that was not found in the address book, or an address with a syntax error.

MAPI_E_BAD_RECIPTYPE

```
#define MAPI_E_BAD_RECIPTYPE 15
```

Recipient type was not TO, CC or BCC

MAPI_E_NO_MESSAGES

```
#define MAPI_E_NO_MESSAGES 16
```

MAPI_E_INVALID_MESSAGE

```
#define MAPI_E_INVALID_MESSAGE 17
```

An attempt was made to save or send a message that contained an error, or a message without at least one recipient.

MAPI_E_TEXT_TOO_LARGE

```
#define MAPI_E_TEXT_TOO_LARGE 18
```

The text note was too large.

MAPI_E_INVALID_SESSION

```
#define MAPI_E_INVALID_SESSION 19
```

The session handle was 0 or did not point to a valid MAPI session.

MAPI_E_TYPE_NOT_SUPPORTED

```
#define MAPI_E_TYPE_NOT_SUPPORTED 20
```

MAPI_E_AMBIGUOUS_RECIPIENT

```
#define MAPI_E_AMBIGUOUS_RECIPIENT 21
```

The specified recipient to MAPIResolveName, MAPISaveMail or MAPISendMail did match more than one recipient, and the flag MAPI_DIALOG was not set.

MAPI_E_MESSAGE_IN_USE

```
#define MAPI_E_MESSAGE_IN_USE 22
```

The message was opened by another application (a MAPI application, MailmaX.400 or the Mail Spooler). Note that all of these only keep the message open while it is being saved, not while it is being viewed or edited.

MAPI_E_NETWORK_FAILURE

```
#define MAPI_E_NETWORK_FAILURE 23
```

A network failure occurred while MAPI was trying to save some information.

MAPI_E_INVALID_EDITFIELDS

```
#define MAPI_E_INVALID_EDITFIELDS 24
```

nEditFields to MAPIAddress was not a legal value.

MAPI_E_INVALID_RECIPS

#define MAPI_E_INVALID_RECIPS 25

MAPI_E_NOT_SUPPORTED

#define MAPI_E_NOT_SUPPORTED 26

The MaXware Simple MAPI module does not support the feature requested.

2.31 Sending a file using C

```
#include <stdio.h>
#include <windows.h>
#include <mapi.h>

MapiFileDesc files = {
    0,                /* reserved - has to be 0 */
    0,                /* 0 -MaXware Simple MAPI doesn't support OLE */
    0,                /* not used when sending */
    ".\\testfile",   /* attached file name */
    NULL,            /* we don't know it */
    NULL};           /* not used in this example */

MapiRecipDesc recipes[2]= {
    0,                /* reserved - has to be 0 */
    MAPI_TO,         /* one of MAPI_TO/CC */
    "Boss",          /* Address book /friendly recipient name */
    "Boss",          /* recipient address - OR name, here alias */
    0,                /* not used */
    NULL},           /* not used */
    {0, MAPI_CC, "Secretary", "Secretary", 0, NULL }};

MapiMessage note = {
    0,                /* reserved - has to be 0 */
    "Test from Simple MAPI", /* message subject */
    "MaXware Simple MAPI 1.0 Interface Test", /* message text */
    NULL,            /* mess. class (later set to IP_MESSAGE) */
    NULL,            /* date received - not appropriate here */
    NULL,            /* not used */
    0,                /* status - is set later */
    NULL,            /* originator pointer - filled later */
    2,                /* number of recipients */
    &recipes,        /* pointer to recipient array */
    1,                /* number of attachments */
    &files};         /* pointer to attachment struct */

main() {
    ULONG ulRes;
    LPHANDLE session;

    ulRes = MAPILogon(0,"myuser", "", 0L, 0, &session);
    ulRes = MAPISendMail(session, 0L, &note, 0, 0);
    ulRes = MAPILogoff(session, 0L, MAPI_LOGOFF_SHARED,0);
}
```

2.32 Reading a message with C

```

/* Find the last (newest) message from the OUTTRAY folder.*/

LPHANDLE session;
pMapiMessage pMessage = NULL;
char messid1[20];

ulRes = MAPILogon(0, "myuser", "", MAPI_EXPLICIT_PROFILE, 0 &session);
/* find the last unread message, (seed MessageID is empty) */
ulRes = MAPIFindNext(session, 0L, "", "", MAPI_SEARCH_OUTTRAY, 0, messid0);
if (ulRes != SUCCESS_SUCCESS) { ; /* process error */
}
/* Do something with this message */
ulRes = MAPILogoff(session, 0L, MAPI_LOGOFF_SHARED, 0);
/* If we read one message sent from Boss, with 1 attached file, the resulting message
record should look like this: */

MapiMessage note = {
    0, /* reserved - has to be 0 */
    "Call for meeting", /* message subject */
    NULL, /* message text, treated as attachment.*/
    "2", /* UAENT_IP_MESSAGE */
    "94/05/12 10:24:15", /* date received */
    NULL, /* not used */
    1, /* MAPI_UNREAD */
    &orig, /* originator record pointer */
    1, /* number of recipients */
    &recips, /* pointer to recipient array */
    1, /* number of attachments */
    &files}; /* pointer to attachment struct */

MapiRecipDesc orig = {
    0, /* reserved - has to be 0 */
    0, /* MAPI_ORIG */
    "Boss", /* friendly recipient name */
    "C=no;A=telemax;S=Jones;G=John;", /*originator's name */
    0, /* not used */
    NULL}; /* not used */

MapiRecipDesc recips = {
    0, /* reserved - has to be 0 */
    1, /* MAPI_TO */
    "myuser", /* friendly recipient name */
    "C=no;A=telemax; ...", /* my O/R name */
    0, /* not used */
    NULL}; /* not used */

MapiFileDesc files = {
    0, /* reserved - has to be 0 */
    0,
    -1, /* never OLE */
    "c:\mailmax\myuser\h-000002\hdr-28.b01", /* attached file name */
    "c:\temp\bp003654", /* the name of the file as we see it */
    "ia5"};

/* Assume that messages were in the following order in the INTRAY.

```

ID	Status	Date	
1050	read	10.05	
1048	UNREAD	09.05	
1047	UNREAD	09.05	
1030	read	08.05	*/

2.33 Read unread messages with C

```
\*
Find the first 'unread' letter, 'read' it but leave it 'unread'. Find the next message (don't
care if 'read' or 'unread'). Read the whole message and process the text note as an
attachment. */

LPHANDLE session;
pMapiMessage pMessage = NULL;
char messid0[20], messid1[20];

ulRes = MAPILogon(0, "myuser", "", MAPI_EXPLICIT_PROFILE, 0,&session);

ulRes = MAPIFindNext(session, 0L, "", "", MAPI_UNREAD_ONLY, 0, messid0);
if (ulRes != SUCCESS_SUCCESS) { ; /* process error */
} /* messid0 has value 1048 */

/* find the next message that is older than the message just found */
ulRes = MAPIFindNext(session, 0L, "", messid0, 0, 0, messid1);
if (ulRes != SUCCESS_SUCCESS) { ; /* process error */
} /* messid1 has value 1047

ulRes = MAPIReadMail(session, 0L, messid0, MAPI_BODY_AS_FILE | MAPI_PEEK, 0,
&pMessage);

/* Do something with the information in pMessage */
ulRes = MAPIFreeBuffer(pMessage);
ulRes = MAPILogoff(session, 0L, MAPI_LOGOFF_SHARED,0);
```

