Meridian

Meridian ACCESS

Application Programming Interface (API) Reference Manual

Publication number:	555-7001-317
Product release:	10.0
Document release:	Standard 1.0
Date:	August 1995

Publication history

August 1995

Manual released as Standard 1.0. This version documents Release 10.0 of Meridian Mail and Release 2.0 of Meridian ACCESS. his edition makes all previous editions obsolete.

March 1994

Manual released as Standard. This version documents Release 9 of Meridian Mail. This edition makes all previous editions obsolete.

Contents

Figures

About this guide

This guide describes Release 2.0 of Meridian ACCESS. Meridian ACCESS allows computers to connect to and interact with a Meridian Mail voice messaging system.

The Application Programming Interface (API) library provides a C-language procedural interface allowing UNIX-based workstations to use Meridian Mail voice services.

The API lets a workstation control voice messaging, file access, call processing, administration and other functions of Meridian Mail.

This reference manual describes the use of the API library functions. More information on programming Meridian ACCESS applications can be found in the Meridian ACCESS *Developer's Guide* (NTP 555–7001–316).

Chapter 1: Meridian ACCESS overview

Meridian ACCESS is a hardware and software option that provides UNIX workstations access to many features of the Meridian Mail voice messaging system. Some familiarity with the purpose and features of Meridian Mail is assumed in this document.

This chapter describes the software environment necessary for the operation of Meridian ACCESS. More information on this topic can be found in the Meridian ACCESS *Developer's Guide* (NTP 555-7001-316).

Note: This guide refers to Product release 2.0 of Meridian ACCESS.

Meridian Mail software

The Meridian Mail server runs its normal voice messaging software, one component of which is a module called a "toolkit." Each toolkit logically connects to a running process on the UNIX workstation; such a connection is called a "session." The toolkit accepts commands from the process and performs the necessary operations. When the operations are complete, the toolkit sends a response to the process.

External Notification Service

The External Notifiction Service (ENS) server is run on the first node of a Meridian Mail system with an ACCESS link. The server allows an application process on the UNIX host to establish itself as an ENS client to track the status of designated mailboxes on the Meridian Mail system. ENS is essential to the following classes of applications:

- paging systems,
- personal communications systems
- desktop messaging

The ENS client is notified when someone has logged out of a mailbox (indicating that the status of the mailbox may have changed), and when a new message has arrived in the mailbox.

API software

Meridian ACCESS applications link to the Application Programming Interface (API) library during compilation. The API library provides most of the functionality of the Meridian Mail voice services while hiding all of the details of communications and conversions.

The library consists of a set of functions for workstations running UNIX. The library functions must be linked with an applications module to provide an executable program which will run as a UNIX process.

Chapter 2: Meridian Mail facilities

The Meridian Mail server is an interface to the telephone switch and also provides several of its own services. Through the Meridian Mail toolkit, a UNIX process can control both the call-processing capabilities of the telephone switch and the voice–messaging capabilities of the Meridian Mail server.

The External Notification Server (ENS) server is run on the first node of a Meridian Mail system with an ACCESS link. The server allows an application process on the UNIX host to establish itself as an ENS client to track the status of designated mailboxes on the Meridian Mail system.

Functions available

The API library provides the following types of functions:

- local functions
- Link Handler functions
- resource management functions
- telephony functions
- file access functions
- voice operations functions
- messaging functions
- voice segment file functions
- External Notification Services
- user administration functions

- event handling functions
- high-level functions

Meridian Mail architecture

The Meridian Mail server is a special purpose computer with an operating system, applications programs, and special telephony interfaces.

Meridian Mail architecture contains several multi-user computer concepts which are mirrored in the API library. This section describes the concepts and data types found in the Meridian Mail architecture, and in the API library.

Accounts

As a multi-user computer system, the Meridian Mail server incorporates the user account concept. Meridian Mail resources and capabilities are allocated to accounts by the Meridian Mail administrator. The resources available include voice file storage and the setting of maximum voice file length; capabilities include password protection and the ability to send messages. To gain access to an account, a user must log on to the Meridian Mail server.

Cabinets and Files

Each user’s account on Meridian Mail has access to a file cabinet, also known as a mailbox. The cabinet contains all of the files available to that user.

Every file in a cabinet has a type. The type may be “voice message,” “voice segment file,” or “simple voice file.” Files have names, subject descriptions, creation times, and other identifying characteristics. See the Meridian ACCESS *Developer’s Guide* (NTP 555-7001-316) for more information on Meridian Mail files.

File names in Meridian Mail are not necessarily unique. To distinguish between two files which have the same name (a rare occurrence), it is possible to specify a file by its position. A file position is assigned to each file when the cabinet–level file–retrieval operations are performed. (See the File Access section for further information.)

