Message Exchange Programmer's Guide

August, 2008

This manual describes how to customize Message Exchange through programming.

Revision/Update Information:	This is a revised manual.
Operating System and Version:	OpenVMS Alpha [tbd] or later
	OpenVMS Industry Standard 64 V8.2 or later
Software Version:	Message Exchange V6.0

Matthew Madison
Santa Cruz, California

27 August 2008

Contents

Preface

Message Exchange (MX) provides two forms of customization: an interface for a site-specific transport, and interfaces for modifying addresses. This manual describes how to write routines or programs to use these interfaces for customizing MX.

Intended Audience

This manual is intended for systems programmers who will be writing code to customize MX.

Document Structure

This guide consists of four chapters.

- Chapter 1 describes the SITE transport interface.
- Chapter 2 describes the address modification interfaces.
- Chapter 3 describes the SMTP authentication callout interface.
- Chapter 4 describes the character-conversion callout interface.
- Chapter 5 describes the SMTP client access callout interface.

Example code for all of these interfaces is provided in the Examples subset of the installation kit.

Related Documents

You can find additional information in the following documents:

- *Message Exchange Installation Guide* describes the installation of MX.
- *Message Exchange Management Guide* describes the management and operation of MX.
- *Message Exchange Release Notes* contain information and updates not included in this manual. The release notes are part of the software distribution kit.

1 The SITE Transport Interface

MX provides delivery agents and message entry agents for SMTP over TCP/IP and DECnet. If your site has some network transport that is not supported by MX, you can interface it with MX through the SITE transport interface.

1.1 The SITE Delivery Interface

When you use the MCP DEFINE PATH command to route mail to the SITE path, the MX_SITE delivery agent takes messages routed to that path and feeds them into a subprocess that executes a command procedure you must provide. The command procedure must be called MX_EXE:SITE_DELIVER.COM and must accept four parameters.

The first parameter is the ''route'' parameter, which is either the host name part of the address or the value of the /ROUTE qualifier from the DEFINE PATH command that routed the message to the SITE delivery agent. This parameter can be used to distinguish among several installed site-specific delivery agents, if needed.

The second parameter is the name of a temporary file that contains the message, including all of the RFC822 headers (corresponding to the DATA part of an SMTP transaction). The third parameter is the name of another temporary file that contains the recipient's address, corresponding to the RCPT TO addresses of an SMTP transaction. The fourth parameter is the RFC822 address of the originator of the message, corresponding to the MAIL FROM address of an SMTP transaction.

Your delivery procedure and the programs it invokes must not cause the subprocess to terminate, nor should they rely on specifics about the filenames provided. The procedure MUST exit with a success status code to let MX know that the message was delivered successfully. If there was an error in the delivery and you wish MX to return an error message to the sender, you should exit with a non-success status code. If the severity of the status is SEVERE (also called FATAL), the SITE delivery agent will return an error message to the sender. Otherwise, the message will be queued for another attempt. The MCP SET SITE command controls how many attempts will be made before the delivery agent gives up; the default is 96, with 30 minutes between each attempt.

Example 1–1 Sample SITE_DELIVER.COM

```
$! Simple SITE_DELIVER.COM which invokes a real program to do all the work.
$! This file must be placed in MX_EXE: for use with the MX SITE interface.
$! It is invoked by MX_SITE with:
$!
$!  @MX_EXE:SITE_DELIVER  route msg-file-spec dest-file-spec origin-address
$!
$! The originator address is stuck in a file since it can contain
$! characters that might confuse DCL when we invoke the delivery program.
$!
$! This is a simple procedure which ignores the "route" parameter.  If
$! you have multiple SITE delivery paths available, use the "route" parameter
$! to route the message to the appropriate delivery program.
$!
$ SET NOON
$!
$ DELIVER = "$my_mail_system:enter_message"
$!
$ CREATE my_temp_dir:SENDER.TMP
$ OPEN/APPEND TMP my_temp_dir:SENDER.TMP
$ WRITE TMP P4
$ CLOSE TMP
$!
$ DELIVER 'P2 'P3 my_temp_dir:SENDER.TMP
$ STAT = $STATUS
$ DELETE my_temp_dir:SENDER.TMP;*
$ EXIT 'STAT
```