2.34 Delete messages with a certain subject

```

/* Delete mail */

ULONG   ulRes;
LPHANDLE session;
pMapiMessage pMessage = NULL;
char messid0[20], messid1[20];

ulRes = MAPILogon(0, "myuser", "", MAPI_EXPLICIT_PROFILE, 0, &session);
ulRes = MAPIFindNext(session, 0L, "", "", 0, 0, messid1);
while (ulRes == SUCCESS_SUCCESS) {
    /* For simplicity - we don't test on errors when reading and deleting mail!
    */
    ulRes = MAPIReadMail(session, 0L, messid1, MAPI_PEEK |
        MAPI_ENVELOPE_ONLY, 0, &pMessage);
    /* Delete all read messages with the word 'test' in the subject.
    */
    if (!(pMessage->flFlags & MAPI_UNREAD))
        && (strstr(pMessage->lpszSubject, "test")) {
            ulRes = MAPIDeleteMail(session, 0, messid1, 0L, 0);
            messid0[0] = '\0'; /* no seed */
        }
    else
        strcpy(messid0, messid1); /* set new seed */
    ulRes = MAPIFindNext(session, 0L, "", messid0, 0, 0, messid1);
}
ulRes = MAPILogoff(session, 0L, MAPI_LOGOFF_SHARED, 0);

```

3. DDE programming with MailmaX.400

3.1 Introduction

MailmaX.400 for Windows can receive simple commands through Windows DDE (Dynamic Data Exchange). These calls are intended to make it possible for Windows programs with DDE capability to send X.400 messages.

In Windows terminology, MailmaX.400 acts as a “DDE Server” and will handle calls from “DDE Clients”.

Procedures:

To give commands to MailmaX.400 for Windows with DDE:

- Start MailmaX.400 (if not started already).
- Create a DDE link with the following parameters:

Application:	Mailmax
Topic:	System
Command:	Help

MailmaX.400 will now return a short string describing the commands available.

- Use DDE calls to execute one or more commands. The list of commands available is documented below.
- Terminate the DDE link.

3.2 DDE Commands

SendAndReceive

(no parameters)

Asks MailmaX.400 to connect to the central Message Store, send all messages with status “Ready” and fetch all messages waiting in the Message Store.

CreateMessage

Creates a message in the MailmaX.400 OUTTRAY.

Parameters:

Flag, Subject, File name, Recipient(s)

Flag = 0: MailmaX.400 packages the document as an attachment, adds recipients and subject, and shows the “Compose message” window on the screen for the user to finish the message.

Flag = 1: The message is stored directly in the OUTTRAY with the status “Ready” without asking the user for additional parameters.

Flag = 3: The message is stored directly in the OUTTRAY with the status “Ready” without asking the user for additional parameters, and MailmaX.400 is asked to transfer all messages with the status “Ready”. (Sending only - no messages are fetched).

Subject: A text string is added as the message subject. The subject string is enclosed by quotation marks (“”) if it contains spaces or commas.

File name: Name of file attached to the message.

Note 1: MailmaX.400 uses pattern recognition to identify the file type of the document, so that when the recipient sees the document in the OUTTRAY, it is displayed with the icon of the program in which it was created. Seven-bit files are sent as the type “Text”, while files containing binary data are sent as the type “Data”.

Note 2: If the user does not edit the message (in the “Compose message window), the attachment created with this call is sent as the first and only document in the message (X.400 BodyPart).

Recipient(s): A list of valid recipients to be added to the message. When multiple users are defined, use a comma as the delimiter between names. Each recipient may be identified by any one of the following:

- A short name defined in the MailmaX.400 address book.
- An address list defined in the MailmaX.400 address book.
- An X.400 address in the form:
S=Maxdata;O=Maxdata;A=Service;C=NO

The recipient list must be enclosed in quotation marks (“”) if it

contains blanks or commas.

Help:

ddeExecute with the command Help will make MailmaX.400 display a dialog box documenting the DDE calls.

AddRecipients

Adds the recipients from another application (for example MaXware Directory Explorer or MaXware Directory Browser) to an open Compose window. If the Compose window is not open, the AddRecipients command opens it to create a new message.

Parameters:

Recipient(s)

Recipient(s): See the description of this parameter for the CreateMessage command..

Help: ddeExecute with the command Help will make MailmaX.400 display a dialog box documenting the DDE calls.

3.3 Starting MailmaX.400 for Windows

When you start MailmaX.400 for Windows, you can use the parameter “fast” on the command line:

mailmax.exe fast

This causes MailmaX.400 to start without opening folders as windows or icons. Only the OUTTRAY Folder is opened. When the user exits MailmaX.400, the configuration changes are not stored.

When MailmaX.400 is started from a program that immediately attempts to establish a DDE link with MailmaX.400, the DDE call will fail. The reason is that MailmaX.400 must be started with a user name and password before it can respond to a DDE call. To avoid this problem, the calling program can wait for a while after MailmaX.400 has been started before DDE communication is attempted. The Access example below demonstrates a different strategy: it is possible to keep on trying DDE until it is successful, or until a counter is full.

3.4 C example

```

HANDLE hLibInstance;    // Library module handle
int StartByDDE(sDocumentname)
{
    CONVCONTEXT CCFilter = { sizeof(CONVCONTEXT),0,0,0,0L,0L};
    DWORD idInst    = 0;
    HSZ hszAppName  = NULL;
    HSZ hszTopic    = NULL;
    HCONV hConv     = NULL;
    BOOL bRes       = FALSE;
    /* MailmaX.400 must be running */
    /* Build the DDE command: */
    strcpy (cDDECommand, "[CreateMessage(0,,");
    strcpy (cLast, ",)");
    strcat (cDDECommand, sDocumentname);
    strcat (cDDECommand, cLast);
    cDDEApplication[] = "MailmaX";
    cDDETopic[] = "system";
    CCFilter.iCodePage = CP_WINANSI; //initial default codep.
    if (!DdeInitialize(&idInst,
        (PFNCALLBACK)MakeProcInstance((FARPROC)DdeCallback,
            hLibInstance), APPCMD_FILTERINITS, NULL))
    {
        hszAppName=DdeCreateStringHandle(idInst,lpAppName,NULL);
        hszTopic =DdeCreateStringHandle(idInst,lpTopic,NULL);
        hConv =DdeConnect(idInst,hszAppName,hszTopic,NULL);
        if (hConv)
            {
                bRes =DdeClientTransaction(lpExecCommand,
                    lstrlen(lpExecCommand)+1,
                    hConv, NULL, CF_TEXT,XTYP_EXECUTE, 1000000L, NULL);
                DdeDisconnect(hConv);
            }
        DdeFreeStringHandle(idInst, hszAppName);
        DdeFreeStringHandle(idInst, hszTopic);
        DdeUninitialize(idInst);
    }
    return(bRes);
}

HDDEDATA EXPENTRY DdeCallback(WORD wType,WORD wFmt,HCONV
hConv, HSZ hszTopic, HSZ hszItem,HDDEDATA hData,DWORD
IData1,DWORD IData2)
{
    switch (wType)
    {
        case XTYP_CONNECT_CONFIRM:
            return(0);
            break;
        case XTYP_DISCONNECT:
            return(0);
            break;
        default:
            break;
    }
    return(0);}