File names in Meridian Mail are strings with a maximum length of 13 characters. Use only alphanumeric characters for file names since some applications reserve non-alphanumeric characters for special purposes.

Some Meridian Mail software creates files which are placed in a user’s cabinet. Voice messaging software, for example, creates voice message files.

Voice messages have standard file names beginning with “v” followed by the date and time of creation. Files delivered by the Message Transfer Agent (MTA) have names beginning with “m” because the MTA changes the ‘‘v” (Voice Messaging) to “m” (MTA) as indication of delivery. The year, month, day, hour, minute, and second are each specified by two characters. An example of a standard file name is v950102090030 which means that the voice messaging file was created at 09:00:30 hours on January 2, 1995.

DNs and Mailboxes

A Directory Number (DN) is a telephone number. This is the number you dial to reach a person by telephone. In a typical office, it would consist of the four–digit extension number of a co worker, but it could also be an eleven-digit long-distance phone number.

If a Meridian 1 PBX is being used, the DN can contain optional pause characters, provided that the DN begins with a Trunk Access Code configured for the Meridian 1. The Trunk Access Code is the number which must be dialed to gain access to an outside line from the PBX. The asterisk (*) is the pause character and results in a pause of approximately two seconds each time it is encountered. For example, if 9 is a Trunk Access Code for the Meridian 1, then specifying the DN 9*5551212 will result in a two second pause before the seven digit external telephone number is dialed.

Meridian Mail also incorporates the mailbox concept. A mailbox is the number you use to leave a voice message for another person. It usually looks like a local DN. Message addressing is also supported to AMIS or proprietary network destinations if these features are present on your system.

In many organizations, a person’s DN and mailbox are the same number. This is typical of organizations that have one phone per person.

In some organizations, people share phones, so there must be some way to leave a message for a specific person. In such organizations, several people may have the same phone number but different mailbox numbers.

The API library functions distinguish between mailbox numbers and DNs.

Meridian ACCESS only supports eight-digit mailboxes. This means that an ACCESS application that attempts to log in to a mailbox longer than eight digits will fail.

However, to maintain Meridian Mail compatibility with 18-digit mailboxes, a Meridian ACCESS application is able to compose and send messages to a user with an 18-digit mailbox. Mailbox information that is returned in an application programming interface (API) is also correct up to 18-digits.

VMUIF Mailboxes

VMUIF is set on a customer basis, and it is possible to add a VSDN entry for Meridian ACCESS in a VMUIF customer. However, VMUIF mailboxes are not supported through Meridian ACCESS.

Voice Channels

A voice channel is a Meridian Mail server resource which provides facilities for voice and telephony functions.

Functions are provided in the API library to issue commands to a voice channel. Such commands include placing a call to a DN, playing a voice message on the channel, recording from a voice channel into a file, and transferring an existing call to a new DN.

Date and Time

Dates and times are always stored in a structure of six elements. The structure is

```
struct DATE {
   short Year;                   /* e.g.,1987 */
   char  Month;                  /* 1..12 */
   char  Day;                    /* 1..31 */
   char  Hour;                   /* 0..23 */
   char  Minute;                 /* 0..59 */
   char  Second;                 /* 0..59 */
   };
```

Among other things, dates are used for specifying or reading the delivery times of voice messages, and for reading the current time as set on the Meridian Mail server.

User Names

Meridian Mail offers a flexible format for specifying a user name as input to an operation. A user name is a C character string which can be specified in any of the following formats:

Format	**Example**
<First Name>	“John”
<Last Name>	“Smith”
<First Name> <Last Name>	“John Smith”
<First Initial> <Last Name>	“J Smith” or “J. Smith”
<Last Name>, <First Name>	“Smith, John”
<Last Name>, <First Initial>	“Smith, J’ or “Smith, J.”

Case is not important when specifying a user name.

Wildcards

Meridian Mail also supports the following three special wildcard characters which can be used when specifying a user name:

Special character	**Matches**
_ (underscore)	any single character
+ (plus)	any sequence of zero or more characters
? (question mark)	phonetically

Example

“mil_er”	matches	“David Miller” or “John Milner”
“park+”	matches	“Jane Park” or “Brian Parker”
“mclean?”	matches	“Mary McLean” or “Don MacLean” or “Thomas McLain”
“rob+t+”	matches	“Robert Jones” or “Michael Robertson”
“j sm_th+”	matches	“Jill Smith” or “Jack Smythe”
“h+s_n, +ed”	matches	“Ed Hansen” or “Fred Henderson”

Note: The “+” and “_” wildcards may appear together anywhere in the same name. However, all “+” and “_” wildcards will be ignored if a “?” character appears anywhere in the name.

Chapter 3: Meridian ACCESS functions

This chapter describes the functions available in the Meridian ACCESS API library. Many functions provide commands to perform voice messaging, call processing, and general file handling operations.