1.2 SITE Message Entry

The SITE message entry program should be used by your SITE transport agent to enter a message into the MX message queue. The program is called MX_SITE_IN and resides in MX_EXE. It should be invoked as a DCL foreign command:

```
$ MX_ENTER = "$MX_EXE:MX_SITE_IN"
$ MX_ENTER  msg-file-spec  dest-file-spec  [origin-address]
```

MX_SITE_IN takes up to three parameters, which correspond exactly to the last three parameters passed out by the MX_SITE delivery agent. The first parameter should be the name of a file containing a properly-formatted RFC822 message. The second parameter should be the name of a file containing a list of RFC822 route addresses (they must have the surrounding angle brackets, just as in an SMTP transaction). The third parameter, which is optional, should be the RFC822 route address of the sender (also including the surrounding angle brackets). If the third parameter is omitted, the address of the user running the program will be used as the origin of the message.

MX_SITE_IN will automatically insert the following headers if they are not present in the message file:

- Date:
- From:, based on the sender specified on the command line or the user's e-mail address
- To:, based on the list of recipients (up to 1,024 characters maximum)
- Message-ID:

1.2.1 Non-Privileged Use of MX_SITE_IN

By default, MX_SITE_IN is not protected against WORLD access and is not installed with any privileges; only a user holding SYSPRV, SYSLCK, and EXQUOTA privilege may use the program. However, it is designed so that non-privileged users may use the program to enqueue messages that would be difficult to compose using VMS MAIL, such as MIME multipart messages.

To allow non-privileged users to invoke MX_SITE_IN, change its protection settings and use the INSTALL utility to install the image with privileges:

```
$ SET PROTECTION=W:RE MX_EXE:MX_SITE_IN.EXE
$ INSTALL CREATE MX_EXE:MX_SITE_IN -
_$              /PRIV=(SYSPRV,SYSLCK,EXQUOTA)
```

For users without SYSPRV enabled as process privilege, MX_SITE_IN ignores the third (origin-address) parameter, and inserts a Sender: header containing the user's actual address if a From: header is included in the message file.

2 Address Modification Interface

MX provides an interface for altering envelope addresses. This interface allows you to add your own routines for performing two different types of address modifications. For each type of modification, the routines must be part of a shareable library, which gets mapped into the appropriate parts of MX at run-time with LIB$FIND_IMAGE_SYMBOL.

The address modification routines are located through the use of logical names.

Logical name	Modification type
MX_SITE_ADDRESS_REWRITER	Modifying headers and envelope addresses for outgoing and incoming mail
MX_SITE_DOM_EXPANSION	Modifying or expanding host names
MX_SITE_NAME_CONVERSION	Translating local aliases or performing directory lookups

In each case, the logical name must be defined /SYSTEM/EXEC and must translate to the name of an image that has been linked /SHARE/NOTRACE and INSTALLed on the system. If you name the shareable images ADDRESS_REWRITER.EXE, DOMAIN_EXPANSION.EXE, and NAME_CONVERSION.EXE, respectively, and place them in the MX_ROOT:[EXE] directory, the MX startup procedure will automatically create the logical names and INSTALL the shareable images for you.

Examples of routines for performing address modifications are included in the directory MX_ROOT:[EXAMPLES] (if they have been installed).

2.1 Address Rewriting

The site address rewriter routines are called by the Router process to allow RFC822 header address rewrites on all outgoing mail, regardless of its origin, and on envelope addresses for incoming mail. The main purpose for these routines is to allow site-specific user and host naming conventions.

The name conversion routines, described in Section 2.3, provide a means for implementing user aliases, but it does not affect domain names. With the address rewriter routines, both usernames and host names can be modified. To ease mail delivery, many sites prefer to hide the various machines used at that site by supplying a generic site name for the address. For example, the generic domain MADGOAT.COM might be used for all addresses, even though the machines in use may be named HUNTER.MADGOAT.COM and MATT.MADGOAT.COM.

Sample address rewriter modules are provided in the Examples subset of the installation kit.

The routines that must be provided by an address rewriter are described on the following pages.