```

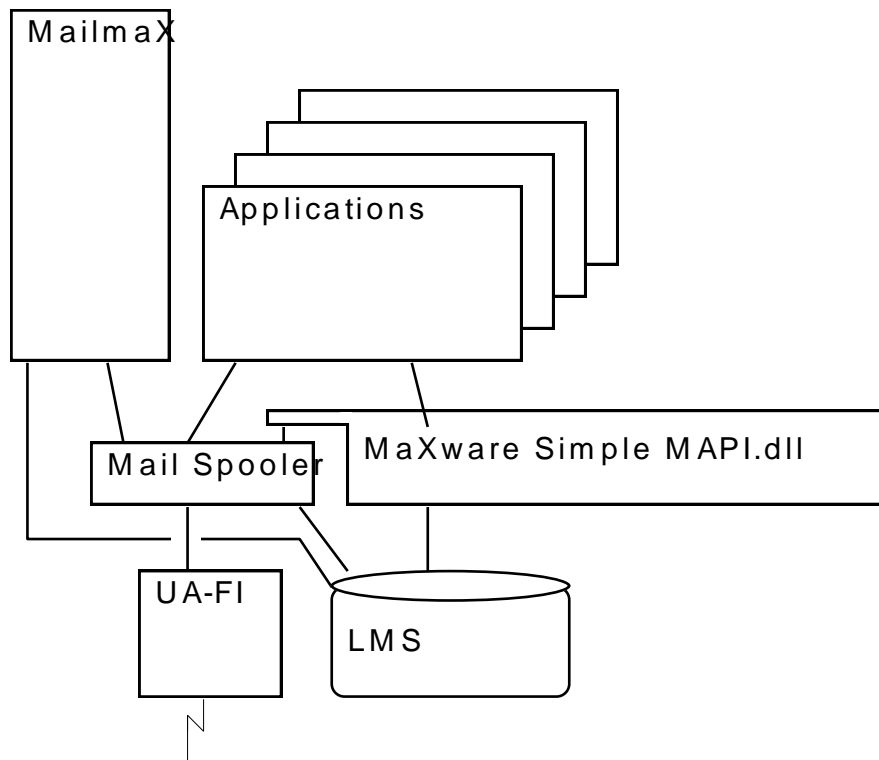
4. Controlling the communication process

4.1 Introduction to the MaXware Mail Spooler

The MaXware Mail Spooler is a separate module in the MailmaX.400 for Windows product. The Mail Spooler handles all communication triggered by the user (MailmaX.400 commands), by applications, and / or by timer events (time intervals).

The module is implemented as a separate .EXE file, and installed with MailmaX.400 for Windows and MaXware Simple MAPI for Windows.

4.2 Architecture



4.3 Basic features

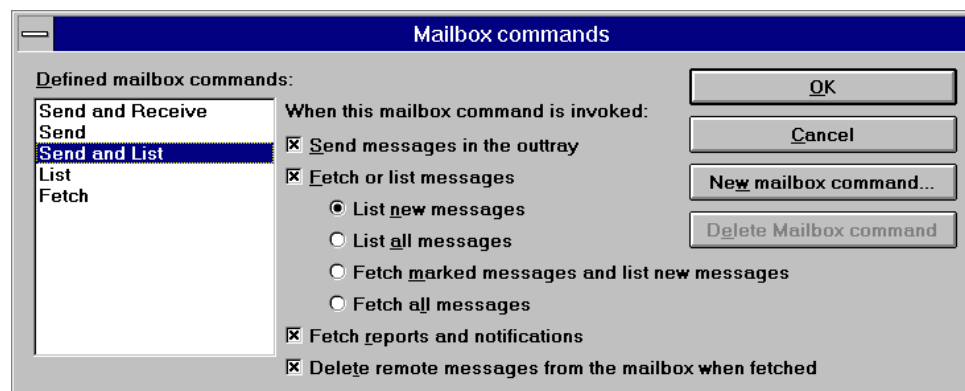
- The Mail Spooler is asynchronous to and runs in parallel with the calling application and MailmaX.400.
- It is callable from MailmaX.400 for Windows and other applications by DDE
- The MailmaX.400 communication commands are implemented as Mailbox Commands: Send, Fetch, List, Send and Receive, etc.
- There is support for scheduled Mailbox Commands at fixed intervals.
- The Mail Spooler will stay active until the communication process is finished, even if the application logs off.
- The Mail Spooler will report a summary back to the calling application with the main results of the communication session.
- It is possible to run the Mail Spooler by giving only a user name, without a password, since the user cannot access local data through the Mail Spooler. (The default user name in MAXWARE.INI makes it feasible to have the spooler loaded at startup).
- The Mail Spooler keeps a queue of DDE-calls, if more than one application tries to give calls while communication is active.

4.4 Configuring the Mailbox Commands

You define the mailbox commands using the dialog box “Mailbox commands” in MailmaX.400:

- Select Tools/Options/Mailbox Commands

The following dialog box is displayed:



The list box to the left lists all the defined Mailbox Commands

4.5 Start the Mail Spooler

Startup:

maxspool <User name, <MailboxCommand>> <-Setup>

Parameters:

User Name:

Local (LMS) user name.

MailboxCommand:

If present, run this MailboxCommand and then terminate immediately.

-Setup:

Use this option to enter the Setup window directly.

(No logon necessary)

4.6 DDE commands to the Mail Spooler

Service: **MaxSpool**

Topic: **System**

Syntax: [DDE command]

4.6.1 Logon (User Name)

Used to change the User Name / LMS / O/R-Name for the Mail Spooler.

Note that the user name must be in double quotes if it contains spaces or other special characters (characters other than A-Z and 0-9).

4.6.2 Connect(MailboxCommand)

Connect according to parameters at startup/logon and the Connect MailboxCommand given.

Parameter:

MailboxCommand:

The name of a defined MailboxCommand to invoke.

Note that it is normally preferable to use the command number and NOT the name, as the names will vary with the language of MailmaX.400, and with user preferences.

4.6.3 IniFileChanged()

Signals to the Mail Spooler that the .INI file has changed, so a re-initialization is necessary.

4.6.4 OUTTRAYChanged()

Signals to the Mail Spooler that a letter is stored in the OUTTRAY with state 'Ready', or that the state of a letter in the OUTTRAY has changed to 'Ready'. Note that the Simple MAPI API and MailmaX.400 for Windows do this automatically when a new message is submitted.

4.6.5 LogOff(LogOffFlag)

Note: It is not necessary to call LogOff if logon was done on the command line, as logoff is then done automatically.

LogOffFlag values:

- 1: Terminate spooler

4.7 DDE Services from the Mail Spooler

4.7.1 State

Service: **MaxSpool**

Topic: **System**

Item: **State**

One of: "Idle", "Preparing", "Communicating", "Processing"

4.7.2 Mail

Service: **MaxSpool**

Topic: **System**

Item: **Mail**

Data: TAB delimited list of parameters:

Error Status: Integer, 0 = ok, >0 = 32bit MaXware module+error msg no

NewMessages: Integer

SentMessages: Integer

UnreadMessages: Integer

FailedMessages: Integer

The 'Unread' counter is updated each time the MailSpooler wakes up, to capture any letters read since last time, and when a communication session is completed. The value is also reflected in the program's caption as described above.

4.7.3 ErrorText

Service: **MaxSpool**

Topic: **System**

Item: **ErrorText**

The Error text of any error reported in the Item: Mail, Always "" when ErrorStatus is 0 in the item 'Mail'.

4.8 DDE System topics

Service: **MaxSpool**

Topic: **System**

Items:

"Topics"

A list of supported topics (System, Status)

"Formats"

"Text"

"TopicItemList"

A list of supported items under System (Topics, Formats, TopicItemList, Help, logon, mail, state, errortext)

"Help"

A string containing a description of the supported calls (logon, connect, OUTTRAYChanged Logoff)

"Version"

A string containing version.revision <space> language code of the Mail Spooler.