Naming conventions

You can identify some function categories in the Meridian ACCESS library by function names. For example, the functions in the Voice Operations section contain the word "Voice" in their names. The following table matches function groups to function names.

Function	Naming conventions
Local Functions	(various)
Link Handler functions	(various)
Resource Management	(various)
Telephony	Call
File Access	Cabinet, File
Voice Operations	Voice
Messaging	Msg, Addr
Voice Segment File	Seg
Administration	(various)
Event Handling	On, Event
High-level Functions	(various)
External Notification Services	(various)

All Meridian ACCESS functions begin with the two-character prefix “m_”. This helps to avoid name clashes with functions that may exist in other libraries. Internal functions (those which are not called directly by an application) use the three-character prefixes “mu_” or “nt_”.

Functions which have the form “m_...Pattern()” and “m_Retrieve...()” always appear in pairs. The former is a selection function which allows a pattern to be specified and the latter is the retrieval function which collects the specified items. An example is the function pair “m_FilePattern()” and “m_RetrieveFile()”. These may be used together to retrieve information on all of the files in a cabinet.

Functions which end with the character “N” are the numeric, menu-oriented versions of the file-handling functions. An example is “m_OpenFileN()”, which opens a file by menu number rather than by name.

Functions which end in “X” are extended versions of simpler functions. An example is “m_PlayVoiceX()”. This function provides more control over the playback of a voice file than is allowed by the non-extended version of the function, “m_PlayVoice()”.

Header files

The constants, function declarations, and macros necessary to compile a Meridian ACCESS application are contained in various C header files. These files must be included in the application during compilation.

Each function description in the remainder of this chapter states which header files must be included for the compilation to succeed. The following table shows header files and their functions.

Header File	**Function**
m_acc.h	General constant, structure, and return code declarations
m_local.h	Local and network functions
m_rm.h	Resource management functions
m_file.h	File access functions and macros
m_voice.h	Voice operations functions
m_msg.h	Messaging functions

Header File	Function
m_seg.h	Voice segment file functions
m_admin.h	User administration functions
m_event.h	Event handling functions
m_lh.h	Link Handler functions
m_ens.h	External message notification

Return codes

All Meridian ACCESS functions, except local functions and the event handlers, return TRUE or FALSE. They also return integer return codes to the application. The following table shows return codes common to all API functions.

Return code	Description
MMS_OKAY	Successful function completion
MME_BAD_PARAMETER	An invalid parameter was specified
MME_TIMEOUT	No response from Meridian Mail
MME_NO_LOCAL_MEMORY	Out-of-memory on the Client
MME_NOT_ACQUIRED	Meridian Mail session not acquired
MME_COMMAND_FAILED	Internal Meridian Mail command error
MME_NOT_REGISTERED	Must register first
MME_NO_MEMORY	Out-of-memory on Meridian Mail server
MME_NO_QUEUE_SPACE	Could not send message to LH—no space on queue
MME_API_QUEUE_DOWN	System error accessing API message queue.

Other return codes are described with the individual functions. Return codes begin with one of these prefixes.

Prefix	**Description**
MMS	Status
MME	Error
MMW	Warning

If an API function returns FALSE, the “rc” parameter to the function will always be set to an error code. If TRUE is returned, the return code parameter will be set to MMS_OKAY.

A status code is returned if the function is completed successfully and no error needs to be communicated to the application. The most common status code is MMS_OKAY, which is returned on the successful completion of virtually all Meridian ACCESS functions.

A warning code is returned if the function succeeded, but the application must be made aware of the circumstances. For example, if a file which was opened in “read” mode is closed with the “commit” option, the m_CloseFile() function would succeed, but would return a warning code of MMW_COMMIT_IGNORED. In many circumstances, warning codes may be ignored provided that the operation of the application will not be affected. The decision of whether or not to ignore a particular warning code should be made carefully.

An error code is returned in all cases where the function could not be performed, or failed attempting to carry out the operation. Error codes should not be ignored by the application.

It is possible that a return code not described here, or under an individual function, will be returned by some function. Such a return code should be considered as a fatal error.

Refer to Appendix A “Meridian ACCESS return codes,” for a complete list of return codes.

Chapter 4: Local functions

There are several functions that do not communicate with the Meridian Mail system. These functions perform local operations to support the Meridian ACCESS API library.

Function	Description
m_Deregister	Processed registration
m_GetVersion	Get API version code
m_Register	Process registration
m_SetTimeout	Local response timeout
m_TimeoutContinue	Respond to Timeout event
m_TimeoutOff	Ignore Timeout events
m_TimeoutOn	Accept Timeout events

m_Deregister—Process deregistration

The m_Deregister() function removes a messaging connection between a UNIX process and the Meridian ACCESS Link Handler.