INIT

Initialization routine.

FORMAT

INIT *context*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

The INIT routine must return a success status value in order for the other address rewriting routines to be used.

ARGUMENTS

context
VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**
This is a longword passed by reference to your routine that you may use for any purpose, such as allocating a block of memory for keeping contextual information.

DESCRIPTION

This routine is called by the Router before any calls to the REWRITE_HEADER and REWRITE_ENVELOPE routines. You may use this routine to set up any context or perform any housekeeping tasks needed to prepare for the subsequent calls to the REWRITE_* routines.

Since your routines must be reentrant, you should not use static storage for keeping track of state information or other contextual information. Instead, you should allocate a block of dynamic memory and return its address in **context**.

REWRITE_HEADER

Routine to rewrite an address in an RFC822 header.

FORMAT

REWRITE_HEADER *context, inaddr, outaddr, header_code*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

To indicate a successful rewrite, return SS$_NORMAL or some other success status code. If you do not return a success status code, the caller will assume that the rewrite did not occur.

ARGUMENTS

context

VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**
This is the same value that was passed to the INIT routine.

inaddr

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor (fixed-length)**
The RFC822 header address to be rewritten.

outaddr

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**
A string into which your routine should copy the rewritten address, if expansion was successful. You must use the STR$ string routines (such as STR$COPY_DX) to copy the string into this argument.

header_code

type: **longword (unsigned)**
access: **read only**
mechanism: **by value**
A value representing one of the following RFC822 header types:

Description	Symbolic name	Value
From:	MX_K_HDR_FROM	1
Sender:	MX_K_HDR_SENDER	2
To:	MX_K_HDR_TO	3
Resent-To:	MX_K_HDR_R_TO	4
CC:	MX_K_HDR_CC	5
Resent-CC:	MX_K_HDR_R_CC	6
BCC:	MX_K_HDR_BCC	7
Resent-BBC:	MX_K_HDR_R_BCC	8
Reply-To:	MX_K_HDR_REPLY_TO	17
Resent-Reply-To:	MX_K_HDR_R_REPLY_TO	19
Resent-From:	MX_K_HDR_R_FROM	20
Resent-Sender:	MX_K_HDR_R_SENDER	21

The symbolic names are defined in MX_HDR.H in MX_ROOT:[EXAMPLES], if you installed the examples.

DESCRIPTION

This routine is called to rewrite an address appearing in an RFC822 header on all outgoing mail, regardless of its origin. The address of the context block you allocated in the INIT routine is passed in here for any information you need to keep track of between calls. This routine may be called more than once between one pair of INIT and CLEANUP calls.

REWRITE_ENVELOPE

Routine to rewrite an RFC821 envelope address.

FORMAT

REWRITE_ENVELOPE *context, inaddr, outaddr*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

To indicate a successful rewrite, return SS$_NORMAL or some other success status code. If you do not return a success status code, the caller will assume that the rewrite did not occur.

ARGUMENTS

context

VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**
This is the same value that was passed to the INIT routine.

inaddr

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor (fixed-length)**
The RFC821 envelope address to be rewritten. RFC821 addresses are enclosed in angle brackets (<>). For example, ‘‘<GENE@MADGOAT.COM>’’ is a valid RFC821 envelope address.

outaddr

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**
A string into which your routine should copy the rewritten address, if expansion was successful. You must use the STR$ string routines (such as STR$COPY_DX) to copy the string into this argument.

Note: The rewritten address *must* be a valid RFC821 address, including the angle brackets.

DESCRIPTION

This routine is called to rewrite an RFC821 envelope address on incoming mail. Envelope addresses are the addresses of the actual recipients of incoming mail and may or may not correspond directly to the RFC822 headers.

The address of the context block you allocated in the INIT routine is passed in here for any information you need to keep track of between calls. This routine may be called more than once between one pair of INIT and CLEANUP calls.

REWRITE_VIRTUAL_DOMAIN

Routine for indicating whether a domain name is considered a "virtual" local domain.

FORMAT

REWRITE_VIRTUAL_DOMAIN *context, domname*

RETURNS

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

This routine should return 1 if the domain name is to be considered local, or 0 if it is to be considered remote for the purposes of SMTP anti-relay checking.