5. Hooks for calling an Extension DLL library

5.1 General

It is possible to register up to ten Extension DLLs containing C routines which will be called by MailmaX.400, the MaXware Mail Spooler, the MaXware S-MAPI, on certain events, like:

- | | |
|---|----------------------|
| • When opening a message | OnMessageOpen |
| • When saving a message | OnMessageSave |
| • When setting a message as “Ready to send” | OnSubmit |
| • When processing messages to be sent | OnConnect |
| • When processing incoming messages | OnDisconnect |
| • When connecting to the MS or DSA | OnCommunicationStart |
| • When disconnecting from the MS or DSA. | OnCommunicationEnd |

To register the Extension DLL and its routines, insert the statement “ExtensionN” in the [Communication] section of the <service profile>.PRF file for MailmaX.400 :

```
[Communication]
Extension1=<ExtensionName>,<implemented-bitmap>,<filename and path>
```

Note:

The number N after ‘Extension’ must be in the range 1-10.

The <ExtensionName> is a user-oriented name for the Extension DLL. It is shown in the “About...” box of MailmaX.400.

The <implemented-bitmap> is a 32-bit integer (decimal) that tells which routines are included in the Extension DLL. The following flags are defined:

```
cOnSubmit = 64
cOnConnect = 128
cOnDisconnect=256
cOnCommunicationStart=512
cOnCommunicationEnd=1024
cOnMessageOpen=2048
cOnMessageSave=4096
```

The <filename and path> should contain the name of an Extension .DLL file in either the MailmaX directory, the MS-DOS path, or in the \windows\system directory. If the .DLL file name contains the two characters ??, they will be substituted with the two-letter ISO-language code of the MailmaX.400 version that is trying to call the DLL. This file name may also be an absolute path pointing to the DLL.

Examples:

CHKMSG?? .DLL expands to CHKMSGEN.DLL, CHKMSGFR.DLL, etc.

5.2 Routines and data structures in the Extension DLL library

An Extension library to be used with the MaXware products **MUST** implement the “Initialize” and the “Terminate” routine. In addition, the library may implement one or more of the “event-handling routines” that will be called by MailmaX.400 when the corresponding event occurs.

Each event-handling routine in the library must be declared by setting the corresponding bit in the <implemented-bitmap> field in the “ExtensionN” statement in the service profile file as described above. No flag is needed to signal that the Initialize and Terminate calls are implemented.

The following MaXware programs/modules may call routines in the Extension DLL:

- MailmaX.400 for Windows.
- The MaXware Mail Spooler.
- The MaXware Simple MAPI.

The Extension .DLL must be implemented so that this does not cause a problem for the calling MaXware program.

The routines in the Extension library must be able to display their own error message dialogs.

NOTE: All flags that are not used must be set to 0.

5.3 Initialize

```
long FAR PASCAL __export Initialize (
    lpMAXFLAGS          pFlags,
    lpMAILMAXVERSION   pMxVer,
    LPSTR               szarIdent)
```

The Initialize routine will always be called by the calling MaXware program/module before any other routines in the Extension .DLL are called. If any error occurs when trying to load the Extension DLL, the calling MaXware program will continue as if no Extension DLL had been registered.

Parameters:

Flags

32-bit flag word. Reserved for future use. Must always be set to 0.

Version

A pointer to a MaXware version struct describing the program/module that is calling the Extension .DLL. The version struct is described below.

ExtensionIdent

A pointer to a buffer containing up to 40 bytes, identifying and describing the extension DLL being used. The calling MaXware program will set up this buffer and the pointer. The called Extension .DLL may fill in a string in the buffer. The calling MaXware program may show the string in its Help/About dialog.

The MaXware version struct:

```
typedef struct
    int          StructVersion,
    HWND        MainWindow,
    LONG        Flags,
    int          Program,
    char         *Version,
    char         *CommsDriverVersion,
    char         *Driver,
    char         *OwnORName,
    char         *Language,
    char         *UserName,
    char         *UserProfileName,
    char         *InstallationPath)
MailmaxVersion;
```

Where:

StructVersion

Set to 0.

MainWindow

A handle to the main window of the program (MailmaX.400, or S-MAPI, or Mail Spooler) that is calling the Initialize routine.

Flags

32-bit flag word. Reserved for future use. Is set to 0 by the calling MaXware program

Program

Identifies the program that is calling the Initialize routine:

- 0: MailmaX.400 for Windows.
- 1: The MaXware Mail Spooler.
- 2: The MaXware Simple MAPI.

Version

A string containing the version id of the program that is calling the Initialize routine.

NOTE: The Mail Spooler and Simple-MAPI will pass the version id describing MailmaX.400 for Windows.

CommsDriverVersion

If MailmaX.400, Mail Spooler or S-MAPI is calling the Initialize routine, a string describing the currently used version of the MaXware UA-FI communication module is passed (like 3.3.02).

Driver

A string describing the communication driver type: APS, EIC, ATL or TCP. Other drivers may be added later.

OwnORName

If MailmaX.400, Mail Spooler or S-MAPI is calling, the calling user's own X.400 address (given as a string on F.400 syntax) is passed.

Language

The language version of the program. Two-letter ISO language code (US, EN, FR, IT, NO.....).

UserName

The full User Name of the current user, as used when logging into MailmaX.400, S-MAPI or the Mail Spooler.

UserProfileName

The "filtered" and "truncated" version of the "MailmaX.400" UserName, used in the file system directory-name for this user's directory (e.g. c:\mailmax\john) and as the name for the "user name" in the [USER.<user name>] section in MAXWARE.INI. Characters that are not part of the "Printable string" set are filtered out, and the name is truncated to 8 characters (if necessary).

InstallationPath

The file system directory where the calling program is installed.

5.4 OnMessageOpen

This routine is called:

- When the user opens a message from the Folder Window in MailmaX.400 (regardless of folder and message status).
- When the user opens a Forwarded BP (message) in the “View” Window or in the “Compose” window of MailmaX.400.
- When the “Print” tool button in MailmaX.400 is pressed.
- When the command “File/Print...” in MailmaX.400 is activated.
- When the “Export” command in MailmaX.400 is activated

Routine definition:

```
long FAR PASCAL __export OnMessageOpen      (  
    lpMAXFLAGS      pFlags,  
    lpMapiMessage   pMessage,  
    ULONG           Id,  
    int             iStatus,  
    int             iEvent)
```

Parameters:

pFlags

32-bit flag word. If the message is read, the flag cMessageRead will be set.

pMessage

A pointer to a Simple-MAPI "MapiMessage" structure defining the message to be opened. The exact definition of this structure can be found in the "MailmaX.400 for Windows Programmers Guide", or in the Microsoft Simple MAPI documentation.

Id

LMS Reference number

iStatus

The status of the message:

- 1000 : Outgoing message with status "Draft".
- 1001 : Outgoing message with status "Ready".
- (1002 : Outgoing message with status "Submitting".)
- 1003 : Outgoing message with status "Sent".
- 1004 : Outgoing message with status "Delivered".
- 1005 : Outgoing message with status "Read".
- 1006 : Outgoing message with status "Replied to".
- 1007 : Outgoing message with status "Failed".

- 1100 : Incoming message.
- (1101 : Incoming message with status "Expired")
- (1102 : Incoming message with status "Obsolete".)

Event

The event that caused OnMessageOpen to be called:

- 0: Open message from a Folder was activated
- 1: The Print tool button was activated.
- 2: The File/Print command was activated.
- 3: The File/Export command or Export tool button was activated.

If the return value is 0 (success and message processed), MailmaX.400 will NOT process the event.

If the return value is 1 (message not processed), MailmaX.400 will process the event.

If the return value is 2 (failure), MailmaX.400 will not process the event (terminate the operation of the event).