Upon successful completion, a value of TRUE is returned. Otherwise a value of FALSE is returned, and status code “rc” is set to indicate the error.

Header files to include	m_acc.h (Constants, return codes) m_local.h (Function declarations)
Prerequisite	Registered
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_API_QUEUE_DOWN System error accessing API message queue MME_NO_QUEUE_SPACE No queue space to send message to LH MME_TIMEOUT Timed out waiting for response from LH MMS_LH_NOT_SYNCH LH not synchronized, command succeeded
See also	m_Register

Declaration

```
short m_Deregister(rc)
    short *rc;                                  /* returned status code */
```

m_GetVersion—API version code

This function returns a pointer to the version code of the Meridian ACCESS API library. This is a local version code—it is not returned from the Meridian Mail machine.

The version code is a string which can be up to VERSION_SIZE in length, as defined in the header files.

Header files to include	m_acc.h Constants, return codes) m_local.h (Function declarations)
See also	m_GetSysVersion
Declaration char *m_GetVersion();	

m_Register—Process registration

A Meridian ACCESS process must register with the Link Handler before any commands can be issued to the Meridian Mail machine. This registration identifies the process to the Meridian ACCESS Link Handler so that responses to commands are directed to the correct process.

The m_Register() function establishes a messaging connection between the calling UNIX process and the Meridian ACCESS Link Handler. In multiple ACCESS link configuration, the default link is 1. For an alternate link, use m_SetEnv function to specify a link before calling m_Register.

> ***Note:*** Changing the environment variable "ACCESS" in the "lh.config" file overrides the link specified with the m_SetEnv function.

Header files to include	m_acc.h Constants, return codes) m_local.h (Function declarations)
Return Codes	MME_API_QUEUE_DOWN Cannot access Link Handler's queue MME_BAD_SEM_KEY Cannot access assigned queue MME_EVENT_QUEUE_DOWN System error accessing event message queue MME_TIMEOUT Timed out waiting for response from LH MME_ALREADY_REGISTERED Calling process is already registered with LH MME_NO_QUEUE_SPACE No queue space to send message to LH MMS_LH_NOT_SYNCH LH not synchronized, command succeeded
See also	m_Deregister m_SetEnv

Declaration

```
short m_Register(rc)
    short *rc  /* returned status code */
```

m_SetTimeout—Local response timeout

All of the Meridian ACCESS functions (except events and local functions) send a command to Meridian Mail and then wait for a response. If a response is not received quickly enough, the function “times out”**.** This function sets the timeout period that is used when waiting for a response from Meridian Mail.

Header files to include	m_acc.h (Constants, return codes) m_local.h (Function declarations)
See also	m_TimeoutContinue m_TimeoutOff
Events	m_OnTimeout

Declaration

```
short m_SetTimeout(Period)
short Period;                                    /* timeout in seconds */
```

Period The period must be in the range 1 to 240 seconds, or a single constant can be used. ST_DEFAULTS is the default mode for timeouts if m_SetTimeout() is never called. In this mode, the timeout period is 40 seconds for most functions.

Period	**Description**
1 - 240	All subsequent function calls will have the specified timeout period, in seconds.
ST_DEFAULTS	Each function uses its own default timeout period.

m_TimeoutContinue—Respond to Timeout event

This function may be used to reset the Meridian Mail watchdog timer, and is typically used within a Timeout event handler. If an application does not respond to the Timeout event within one minute, the session is lost.

Header files to include	m_acc.h (Constants, return codes)
Prerequisites	Registered Acquired
Return Codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_API_QUEUE_DOWN System error accessing API message queue MME_TIMEOUT Timed out waiting for response from LH
See also	m_SetTimeout m_TimeoutOff
Events	m_OnTimeout
Declaration void m_TimeoutContinue()	

m_TimeoutOff—Ignore Timeout events

This function will prevent Timeout events from being sent to an application. With autoevent notification turned on, m_TimeoutOff is sent automatically. This function enables sending the m_TimeoutContinue function automatically to Meridian Mail.

Header files to include	m_acc.h (Constants, return codes)
See also	m_SetTimeout m_TimeoutContinue
Events	m_OnTimeout
Declaration void m_TimeoutOff()	

m_TimeoutOn—Accept Timeout events

This function re-enables Timeout events to be sent to an application.

Note: The application should issue the m_TimeoutContinue function to verify that the Meridian Mail session is still up.