ARGUMENTS

context

VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**
This is the same value that was passed to the INIT routine.

domname

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor (fixed-length)**
A domain name to be checked for virtual local domain status.

DESCRIPTION

This routine is called by the SMTP server to find out whether or not a domain name should be considered local for relay-checking purposes. This callout routine can be used to extend the built-in LOCAL_DOMAIN list checking, normally done by the SMTP server, to include those domains managed by the ADDRESS_REWRITER callout. Such domain names are called ''virtual'' local domains.

This routine is optional, and is called only if provided by the ADDRESS_REWRITER callout module.

CLEANUP

Context cleanup routine.

FORMAT

CLEANUP *context*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

This routine should return a status value indicating the success or failure of the cleanup operation. The caller may or may not ignore the returned value.

ARGUMENTS

context
VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**
The address of the context block you allocated in the INIT routine.

DESCRIPTION

This routine is called to clean up after a series of REWRITE_* calls. You should clean up the context information and deallocate the context block allocated by the INIT routine.

If you did not allocate a context block in the INIT routine, you must still have a CLEANUP routine, even if it just returns to the caller.

2.2 Host Name Expansion

The site host name routines are called by the Router process just before path identification. The main purpose for these routines is to expand abbreviated host names into full host names that will properly match one of the paths defined in the MX configuration file.

When you install SMTP support with MX, host name expansion routines are automatically provided that call on the underlying TCP/IP package to catch abbreviated host names that might be recognized by the TCP/IP name resolver but are not defined in the MX configuration file. The source for these routines is included in MX_ROOT:[EXAMPLES] for you to modify if needed.

The routines that must be provided by a host name expander are described on the following pages.

INIT

Initialization routine.

FORMAT

INIT *context*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

The INIT routine must return a success status value in order for the other expansion routines to be used.

ARGUMENTS

context

VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

This is a longword passed by reference to your routine that you may use for any purpose, such as allocating a block of memory for keeping contextual information.

DESCRIPTION

This routine is called by the Router before any calls to the EXPAND routine. You may use this routine to set up any context or perform any housekeeping tasks needed to prepare for the subsequent calls to EXPAND.

Since your routines must be reentrant, you should not use static storage for keeping track of state information or other contextual information. Instead, you should allocate a block of dynamic memory and return its address in **context**.

EXPAND

Routine to expand a host name.

FORMAT

EXPAND *context, hostname, expname*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

To indicate a successful expansion, return SS$_NORMAL or some other success status code. If you do not return a success status code, the caller will assume that expansion did not occur.

ARGUMENTS

context

VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**
This is the same value that was passed to the INIT routine.

hostname

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor (fixed-length)**
The host name to be expanded.

expname

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**
A string into which your routine should copy the expanded host name, if expansion was successful. You must use the STR$ string routines (such as STR$COPY_DX) to copy the string into this argument.

DESCRIPTION

This routine is called to perform a host name expansion. The address of the context block you allocated in the INIT routine is passed in here for any information you need to keep track of between calls. This routine may be called more than once between one pair of INIT and CLEANUP calls.

CLEANUP

Context cleanup routine.

FORMAT

CLEANUP *context*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

This routine should return a status value indicating the success or failure of the cleanup operation. The caller may or may not ignore the returned value.

ARGUMENTS

context

VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**
The address of the context block you allocated in the INIT routine.

DESCRIPTION

This routine is called to clean up after a series of EXPAND calls. You should clean up the context information and deallocate the context block allocated by the INIT routine.

If you did not allocate a context block in the INIT routine, you must still have a CLEANUP routine, even if it just returns to the caller.

2.3 Name Conversion

The local name conversion routines are used by the MX_MAILSHR VMS Mail interface to translate a username into an alias and by the Router to translate aliases back into real usernames. This can be used, for example, to map usernames into ‘‘real’’ names and vice-versa.

The following pages describe the routines that must be provided for the name conversion interface.

In addition to the required CONVERT routine, an optional FULL_CONVERT routine may be provided to allow for conversion of a username to a full RFC822 address, as opposed to just username substitution.