Note: The routine OnMessageOpen is responsible for its own error handling and error dialogs. The return value is only used to tell MailmaX.400 whether it should complete the operation or not.

5.5 OnMessageSave

This routine is called:

- When the user saves a message from the MailmaX.400 Compose window.
- When the user saves a message using MapiSendMessage, MapiSendDocuments or MapiSaveMail

Routine definition:

```
long FAR PASCAL __export OnMessageSave(  
    lpMAXFLAGS          pFlags,  
    lpMapiMessage       pOriginalMessage,  
    lpMapiMessage FAR * ppNewMessage,  
    LPINT               pSensitivity,  
    BOOL                bPastelsDone)
```

Parameters:

pFlags

32-bit flag word. If the extension returns a changed message in ppNewMessage, the cOnSaveChangedFlag has to be set

pOriginalMessage

A pointer to a Simple-MAPI "MapiMessage" structure defining the message to be saved. The exact definition of this structure can be found in the "MailmaX.400 for Windows Programmers Guide", or in the Microsoft Simple MAPI documentation.

ppNewMessage

Pointer to a pointer to a new "Mapi Message". This message will be saved on the return to MailmaX.400.

pSensitivity

The value of the sensitivity parameter set by MailmaX.400.

0 = None, 1 = Personal, 2 = Private, 3 = Confidential.

The parameter may be changed by the extension DLL.

(The message must be returned by ppNewMessage, and the changed flag must be set.)

bPastelsDone

When this parameter is true, it reports that a paste has been done in the Compose window of MailmaX.400

If the return value is 0, the changed flag will be checked. If the flag is not set, the original message is saved; if it is set, the new message is saved.

If the return value is 1, no message will be saved. MailmaX.400 will return to the Compose window, and MAPI will return MAPI_FAILURE.

NOTE: The routine OnMessageSave is responsible for its own error handling and error dialogs. The return value is only used to tell MailmaX.400 whether it should complete the operation or not.

5.6 OnSubmit

```
long FAR PASCAL __export OnSubmit (  
    lpMAXFLAGS    pFlags,  
    lpMapiMessage pMessage)
```

The OnSubmit routine is called by MailmaX.400 and MaXware Simple MAPI when a message is to be saved to the OUTTRAY with the status 'Ready' (ready to be sent).

Parameters:

Flags

32-bit flag word. Reserved for future use. Is set to 0 by the calling MaXware program.

Message

A pointer to a Simple-MAPI structure defining the message to be submitted. For the exact definition of this structure, see the "MailmaX.400 for Windows Programmers Guide", or the Microsoft Simple MAPI documentation.

If the return value from the OnSubmit routine is 0 (success), the MailmaX.400/Simple MAPI operation "Save to Outtray" is performed as usual.

If the return value is different from 0, the following actions are performed by the calling program:

- If OnSubmit was called from the MailmaX.400 "Compose window":
The Compose window is still open and the message has not been saved. The user must modify the message according to the error/warning message that was given by the OnSubmit routine.
- If OnSubmit was called from the MailmaX.400 command "Mailbox/Ready to Send":
The message is still stored with status "Draft".
- If OnSubmit was called from the MaXware Simple MAPI "Compose window":
The Compose window is still open and the message has not been saved. The user must modify the message according to the error/warning message that was given by the OnSubmit routine.
- If OnSubmit was called from one of the Simple MAPI routines MAPISendMail or MAPISendDocument:
The MAPI error code MAPI_E_INVALID_MESSAGE will be returned.

NOTE: The OnSubmit routine is responsible for its own error handling and error dialogs. The return value is only used to tell MailmaX.400 / Simple MAPI whether the "Save to Outtray" operation should be completed or not.

5.7 OnConnect

```
long FAR PASCAL __export OnConnect      (
    lpMAXFLAGS      pFlags,
    int              iSpoolerCommand,
    LPSTR            pCommandFile,
    WORD             wCommandLen,
    LPSTR            pResponseFile,
    WORD             wResponseLen)
```

The OnConnect routine is called by the Mail Spooler when MailmaX.400, an application using the Mail Spooler, or an automatic Mail Spooler scheduling rule makes the Mail Spooler connect to the Message Store. OnConnect is called BEFORE any processing is done by the Mail Spooler (this means that OnConnect may add/delete/modify messages in the Local Message Store, but it can NOT modify the “Standard UA-FI CommandFile” to be generated by the Mail Spooler after the return from OnConnect).

The parameters CommandFile and ResponseFile must be set up by the OnConnect routine, and must be valid file names. The CommandFile must point to a valid UA-FI CommandFile to be INCLUDED in the “Standard UA-FI CommandFile” made by the Mail Spooler.

The Mail Spooler will include the specified Command file into the “Standard UA-FI Command file” by inserting the UA-FI INCLUDE command (See the “UA-FI Programmer Guide”). The flag “After” specifies the insertion position for the specified CommandFile in the “Standard UA-FI Command file”.

NOTE: The routines OnConnect / OnDisconnect will NOT be called if the Mail Spooler connects to the Message Store to change the Mailbox password.

Parameters:

Flags

32-bit flag word. Is set to 0 by the calling MaXware program.

Defined flags:

After=1. If this flag is NOT set by the OnConnect routine, the Mail Spooler inserts the "INCLUDE command" before any MailmaX.400-generated UA-FI commands in the main UA-FI Command file. If the flag is set, the "INCLUDE command" is inserted before the MailmaX.400 generated UA-FI commands.

SpoolerCommand

Integer describing the Mail Spooler mailbox command that caused the OnConnect routine to be called.

- 0: Send and Fetch
- 1: Send
- 2: Send and List
- 3: List
- 4: Fetch Marked Messages
- >4: User-defined Mail Spooler commands

CommandFile

Pointer to a buffer containing the file name of the UA-FI CommandFile to be included in the "Main UA-FI CommandFile". An empty string for the CommandFile name indicates that no CommandFile is to be included. The buffer is set up by the Mail Spooler.

CommandLen

Size of the buffer for the CommandFile file name. The size is given by the Mail Spooler.

ResponseFile

Pointer to a buffer containing the file name of the UA-FI ResponseFile in which the results of the included CommandFile should be written. An empty string for the ResponseFile name indicates that the results of the included CommandFile must be written to the "Standard ResponseFile" (UAFI.CMD on the \uafework sub-directory for the current user). The buffer is set up by the Mail Spooler.

ResponseLen

Size of the buffer for the ResponseFile file name. The size is given by the Mail Spooler.

If the return value is 0 (success), the Mail Spooler will continue preparing messages to be sent and the "Standard UA-FI CommandFile", and will connect to the Message Store as usual.

If the return value is different from 0, the Mail Spooler operation is aborted. The Mail Spooler will not connect to the Message Store.

5.8 OnDisconnect

```
long FAR PASCAL __export OnDisconnect      (  
    lpMAXFLAGS      pFlags)
```

The Mail Spooler calls OnDisconnect when it has disconnected a connection to the Message Store, and processed the “Standard UA-FI ResponseFile” (e.g. put new messages into the LMS). OnDisconnect is called BEFORE any auto-folding actions are done on the new messages in the LMS.

MailmaX.400 will NOT show the new messages (refresh the INTRAY) to the end user before the OnDisconnect routine has returned. The user is not allowed to manipulate new messages before the OnDisconnect routine is finished.

There are no parameters - normally this routine will act upon UA-FI statements found in the Response file that was handed over to the Mail Spooler (included) with the OnConnect call.

Typical operations performed by a OnDisconnect routine:

- Process the results of the included UA-FI CommandFile.
- Delete (from the LMS) messages that were sent successfully.
- Process fetched messages.