Header files to include	m_acc.h (Constants, return codes)
See also	m_SetTimeout m_TimeoutOff
Events	m_OnTimeout
Declaration void m_TimeoutOn()	

Chapter 5: Resource management functions

Resource management consists of acquiring and releasing the hardware and software resources of the Meridian Mail system. Such resources include the following:

- access to Meridian Mail sessions
- access to Meridian Mail accounts
- Meridian Mail status information

Function	**Description**
m_Acquire	Acquire a Meridian Mail session.
m_AcquireOnIncomingCall	Acquire session and channel when call arrives.
m_GetSysDate	Get current Meridian Mail date.
m_GetSysVersion	Get Meridian Mail version code.
m_Logoff	Log off of a Meridian Mail account.
m_Logon	Log on to a Meridian Mail account.
m_Release	Release an acquired Meridian Mail session and channel.

m_Acquire—Acquire Meridian Mail session and channel

A UNIX process must establish a session with the Meridian Mail server before any of the Meridian Mail resources become available. The m_Acquire() function asks a session manager within Meridian Mail for both a session and a voice channel.

If no API requests are detected on the acquired voice channel for a period of one minute, the session will time out, the voice channel will be released, and a m_SessionEnd event will be sent to the application. To learn how to avoid losing a channel, see the description of the m_OnTimeout function in this guide.

In systems configured with either SMDI or AML/CSL, the PBX, by default, controls incoming calls to applications. In systems configured with SMDI, incoming calls are presented anytime after the application issues the m_Acquire function. In systems configured with AML/CSL, an incoming call is presented only after the application issues the m_AcceptCall function. The application must issue this function after every call to receive the next call. For more information about the m_AcceptCall function, see its description in this guide.

Applications which wait for incoming calls do not always need to tie up Meridian Mail resources by immediately acquiring a session and a voice channel. Such applications should use the m_AcquireOnIncomingCall() function instead of m_Acquire(). See the description of the m_AcquireOnIncomingCall() function for details about its behaviour. More information regarding the differences between using the m_Acquire() and m_AcquireOnIncomingCall() functions also can be found in the *Meridian ACCESS Developer's Guide* (NTP 555-7001-316).

Header files to include	m_acc.h (Constants, return codes) m_rm.h (Function declarations, constants)
Prerequisite	Registered
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_BAD_VERSION API library being used not supported by Meridian Mail
–continued–	

Return codes (continued)	MME_ALREADY_ACQUIRED Only one acquire per session MME_OPTION_NOT_AVAIL ACCESS option not set MME_INVALID_CLASS Invalid service class MME_NO_TASK No Meridian ACCESS Toolkit available MME_MAX_REQUESTS Maximum number of acquire requests reached
See also	m_Release m_AcquireOnIncomingCall m_AcceptCall
Events	m_OnSessionEnd m_OnTimeout

Declaration

```
short m_Acquire(Class, rc)
short Class;       /* MM application class type */
short *rc;         /* returned error code */
```

Class Is used to identify the Meridian ACCESS application to the system. Either a specific class number in the range 0-8999 (agreed upon with the Meridian Mail administrator) or a single constant can be used. See the Meridian ACCESS *Developer’s Guide* (NTP 555-7001-316) for a description of ACCESS classes.

Class	Description
0 - 8999	Acquire a dedicated voice channel.
AC_SHARED	Acquire any non-dedicated voice channel.

m_AcquireOnIncomingCall—Acquire Meridian Mail session and channel when call arrives

This function tells a session manager within Meridian Mail that a session and voice channel should be acquired only when an incoming call arrives.

m_AcquireOnIncomingCall (AOIC) allows voice channels to be used for services in addition to Meridian ACCESS, since channels may be configured with an “ALL” service type in the CAT. AOIC should be used for Meridian ACCESS applications that wait for incoming calls (for example, IVR applications). These applications do not always need to perform Meridian Mail operations, which require a session and a voice channel, until after a call has arrived.

Before an incoming call arrives, the application does not yet have control of an active Meridian Mail session and, thus, cannot use any of the session-dependent Meridian ACCESS API functions. Incoming calls will be presented to the application via the OnIncomingCall event. When this event arrives, a Meridian Mail session and voice channel will already have been acquired automatically for the application. At this point, the application freely can perform other Meridian ACCESS API functions.

If an incoming call does not arrive within three minutes of calling this function, then the system automatically will cancel the AcquireOnIncomingCall request, and an OnSessionEnd event will be sent to the application. For this reason, applications should ensure that the AcquireOnIncomingCall request is reissued *at least* once every three minutes. To facilitate this process, a high-level API function called m_WaitingForCall() has been provided.

Note that if an application has a voice channel acquired when the m_AcquireOnIncomingCall function is called, the system automatically will release the voice channel. Releasing the channel prevents an application from “missing” calls that arrive between a call to m_Release(), and subsequent m_AcquireOnIncomingCall().

To explicitly cancel the m_AcquireOnIncomingCall() function, use the m_Release() function.