INIT

Initialization routine.

FORMAT

INIT *context*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

The INIT routine must return a success status value in order for the other expansion routines to be used.

ARGUMENTS

context

VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

This is a longword passed by reference to your routine that you may use for any purpose, such as allocating a block of memory for keeping contextual information.

DESCRIPTION

This routine is called by the Router before any calls to the CONVERT routine. You may use this routine to set up any context or perform any housekeeping tasks needed to prepare for the subsequent calls to CONVERT.

Since your routines must be reentrant, you should not use static storage for keeping track of state information or other contextual information. Instead, you should allocate a block of dynamic memory and return its address in **context**.

CONVERT

Routine to convert a username to an alias or an alias to a username.

FORMAT

CONVERT *context, code, inname, outname*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

On successful conversion, return SS$_NORMAL or some other success status code. If you do not return a success status code, the caller will assume that expansion did not occur.

ARGUMENTS

context

VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**
This is the same value that was passed to the INIT routine.

code

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**
This argument indicates what type of name conversion should occur. It will have one of the following values:

Value	Meaning
1	Perform alias-to-username conversion.
2	Perform username-to-alias conversion.

inname

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor (fixed-length)**
The name to be converted.

outname

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

A string into which your routine should copy the result. This is only used if you return a success status code.

DESCRIPTION

This routine is called to perform a name conversion. For alias-to-username translation, a string containing the potential alias is passed in **inname**. If the conversion succeeds, the address returned in **outname must** be in RFC821 format:

<username@hostname>

This format must be used even if the address is intended for the local host.

For username-to-alias conversion, the username to be converted is passed in **inname**. If no conversion is performed, return a non-success status code; otherwise, provide a result in **outname**. The result should be only the local part of an address; no host name should be appended nor any punctuation added.

This routine may be called more than once between one pair of INIT and CLEANUP calls.

CLEANUP

Context cleanup routine.

FORMAT

CLEANUP *context*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

This routine should return a status value indicating the success or failure of the cleanup operation. The caller may or may not ignore the returned value.

ARGUMENTS

context
VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**
The address of the context block you allocated in the INIT routine.

DESCRIPTION

This routine is called to clean up after a series of CONVERT calls. You should clean up the context information and deallocate the context block allocated by the INIT routine.

If you did not allocate a context block in the INIT routine, you must still have a CLEANUP routine, even if it just returns to the caller.

FULL_CONVERT

Routine to convert a username to an alias (as a full RFC822 address).

FORMAT

FULL_CONVERT *context, code, inname, outname*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

On successful conversion, return SS$_NORMAL or some other success status code. If you do not return a success status code, the caller will assume that conversion did not occur.

ARGUMENTS

context

VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**
This is the same value that was passed to the INIT routine.

code

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**
This argument indicates what type of name conversion should occur. Only the following value should be accepted:

Value	Meaning
2	Perform username-to-alias conversion.

All other values for this argument are reserved for future use.

inname

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor (fixed-length)**
The name to be converted.

outname

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

A string into which your routine should copy the result. This is only used if you return a success status code.

DESCRIPTION

This routine is called to perform a username-to-full-address conversion. The username to be converted is passed in **innname**. If no conversion is performed, return a non-success status code. Unlike the CONVERT routine, the result you provide in **outname** on a successful conversion **must** be a full RFC822-type address (user@host format).

This routine may be called more than once between one pair of INIT and CLEANUP calls, and may be intermixed with CONVERT calls.

3 SMTP Authentication Callout

The MX SMTP server supports the SMTP extension for authentication of clients, with built-in support for a private authentication database (when using the CRAM-MD5 authentication mechanism) as well as authentication using the VMS user authorization file (when using the PLAIN and LOGIN authentication mechanisms).

The authentication callout provides a way to replace the built-in PLAIN/LOGIN authentication mechanism, so that usernames and passwords may be validated against an alternative source besides the SYSUAF, or to extend the authentication system to include additional features (such as customized intrusion detection and evasion or per-message accounting).

The authentication callout routines are located through the use of the logical name MX_SITE_SMTP_AUTHENTICATION, which should be defined in executive mode in the system logical name table to point to the shareable image containing the callout module.