The OnDisconnect routine may also use Simple MAPI to read/modify new messages in the LMS.

Parameters:

Flags

32-bit flag word. Reserved for future use. Is set to 0 by the calling MaXware program.

Return value

The return value must always be 0, as this version of the Mail Spooler does not use it.

5.9 OnCommunicationStart

```
long FAR PASCAL __export OnCommunicationStart (  
    lpMAXFLAGS    pFlags,  
    LPSTR         szarError)
```

OnCommunicationStart is called by the Mail Spooler just before starting UA-FI to connect to the MS.

Typical operations performed by a OnCommunicationStart routine:

- Start a TCP-IP dialer to establish a dial-up IP-link connection to a TCP/IP network.

Parameters:

Flags

32-bit flag word. Reserved for future use. Is set to 0 by the calling MaXware program.

ErrorMessage

A pointer to a buffer containing an textual error message stating why the connection to the MS/DSA could not be established. The string can contain a maximum of 256 bytes. The buffer and pointer are set up by the calling MaXware program.

If the Mail Spooler is the calling program, the Mail Spooler will write the ErrorMessage to the UA-FI.RSP ResponseFile as "ABORTED : <ErrorMessage>" (as the only statement in the ResponseFile), so that the OnDisconnect routine knows that the communication failed. The Mail Spooler will call the OnDisconnect routine before aborting.

The return value 0 (success) means that the Mail Spooler can continue connecting to the server.

The return value 1 (failure) means that the Mail Spooler should not connect to the server. The OnCommunicationStart routine is responsible for giving an error message, etc. to the end user in its own dialog.

The return value 2 (wait) means that the Mail Spooler should wait 1 second and then call the OnCommunicationStart again. This allows the OnCommunicationStart routine to have a loop waiting for the IP dialer to set up the connection.

5.10 OnCommunicationEnd

```
long FAR PASCAL __export OnCommunicationEnd (  
    lpMAXFLAGS    pFlags)
```

OnCommunicationEnd is called by the Mail Spooler immediately after the UA-FI communication session with the MS is finished

Typical operations performed by a OnCommunicationEnd routine:

- Signal to a TCP/IP dialer that the IP-Link connection to the TCP/IP network should be disconnected.

Parameters:

Flags

32-bit flag word. Reserved for future use. Is set to 0 by the calling MaXware program.

The return value must always be 0, as this version of the Mail Spooler does not use it.

5.10.1 Terminate

```
long FAR PASCAL __export Terminate (  
    lpMAXFLAGS    Flags)
```

Terminate is always called when the calling MaXware program has finished using the Extension DLL library.

Parameters:

Flags

32-bit flag word. Reserved for future use. Is set to 0 by the calling MaXware program.

6. Start program from MailmaX.400 command

If you have made a mail-related application, and want it to appear as a command in MailmaX.400 for Windows, you can define it in the [Menu] section in MAXWARE.INI:

[Menu]

AddentryN=<sub-menu number>,<menu text>,<.EXE file name>

Where N is a number, all commands from 1 to the first missing number will be included.

The sub-menu number starts with 0 as the File menu, 1 as the Edit menu, etc.

Example:

```
[Menu]
addentry1=0,&Secure mail,C:\WA\MMW\mailer.exe
addentry2=1,&MaXware.Ini,notepad c:\wa\mmw\maxware.ini
addentry3=1,&UA-FI.LOG,notepad c:\wa\mmw\ua-fi.log
```

7. File viewers in MailmaX.400

MailmaX.400 for Windows automatically recognizes the most common file types used by Windows programs. MailmaX.400 uses the table file DOCMAGIC, which is described later in this guide, to perform pattern recognition.

NOTE: For the right program to be started when the user double-clicks on an attachment icon, it is important that the program is installed correctly. You check this in Program Manager by double-clicking a document file generated by the program and seeing whether the program starts and the document is opened in it. If the program is correctly registered in the REGEDIT database in Windows, MailmaX.400 hands the document over to the program without starting a new copy of the program. If the program has registered a printing command in REGEDIT, it will also be possible to print attachments from this program directly from MailmaX.400 for Windows.

WordPerfect for Windows has not assigned a unique file type. To start WordPerfect for Windows automatically, the program must be located in a directory that is part of the path. To define PATH, you must use the Control Panel System/Environment/Path setting.

MailmaX.400 will start some programs (AmiPro, JETFORM Filler/G...) with a copy of the document file. This means that any editing changes performed in the document or form will **not** be stored. Changes are not copied back to MailmaX.400 or to the original file.

You can get around this problem by using the "Send" command once more to mail a new copy of the edited document.

8. File format recognition

When reading and writing a message, MailmaX.400 automatically identifies the format of the attached documents by comparing their contents with the definitions in the DOCMAGIC file. This file is an ASCII file with key information about the file formats that various programs use.

The file is divided into two sections:

[User]

User-defined entries. These will be checked before the system-defined entries. The user section will not be changed / overwritten by the MailmaX.400 installation job.

The 'User' section can be used to define the file format of work group applications, forms applications, etc.

[System]

The system-defined part of the table. This section will be totally overwritten by the MailmaX.400 installation job.

You should never change the 'System' section. If there is an error in it, copy the entry to the 'User' section and correct it, and then report the problem to your MailmaX.400 distributor.

Each line in the file contains the definition of one file format. One definition contains the following information:

Position content application extension filter X.400 format fieldtypename format description.

The following example shows the definitions needed in DOCMAGIC to allow MailmaX.400 to recognize attached documents produced in Microsoft Excel version 4 and Lotus AmiPro version 3:

Postype	content	appl	ext	filt	X.400	name	description.
0	string \011\004\006 BIFF 4.0 file	Excel	".XLS"	NONE	BILATERAL	Excel	Excel
>6	byte 16 Excel 4.0 Workspace	Excel	".XLS"	NONE	BILATERAL	Excel	Excel
>6	byte 32 Excel 4.0 Graph	Excel	".XLS"	NONE	BILATERAL	Excel	Excel
>6	byte 64 Excel 4.0 Macro	Excel	".XLS"	NONE	BILATERAL	Excel	Excel
>6	string \000\001 Excel 4.0 workspace	Excel	".XLW"	NONE	BILATERAL	Excel	Excel
# Ami Pro							
0	string [ver] AmiPro AmiPro doc	Amipro	".SAM"	copytotemp		BILATERAL	
>11	string [sty] AmiPro AmiPro doc	Amipro	".SAM"	copytotemp		BILATERAL	
# Example with filter							
#0	string [ver] BILATERAL	Amipro	".SAM"	C:\mailmax\startami.pif		AmiPro	
#0	string [sty] BILATERAL	Amipro	".SAM"	C:\mailmax\startami.pif		AmiPro	

(lines in the DOCMAGIC. file must be unbroken)

The ">" character in DOCMAGIC means "logical or" between lines.

When you edit DOCMAGIC, you can use one or more spaces, or a tab, as a delimiter between the parameters.

Parameters:

The parameters in the example are:

Position:

the position of the byte in the file from which it is possible to recognize elements that identify the document.

Type:

'Byte' or 'String'. Specify a byte as parameter 2 if the value is to be searched for in the specified position is a byte; a string to search for a text string that starts at the specified position.

Content:

Specify the value in the byte or the content in the text string that identifies the document type.

/xxx/xxx/ is taken as a set of octal values

<string> is taken as a character string

Application:

The name of an application that can handle the document type.

File type:

Used to look up in the EXTENSIONS section in the WIN.INI and in the Windows REGEDIT database file to find the correct application to handle the file type.

Also used to suggest a proper file extension for the file when the command File/Export is given.