See the *Meridian ACCESS Developer’s Guide* (NTP 555-7001-316) for further information regarding the differences between using the m_AcquireOnIncomingCall() and m_Acquire() functions.

Header files to include	m_acc.h (Constants, return codes) m_rm.h (Function declarations)
Prerequisite	Registered
Return codes	MME_BAD_VERSION API library being used not supported by Meridian Mail MME_ALREADY_ACQUIRED Only one acquire per session MME_OPTION_NOT_AVAIL ACCESS option not set MME_INVALID_CLASS Invalid service class MME_MAX_REQUESTS Max. number of outstanding acquires reached MME_NOT_REGISTERED Calling process is not registered with the LH MME_CHAN_IN_USE Voice channel is already in use MME_CHANNEL_READY m_AcceptCall (already) issued
See also	m_Acquire m_Release m_WaitingForCall
Events	m_OnSessionEnd m_OnIncomingCall

Declaration

```
short m_AcquireOnIncomingCall(Class, rc)
short Class;     /* MM application class type */
short *rc;       /* returned error code */
```

Class Is used to identify the Meridian ACCESS application to the system. A specific class number in the range 0-8999 (agreed upon with the

Meridian Mail administrator) must be used to ensure that an incoming call will get routed to the correct application.

Class	**Description**
0 - 8999	Acquire a session and channel when an incoming call arrives for an application with the given class.

m_ChgCustomerNo—Change customer identification number

The current Meridian Mail architecture allows for the possibility of having multiple customers on one physical Meridian Mail machine. (This is an optional feature on most systems.) Every customer in this architecture has their own "virtual" machine, (that is, they have their own unique logon accounts, passwords, data, and so on). Every ACCESS session will have a "default Customer ID number" that is set when the session is invoked. This function allows an application to change the current customer number specified for their ACCESS session on Meridian Mail. The application will then have access to the accounts and data associated with the new customer identification number. This command can only be issued when the application does not have a logon session currently in effect (that is, a user must not be logged into a Meridian Mail account) given the fact that the Meridian Mail model does not permit different customers to share data.

Header files to include	m_acc.h (Constants, return codes) m_rm.h (Function declarations)
Prerequisites	Registered
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_ALREADY_LOGON Cannot set customer ID while logged on MME_BAD_ID Not a valid customer number

Declaration

```
short m_ChgCustomerNo(CustomerNo, rc)
    short CustomerNo;    /*Customer Identification Number  */
    short *rc            /*status return code   */
```

CustomerNo This parameter is the identification number of the customer to be accessed. The default customer is requested by specifying "-1"; the system administrator can configure the default for ACCESS applications through the MMI when the multi-customer feature is present.

m_GetChanInfo—Retrieve voice channel information

This function allows an application to retrieve specific channel information for the voice channel currently in use. The information is returned in a structure which includes channel TN, channel DN, and ACD agent position ID. This function will primarily be used in servicing “coordinated screen transfer–type” applications. It also provides a method for applications to monitor which voice port they are controlling.

Header files to include	m_acc.h (Constants, return codes)
Prerequisites	Registered Acquired Connected
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_NO_ACTV_CHNL No active voice channel

Declaration

```
short m_GetChanInfo(ChanRec,rc)
    struct ChanInfo *ChanRec;            /*Returned Channel Information
                                          Structure */
short*rc;   /*status return code*/
```

ChanRec This parameter is a pointer to a structure that contains the following information:

```
struct ChanInfo{
    char TN[TN_SIZE]
    char DN[DN_SIZE]
    char AgentPos[AGENT_SIZE]
    }
```

The following ChanInfo fields are defined when your Meridian Mail system is installed, and most can be viewed in the channel allocation table administration screen.

TN This is a 4–byte field which can be used to identify the actual hardware location of the voice port.

- TN [0] is always one.
- TN [1] is the node on which the voice port resides.
- TN [2] is the voice card on which the voice port resides; on an MSM system, this field contains the T1 card pair number.
- TN [3] is the logical port on the voice card; on an MSM system, this field contains the T1 channel number.

DN This is the channel DN for the voice port.

Agent Pos This is the Agent Position ID for the voice port. It has the format of a DN. It is a number identifying an ACD agent position within an ACD group. The position ID is an actual DN (although not directly callable). It is unique across the entire Meridian 1 customer and is associated with a particular (agent) telephone set. This field is not applicable to other switches.

m_GetSysDate—Get current Meridian Mail date

This function returns the date and time as set on the Meridian Mail machine. The returned date and time are those of the general system clock of the Meridian Mail machine. The returned date is used within Meridian Mail as the time stamp on all files (including voice messages), and as the reference point for delayed message delivery. In systems configured with AML/CSL, the time actually originates from the PBX.