An example of an authentication callout module is provided in the Examples saveset of the installation kit.

INIT

Initialization routine.

FORMAT

INIT *context*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

The INIT routine must return a success status value in order for the other authentication routines to be used.

ARGUMENTS

context
VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**
This is a longword passed by reference to your routine that you may use for any purpose, such as allocating a block of memory for keeping contextual information.

DESCRIPTION

This routine is called by the SMTP server before any calls to the AUTHENTICATE or ACCOUNTING routines. You may use this routine to set up any context or perform any housekeeping tasks.

Note that the SMTP server is multi-threaded, so while this routine is called no more than once per SMTP session, there may be multiple simultaneous sessions active in the SMTP server at one time. Since your routines must be reentrant, you should not use static storage for keeping track of state information or other contextual information. Instead, you should allocate a block of dynamic memory and return its address in **context**.

AUTHENTICATE

Routine to authenticate a client.

FORMAT

AUTHENTICATE *context, usrnam, pass, cliaddr, cliaddrlen, sessid, authstat, astadr, astprm*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

The return status is used only to determine whether this routine has completed synchronously (non-success status) or asynchronously (success status).

ARGUMENTS

context

VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**
The context longword, as set by the INIT routine.

usrnam

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor (fixed-length)**
The username provided by the client.

pass

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor (fixed-length)**
The password provided by the client.

cliaddr

VMS Usage: **sockaddr**
type: **socket address**
access: **read only**
mechanism: **by reference**
A socket address structure containing the address of the client.

cliaddrlen

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

The size, in bytes, of the *cliaddr* socket address structure.

sessid

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

A ‘‘session identifier’’ to be returned for this authenticated SMTP session, if authentication is successful. If set to a non-zero value, the SMTP server will include this value in the Received: header of any message sent by the client during this session.

authstatus

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

A VMS status code indicating the success or failure of the authentication sequence. If this routine completes synchronously, this value should be set before returning to the caller. Typical values for *authstatus* are SS$_NORMAL for success, and SS$_INVLOGIN for failure.

astadr

VMS Usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

The address of the caller’s AST completion routine. If the AUTHENTICATE routine needs to perform asynchronous I/O, its own AST completion routine should call this routine to indicate to the caller that it is finished.

astprm

VMS Usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Argument to be passed to the caller’s AST completion routine.

DESCRIPTION

This routine performs authentication for a username/password combination. If this routine requires any I/O operation that may not complete immediately, it should use asynchronous I/O and its AST completion routine should call the AST routine that is passed in by the caller.

Only one authentication request will ever be outstanding for a single authentication context, so the context block can be used to store the caller's AST routine address, AST parameter, and authentication status address for later use by its AST completion routine.

Note that the SMTP server provides the username and password *exactly* as sent by the client. No case conversion, blank stripping, or other editing is done by the server.

ACCOUNTING

Routine to record the acceptance of a message in an accounting log.

FORMAT

ACCOUNTING *context, sessid, msgsize, fromadr, toadr*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

The SMTP server currently ignores the returned status. For future compatibility, however, you should return a status code indicating the success or failure of the accounting request.

ARGUMENTS

context

VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

This is the same value that was passed to the INIT routine.

sessid

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

This argument contains the session ID value that was returned by the AUTHENTICATE routine.

msgsize

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

The size, in bytes, of the body of the message sent by the client.

fromadr

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The SMTP MAIL FROM: address provided by the client for this message.

toadr

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The SMTP RCPT TO: address provided by the client for this message. Note that this routine is called once for each recipient named by the client.

DESCRIPTION

This routine is called by the SMTP server after a message is accepted from the authenticated client. Note that this routine may be called multiple times for each message—once per recipient address.

Note that this routine is optional; it is only called by the SMTP server if it is provided by the installed authentication callout module.

CLEANUP

Context cleanup routine.

FORMAT

CLEANUP *context*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

This routine should return a status value indicating the success or failure of the cleanup operation. The caller may or may not ignore the returned value.

ARGUMENTS

context
VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**
The address of the context block you allocated in the INIT routine.