Filter:

A program that will be run with the file as a parameter before the application is started.

Special values:

- NONE No program
- copytotemp The file will be copied to a temporary file before the application is given this temporary file as a parameter. (used for applications that perform strict file-locking)
- <program>

X.400 document type:

Defines the attachment type that the underlying X.400 system will use for files of this type.

Possible values are IA5 (text files) and BILATERAL (binary data).

Format name:

A short description of the document type.

Description:

A descriptive text that explains which document type this is. This text is used in the message view window when an attachment of this type is in focus, and in the attachment list given by the command View/Documents.

9. UA-FI based applications with MailmaX.400

The Mail Spooler in MailmaX.400 for Windows implements a feature that makes it possible to integrate a UA-FI based (often MS-DOS based) application with MailmaX.400.

NOTE: For more information on MaXware UA-FI, see the manual “MaXware UA-FI Programmer’s Guide”.

For an overview of how this option is intended to be used, see the section “Scenario: MailmaX.400 user with automatic routing of EDI documents to an EDI application” in the document “Programming towards the MaXware products”.

To use the feature, your application or your installation job must insert an `INCLUDE` statement in the `[UA]` section of the `MAXWARE.INI` file. If there is more than one user defined in MailmaX.400 on that PC, the `INCLUDE` statement can be inserted in the `[User.<user name>]` section for the user who is currently logged on instead. The Mail Spooler will look at both the `[User.<user name>]` section and the `[UA]` section, and if one `INCLUDE` statement is found in each of the sections, both will be used.

The syntax of the `INCLUDE` statement is:

`INCLUDE-FILE= <command file name with path>`

`INCLUDE-RESPONSE=<response file name with path>`

`INCLUDE-FIRST= 1/0`

Parameters:

`<command file name with path>`: The file name of any legal UA-FI command file. Normally this file should contain a `FETCH` statement with a filter to fetch all messages intended for this application, and zero or more `SUBMIT` statements used to submit messages on behalf of the application.

`<response file name with path>`: The name of a file to receive the output from the UA-FI communication session. Normally, this file name should be defined with a placeholder for a sequence number, to avoid overwriting old communication results.

If the parameter to the `INCLUDE-FIRST` statement is 1, or the statement is not used, the included operations will be performed before any MailmaX.400 operations.

If the parameter to the `INCLUDE-FIRST` statement is 0, the included operations are performed after the standard MailmaX.400 operations.

The placeholder is defined with a maximum of 4 question marks. Examples:

`edi????.rsp` `edi001.rsp`, `edi0002.rsp`, `edi0003.rsp`, etc.

`edirsp.???` `edirdp.001`, `edirsp.002`, `edirsp.003`, etc.

(Avoid `.b??` if the messages are fetched to the same directory as the response file, as message BodyParts (attachments) automatically get `.b00`, `.b01` extensions).

The number will always start with the lowest free number, so your application should normally delete or rename each response file as it is processed to avoid processing the same response file more than once.

Notes on the UA-FI command file you need to use:

SUBMIT:

It is not necessary to include any SUBMIT statements if your application uses UA-FI directly for sending. If this is done, the INCLUDE Statement and the Command file it is pointing to can be completely static and installed by your application's installation program.

Fetch filters:

You should use a filter that selects **all** the messages that are intended for your application and **no** other messages. If you need to filter on more than one criterion, you can specify these in the same FETCH statement, or you can use several FETCH statements in the same INCLUDE file.

Fetch path:

If, in the path parameter to the FETCH statement, you give a path for the resulting messages, your application can be programmed always to check for “*.b??” files in this directory, and to delete each file when it has been processed. If you need to inspect the response file anyway to get delivery reports and notifications, you should use the FETCHED= statements in the response file to find new messages and files.

Multiple Fetches:

There is always a possibility that a message arrives in the user's mailbox after your FETCH statement has been initiated, and before the standard FETCH statements of MailmaX.400 are initiated. These messages will arrive in the user's INTRAY and not in your response file. To minimize the problem, always repeat your FETCH statement(s) in your command file. The last statement will normally return NONE, and, since this is a fast operation, the time interval in which messages can end up in the wrong place will be as small as possible.

To make sure no messages are lost, you should also define your document type in DOCMAGIC and supply a small program that can copy the file to your in-queue when the user double-clicks on the attachment.

Example:

In your installation job, install the following in the [UA] section:

```
INCLUDE-FILE : c:\EDI\edi.cmd
```

```
INCLUDE-RESPONSE : c:\EDI\edi???.rsp
```

And install the file c:\EDI\edi.cmd:

```
Fetch : <s=EDI;o=Fastspeed;a=TelemaX;c=no Delete> c:\EDI
```

```
Fetch : <s=EDI;o=Fastspeed;a=TelemaX;c=no> Delete c:\EDI
```

(Note that the same FETCH statement is repeated to minimize the interval between initiating the last FETCH and starting the normal MailmaX.400 fetch operation).

In your EDI application, you should check at startup if there are any files c:\EDI\edi???.rsp.

If there are one or more such files, process them, and delete them afterwards.

If you do not need the response file info, you can fetch the files (BodyParts) directly. (You can do this if your project uses the EDIFACT CONTROL messages and not the X.400 receipts, and you trust the user to re-send any messages that have not been confirmed by a CONTROL message within a reasonable time, and the EDI addresses inside the EDI Interchange are enough for your EDI application). To do this, simply check for files with the name "c:\EDI*.b??", and delete / rename each file when processed.

10. Address syntax

MaXware Simple MAPI accepts X.400 addresses (O/R-Names) compliant with a string format defined in the ITU/TS specification F.400 (1992).

The addresses consist of a list of address elements in the format:

keyword=key value;

Example:

“s=maxware;o=maxware;a=maxware;c=no”

for an address consisting of **S**urname: MaXware, **O**rganization: MaXware; **ADMD**: Telex and **C**ountry code: NO (Norway).

Keywords accepted:

C	Country
A	Administrative management domain name (service provider)
S	Surname
G	Given name
I	Initials
Q	Generation qualifier (to distinguish between users with identical names, for example father and son, using Sr. and Jr.)
X.121	The X.121-address (network address)
T-ID	Terminal identifier
P	Private management domain name
O	Organization name
OU1	Organization unit name 1
OU2	Organization unit name 2
OU3	Organization unit name 3
OU4	Organization unit name 4
N-ID	Numeric user identifier
T-ID	Terminal identifier
T-TY	Terminal type
DDA:	Domain-defined attribute 1 - 4. This part has the syntax DDA:type=value where “type” is the type, and “value” is the attribute itself. Both must contain only printable string characters (see below). You can use this, for example, to address users on a network that is connected to the X.400 service through a gateway: DDA:RFC-822=Harald(a)vax.delab.nth.edu
FREEFORM	Freeform name. Header elements only (primary, copy and p2_originator)

Special keywords used to access a “Physical Delivery” (postal) service:

CN	Common name
PD-PN	Physical delivery personal name
PD-EA	Physical delivery extension O/R name components

PD-ED	Physical delivery extension name components
PD-OFN	Physical delivery office number
PD-OF	Physical delivery office name
PD-O	Physical delivery organization name
PD-S	Street address
PD-A	Unformatted postal address
PD-U	Unique postal name
PD-L	Local postal attributes
PD-R	Poste restante address
PD-B	Post office box address
PD-PC	Postal code
PD-SN	PDS name
PD-C	Physical delivery country name

11. The MAXWARE.INI file

The file MAXWARE.INI contains the configuration for MailmaX.400 for Windows. The location and name of the MAXWARE.INI file is defined in the WIN.INI entry:

[MailmaX]

Initfile=<path and file name>

The MailmaX.400 installation program sets this path and file name, and they must NOT be changed. If they are changed, MailmaX.400 will not find all its files.