Header files to include	m_acc.h (Constants, return codes) m_rm.h (Function declarations)
Prerequisites	Registered Acquired
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire”

Declaration

```
short m_GetSysDate(Date, rc)
struct DATE *Date;                    /* returned date and time */
short *rc;  /* returned status code */
```

Date Is returned in the standard API format (see the Date and Time section in the Meridian Mail Facilities chapter 2).

m_GetSysVersion—Get Meridian Mail version code

This function returns the version code of the session software currently being executed on the Meridian Mail machine. The format of the returned version code is “MMn” where “n” is the Meridian Mail release number.

Header files to include	m_acc.h (Constants, return codes) m_rm.h (Function declarations)
Prerequisites	Registered Acquired
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire”
See also	m_GetVersion

Declaration

```
short m_GetSysVersion(VersionCode, rc)
char *VersionCode;                      /* returned version string */
short *rc;  /* returned status code */
```

VersionCode This should point to an area large enough to accept a string up to VERSION_SIZE in length.

m_Logoff—Log off of Meridian Mail account

The m_Logoff() function logs off of a Meridian Mail account without releasing the Meridian Mail session. This allows an application to log on to a different account while guaranteeing that a session and voice channel are still available. If a voice connection has been established, it will not be dropped. All open files will be closed and all changes will be saved.

Header files to include	m_acc.h (Constants, return codes) m_rm.h (Function declarations)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_DO_LOGON Must be logged on to use this command
See also	m_Logon m_Release
Declaration short m_Logoff(rc) short *rc; /* returned status code */	

m_Logon—Log on to Meridian Mail account

This command provides the ability to log on to a Meridian Mail account. Logging on provides access to the files in the given user's cabinet.

A user may log on to a given account from more than one session at a time (from two different client workstations, for example). Such secondary logons will succeed (m_Logon() returns TRUE), and the Info parameter will be set accordingly.

Attempting to log on twice from the same session will cause the logoff of the current user and then the logon of the new one. If the function fails, it is possible that the implicit logoff succeeded but that the new logon failed. You would no longer be logged on to any Meridian Mail account.

Header files to include	m_acc.h (Constants, return codes) m_rm.h (Function declarations)
Prerequisites	Registered Acquired
Status Codes	MMS_OKAY Logon okay MMW_DUP_LOGON Specified UserID logged on elsewhere MME_PSWD_OLD User's password has expired MME_DUP_OLD Specified UserID logged on elsewhere and user's password has expired
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_BAD_ID Invalid User ID MME_BAD_PSWD Incorrect Password MME_ACCESS_DENIED Account is locked out MME_MAX_LOGONS Too many failed logon attempts

–continued–

See also	m_Logoff m_Release

Declaration

```
short m_Logon(UserID, Password, Info, rc)
char *UserID;                          /* userid of account */
char *Password;                        /* password to logon with */
short *Info;                           /* returned info if logon
                                        successful*/
short *rc;  /* returned status code */
```

Info Additional information given upon successful logon. May indicate that the UserID is already logged on and/or that the password has expired.

UserID The maximum length of this field is given in the format USERID_SIZE as defined in m_acc.h.

Password The maximum length of this field is given in the format PSWD_SIZE as defined in m_acc.h.

m_Release—Release Meridian Mail session and channel

The m_Release() function allows an active Meridian Mail session to be released or an outstanding m_AcquireOnIncomingCall() request to be canceled.

Releasing an active session automatically logs off any user and frees the voice channel which was associated with the session. All open files will be closed and all changes will be saved.

Header files to include	m_acc.h (Constants, return codes) m_rm.h (Function declarations)
Prerequisites	Registered Acquired
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MMW_ALREADY_RELEASED Warning: session already released by system
See also	m_Acquire m_AcquireOnIncomingCall

Declaration

```
short m_Release(rc)
short *rc;  /* returned error code */
```

Chapter 6: Telephony functions

Telephony functions are used for manipulating connections on the telephone set. They allow calls to be placed, answered, transferred, conferenced, and disconnected.

Functions	**Description**
m_AcceptCall	Signal readiness to accept calls.
m_AddOnCall	Put call on hold and place new call.
m_AnswerCall	Answer incoming call.
m_ConferenceCall	Create a conference call.
m_DisconnectCall	Disconnect telephone connection.
m_GenerateDTMF	Generate DTMF tones.
m_GetCallInfo	Get detailed information on calls.
m_MakeCall	Connect to a telephone.
m_ReconnectCall	Reconnect to call currently on hold.
m_SoundDetect	Start detecting presence of sound/ silence on voice channel.
m_StopSoundDetect	Stop detecting presence of sound/ silence on voice channel.
m_TransferCall	Transfer call to given telephone number.
m_TransferCallRevert	Transfer call to revert DN.