DESCRIPTION

This routine is called to clean up at the end of an authentication sequence or session. You should clean up the context information and deallocate the context block allocated by the INIT routine.

If you did not allocate a context block in the INIT routine, you must still have a CLEANUP routine, even if it just returns to the caller.

4 Character-Conversion Callout

The MX VMS MAIL interface supports conversion between the local native character set and a character set used in e-mail messages. MX has built-in support for using the ISO-Latin-1 character set for network messages, with either ISO-Latin-1 or the DEC Multinational Character Set as the local, native character set.

If you need to use any other character sets for either local or network messages, you should install a character conversion callout module. Example code for this module is provided in the Examples saveset of the installation kit.

Note: The routines in this module must be reentrant, as a single process may have multiple conversion sequences active simultaneously.

INIT

Initialization routine.

FORMAT

INIT *code, context, [lcslen], lcsname, [ncsname] [,usrnam]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

If a success status is returned, the CONVERT routine will be called to perform the conversion.

ARGUMENTS

code

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

A coded value indicating the direction of the conversion. Possible values are:

Value	Description
1	Conversion from local character set to network character set.
2	Conversion from network character set to local character set.

Note that this value is passed by reference.

Arguments

context

VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

This is a longword passed by reference to your routine that you may use for any purpose, such as allocating a block of memory for keeping contextual information.

lcslen

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

If provided by the caller, the INIT routine should write the length of the local character set name in this parameter. If this parameter is null, the caller does not need the length returned.

lcsname

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

The name of the local character set. This should be one of the character set names registered with the Internet Assigned Numbers Authority, preferably the ‘‘MIME preferred’’ charset name.

Note: This argument and *lcslen* should be filled in even if this routine returns a non-success status value.

ncsname

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The name of the network character set to be used for the conversion. If this parameter is null, the callout module’s default network character set should be used. Otherwise, this routine should make sure that it can support conversion between the specified character set and the local character set.

usrnam

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

The username of the user for whom the character-set conversion is being performed, provided for per-user customization of character set usage. This argument is provided only during conversion of message contents; VMS MAIL header conversions always use the system-wide defaults. Check the argument count and test for the pointer to the descriptor being NULL before attempting to access this argument.

For users that use POP or IMAP clients to read their e-mail, you may want to defeat the character set conversion on local message delivery, especially if the POP/IMAP clients cannot handle the character set used on the VMS system.

DESCRIPTION

This routine is called by the MX VMS MAIL interface (in MX_MAILSHR, the MX Router agent, and the MX Local agent) to initialize a character conversion sequence.

This routine should validate the parameters provided and return success status only if the callout will handle the requested conversion. If the network and local character sets are identical, this routine should return a non-success status.

Any context required for the conversion should be allocated by this routine. The context should store information about the direction of the conversion, provided in the *code* argument.

CONVERT

Perform character-set conversion on a string.

FORMAT

CONVERT *context, instr, [outlen], outstr, [converted], [remaining]*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

A success value indicates that characters have been converted and/or copied between the input and output strings.

ARGUMENTS

context

VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**
The context longword, as set by the INIT routine.

instr

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor (fixed-length)**
The string containing characters to be converted.

outlen

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**
If non-null, the actual length of the converted output string should be returned in this argument.

outstr

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor (fixed-length)**
The output string. The descriptor for this string will always point to a fixed-length string, either static (DSC$K_CLASS_S) or dynamic (DSC$K_CLASS_D).

converted

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

If non-null, this argument should be set to 1 if there any characters in the input string were translated before being written to the output string. If the conversion resulted in a simply copy of identical characters, this argument should be set to zero.

remaining

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

If non-null, this argument should be set to the number of characters in the input string that were not processed by this call. This should only happen if the output string is not large enough to hold the entire converted input string.

DESCRIPTION

This routine is called to perform character set conversion on a string.

GETCSNAME

Returns a character set name.

FORMAT

GETCSNAME *context, code, [len], name*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

context

VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

The context that was returned by the INIT routine.

code

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

This argument indicates which character set name should be returned:

Value	Description
1	The local character set.
2	The network character set.