The various sections and parameters are documented in the manual “Configuration Guide for MaXware Products”.

12. Address and message templates

The FORMS.INI file contains address templates for MailmaX.400 for Windows:

The address templates are small “form” definitions used to make it easy for the user to fill in information for the various address types. There is one .FRM file for each service provider profile. Normally, the file contains one address template for each gateway on the server (fax, telex, Internet etc.), one address template for each (if any) query-by-mail service at the service provider, and one DefineUser template used for the “Create new user” dialog.

The file format of the <service profile>.FRM files is documented in the manual “Configuration Guide for MaXware Products”.

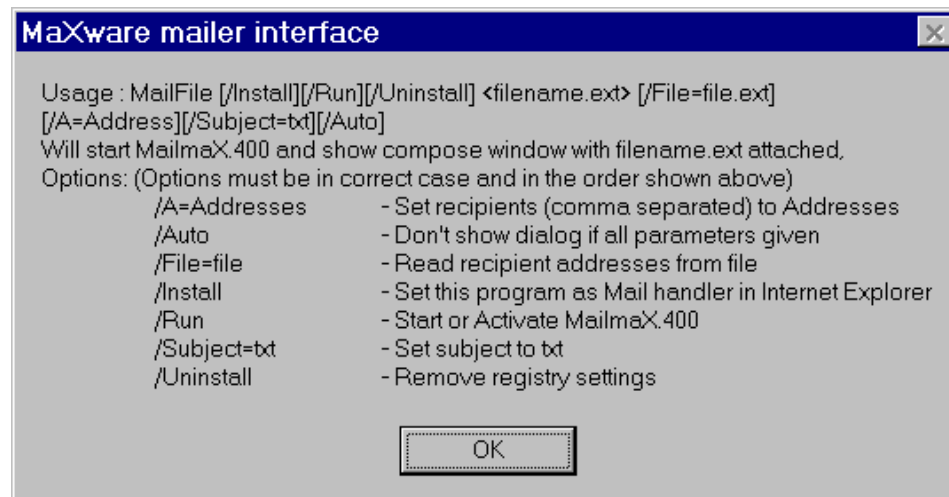
13. Appendix B: MailFile - A MailmaX.400 extension

13.1 Introduction

'Mailfile' is an integrated part of MailmaX.400 for Window. It defines a simple interface to enable other applications to submit a single file as an attachment to an outgoing message.

Integrated and installed with MailmaX.400, it makes it possible to use the 'Send to...' function in Windows Explorer to send any *single file* selected. It also builds a link between Microsoft Internet Explorer and MailmaX.400 so that when you click on a 'MailTo:' link, MailmaX.400 will be used as the email application.

The screenshot below shows what you will see if you start the MAILFILE.EXE program without any parameters on the command line.



13.2 Installation

There is no installation program! Just copy the program file to your MailmaX.400 directory. That's all.

The 32-bit version is automatically installed together with MailmaX.400. The (un-)install options shown in the screenshot apply only to the 32-bit version. This option will not (un-)install the program, but just update the Windows Registry setting to enable Explorer or Internet Explorer to use MailmaX.400 as its e-mail application.

The program must be manually installed as a desktop icon with properly defined command-line options if required.

To install MailFile as a desktop icon:

- Click with the right-mouse button on the Desktop and select "New/Shortcut".
 - In the "Create Shortcut" dialog you must fill in the "Command Line" field completely. (See below).
 - Click "Next" and select the name for the shortcut that will appear on the desktop.
 - Click "Finish".
-

13.3 MailFile – How it works

When MailFile is started, the command-line parameters is passed on to the program and executed. MailmaX.400 will be started automatically if it is not already running. The command-line options are described below.

13.3.1 Some command-line examples:

mailfile /Install

Update the 32-bit Windows registry so 'MailTo:' will work from Microsoft Internet Explorer.

mailfile /Uninstall

Remove the 32-bit Windows registry added using the /Install option.

mailfile /Run

Activate MailmaX.400 without sending anything.

mailfile c:\windows\win.ini

MailmaX.400's Compose window will be opened, without any subject field or recipients filled in. The file 'C:\WINDOWS\WIN.INI' will be attached to the message. The path is required only if the file is not in the default directory. The document is sent as TEXT or DATA, depending on the content of the file. This cannot be configured from the command line.

mailfile message.doc /File=names.txt /Subject=My Message/Auto

The file 'MESSAGE.DOC' in the default directory is attached to a message with the subject 'My message'. The recipient(s) of the message are read from the file 'NAMES.TXT'. If all recipients are correctly defined, the message is prepared for sending and the Compose window is closed. (See details about the syntax of the address file below).

mailfile message.doc /A=Office,G=James;S=Bond;P=007;A=Secret;C=ww /Subject=Secret

The file 'MESSAGE.DOC' is attached to a message addressed to the address-book nickname 'Office' and the complete X.400-address of a Mr. James Bond (who is not in the address book). The message subject is "Secret", and the Compose window is left open for the sender to include more recipients or text.

13.3.2 Syntax of the address-list file: (/File=)

The file is built up as a general Windows .INI file with sections, keywords and values. Every address has to be in a separate, unique section. The section name itself is not important for the mail file, but may be useful to external programs. Every section must have one 'addr=' keyword, and the value must have the same format as if you typed the addresses manually with the /A= option. All sections will be included in the recipient list for the message.

Example:

[1]

addr=Office

[2]

addr=G=James;S=Bond;P=007;A=Secret;C=ww/Subject=Secret

[n]

addr=

13.4 Hints

13.4.1 MailmaX.400 start-up:

MailmaX.400 will be started if it is not already running. The user will then have to type his or her user name and password. If you like, you can include the password in a special variable in MAXWARE.INI to perform an automatic login.

```
[Mailmax]
UserName=LoginName
Password= Password
```

This should be used with care, but in a secure environment this can be used for automating the sending of messages to an even greater extent.

13.4.2 Mail Spooler set-up

The Mail Spooler can be configured to connect to the Message Store every time there is one message ready to be sent. MailFile used with the /Auto option will generate such messages. It may be convenient for the user simply to use the MailFile icon, never needing to click on the telephone-pole icon in MailmaX.400.

13.4.3 Run minimized

The Mail Spooler will always run as an icon, but if you also start MailmaX.400 and the UA-FI communication window as icons, the operations here may go totally unattended by the user. This is also faster because no screen update will take place. To make MailmaX.400 and UA-FI run iconized, just minimize them when you see them, and they will appear as icons the next time.

13.4.4 Automatic recognition of SMTP addresses

The service-profile's FRM-file may be configured with an Internet gateway address, making it possible to type SMTP addresses directly in the address-string. This general MailmaX.400 feature can be used to simplify addressing when addresses are collected inside another application that is using this syntax.

13.5 Known problems

13.5.1 Same message sent twice

Starting MailmaX.400 and look-ups in the address-book may take a little too long for the MailFile application in some cases. Because the connection between MailFile and MailmaX.400 is based on DDE, the MailFile application may think that the submission failed, and try again, with the result that the same message/file is sent twice! To solve this problem, we recommend the following:

- Start MailmaX.400 manually instead of automatically.
- Use the full X400-addresses instead of nicknames.
- Be careful with the /Auto –option.

13.5.2 Original file-name not (always) maintained

An attached file will be copied to the temp directory before being attached to an outgoing message. As a result of this, the original file name is not used on the attachment if you look at the Message/Properties in your Outtray.