Note that the returned name should be fore the actual character set involved in the current conversion sequence, not the default character set supported by the callout.

len

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference**

If non-null, this argument should be set to the length, in bytes, of the returned name.

name

VMS Usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor (fixed-length)**

This argument should be set to the name of the requested character set.

DESCRIPTION

This routine is called during a conversion sequence to retrieve the name of either the local or network character set being used.

FINISH

Context cleanup routine.

FORMAT

FINISH *context*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

context
VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**
The address of the context block allocated in the INIT routine.

DESCRIPTION

This routine is called to clean up at the end of a conversion sequence. You should clean up the context information and deallocate the context block allocated by the INIT routine.

If you did not allocate a context block in the INIT routine, you must still have a FINISH routine, even if it just returns to the caller.

5 SMTP Client Access Callout

The MX SMTP server normally accepts all incoming connections from any client, regardless of the client's source address. The client access SITE callout provides a way to modify this behavior and exercise administrative control over which clients have access to the server.

The client access callout routines are located through the use of the logical name MX_SITE_CLIENT_ACCESS_CHECK, which should be defined in executive mode in the system logical name table to point to the shareable image containing the callout module.

An example of a client access callout module is provided in the Examples saveset of the installation kit.

INIT

Initialization routine.

FORMAT

INIT *context*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

The INIT routine must return a success status value in order for the other client access check routines to be used; otherwise, the server assumes that access is granted.

ARGUMENTS

context
VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**
This is a longword passed by reference to your routine that you may use for any purpose, such as allocating a block of memory for keeping contextual information.

DESCRIPTION

This routine is called by the SMTP server before any calls to the CHECK routine. You may use this routine to set up any context or perform any housekeeping tasks.

Note that the SMTP server is multi-threaded, so while this routine is called no more than once per SMTP session, there may be multiple simultaneous sessions active in the SMTP server at one time. Since your routines must be reentrant, you should not use static storage for keeping track of state information or other contextual information. Instead, you should allocate a block of dynamic memory and return its address in **context**.

CHECK

Routine to check access for a client.

FORMAT

CHECK *context, cliaddr, cliaddrlen, accstat, astadr, astprm*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

The return status is used only to determine whether this routine has completed synchronously (non-success status) or asynchronously (success status).

ARGUMENTS

context

VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**

The context longword, as set by the INIT routine.

cliaddr

VMS Usage: **sockaddr**
type: **socket address**
access: **read only**
mechanism: **by reference**

A socket address structure containing the address of the client.

cliaddrlen

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

The size, in bytes, of the *cliaddr* socket address structure.

accstatus

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

A VMS status code indicating the success or failure of the access check. If this routine completes synchronously, this value should be set before returning to the caller. Typical values for *accstatus* are SS$_NORMAL for success, and SS$_INVLOGIN (or any other non-success status) for failure.

astadr

VMS Usage: **ast_procedure**
type: **procedure entry mask**
access: **call without stack unwinding**
mechanism: **by reference**

The address of the caller's AST completion routine. If the CHECK routine needs to perform asynchronous I/O, its own AST completion routine should call this routine to indicate to the caller that it is finished.

astprm

VMS Usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Argument to be passed to the caller's AST completion routine.

DESCRIPTION

This routine checks whether the SMTP client at the specified address should be granted access to the SMTP server. If this routine requires any I/O operation that may not complete immediately, it should use asynchronous I/O and its AST completion routine should call the AST routine that is passed in by the caller.

Only one access check request will ever be outstanding for a single access check context, so the context block can be used to store the caller's AST routine address, AST parameter, and authentication status address for later use by its AST completion routine.

CLEANUP

Context cleanup routine.

FORMAT

CLEANUP *context*

RETURNS

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

This routine should return a status value indicating the success or failure of the cleanup operation. The caller may or may not ignore the returned value.

ARGUMENTS

context
VMS Usage: **context**
type: **longword (unsigned)**
access: **modify**
mechanism: **by reference**
The address of the context block you allocated in the INIT routine.

DESCRIPTION

This routine is called to clean up after an access check. You should clean up the context information and deallocate the context block allocated by the INIT routine.

If you did not allocate a context block in the INIT routine, you must still have a CLEANUP routine, even if it just returns to the caller.