With several of the telephony functions, the application has a choice regarding how the operation should be performed. These functions normally return Call Progress events indicating the current state of the call or information regarding the success or failure of the operation. However, an application may not be interested in handling these Call Progress events if, after initiating the operation, it will simply wait for the operation to complete.

To facilitate this process, these functions are provided with a TelephonyReturn parameter. This parameter allows the application to choose when the telephony function should return control back to the application.

The possible values for the TelephonyReturn parameter are the following:

Telephony Return	**Description**
TR_IMMEDIATE	Initiate operation and return immediately; application must check CALL Progress events. ***Note:*** The TR_IMMEDIATE option should be used with extreme caution, or calls may be lost.
TR_ON_COMPLETE	Wait for operation to complete before returning; Call Progress events will not be sent to the application. ***Note:*** When TR_ON_COMPLETE is specified, the application will not receive CallProgress events, since the CallProgress event handler will temporarily be replaced.

Meridian Mail can be connected to different types of switches. In SMDI configurations, some of the ACCESS telephony operations may work differently. Please refer to the telephony chapter in the Meridian ACCESS *Developer’s Guide* (NTP 555–7001–316) for more details.

m_AcceptCall—Signal readiness to accept calls

This function informs the PBX that the calling application is ready to accept calls. In systems configured with AML/CSL, no incoming calls are presented to an application until it issues this command. In systems configured with SMDI, calls cannot be blocked by MM and are presented when they arrive. Calling this function in SMDI configured systems has no effect (that is, it is ignored but will not fail). This function allows applications to complete any necessary processing between calls or during startup. The m_AcceptCall function only applies to applications using a dedicated voice channel (m_Acquire).

An application may hold a channel and block calls for up to one hour after calling m_Acquire(). However, incoming calls will automatically be presented to an application even if it does not call m_AcceptCall() within one hour of acquiring a voice channel. Note that the one-hour timer is reset each time an inbound or outbound call is made to or from the application.

Header files to include	m_acc.h (Constants, return codes)
Prerequisites	Registered Acquired
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_TIMEOUT Timed out waiting for response from LH MME_CHANNEL_READY m_AcceptCall (already) issued
See also	m_AcquireOnIncomingCall m_Acquire
Declaration	

```
short m_AcceptCall (rc)
short *rc      /* returned status code */
```

m_AddOnCall—Put current call on hold; place a new call

This routine puts the current call on hold and places a new call from the voice channel to a specified telephone number (any telephone number, subject to Meridian Mail system-wide restrictions). To take the original call off hold, use the m_ReconnectCall() or m_ConferenceCall() function.

This function returns TRUE if successful. Otherwise, it returns FALSE and the status code may be checked to determine the nature of the problem.

When attempting m_AddOnCall() within the first few seconds after an answer is detected, call failure may occur on non-AML trunks. If this happens, append the “#” character to the dialed DN. This will force the switch to permit the command as soon as the DSP has detected voice.

Note that if the new call is disconnected by the remote end while the original call is on hold (that is, prior to m_ReconnectCall(), or m_ConferenceCall()), a CallProgress-Disconnect event will not be received. As a result, the application must issue an m_ReconnectCall request to reconnect to the original call.

If the add-on operation results in busy or reorder, the call is automatically reconnected.

Header files to include	m_acc.h (Constants, return codes) m_voice.h (Function declarations)
Prerequisites	Registered Acquired Connected
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_NO_ACTV_CHNL No active voice channel MME_BAD_DN DN is invalid
–continued–	

Return codes (continued)	MME_BUSY_DN Phone was busy MME_NOT_ANSWERED Call not answered within given time period MME_CALL_REORDER Call has been rejected MME_CALL_FAILURE Call connection attempt has failed MME_CALL_COLLISION Call resulted in collision MME_RESTRICTED_DN DN has a restricted prefix. MME_NO_CHNL No voice channel was available
See also	m_MakeCall m_ReconnectCall m_ConferenceCall m_TransferCall m_TransferCallRevert
Events	m_OnCallProgress

Declaration

```
short m_AddOnCall(DN, TelephonyReturn, MaxTime, rc)
    char *DN;                    /* new telephone number to call */
    short TelephonyReturn;       /* when to return from operation */
    unsigned short MaxTime; /* max. time for completion
                                        (sec.)*/
    short *rc;                   /* returned status code */
```

DN The DN should point to a null-terminated string with a maximum length DN_SIZE as defined in m_acc.h.

TelephonyReturn If TR_IMMEDIATE, the application must check the incoming m_OnCallProgress() events to determine the call state. When TR_ON_COMPLETE is specified, the application will not receive Call Progress events until the function is complete since the CallProgress event handler will be temporarily replaced.

MaxTime This specifies the maximum time the function should wait for the operation to complete. Set the value to MIN_RING_TIME or greater to ensure that the called telephone has enough time to ring. If TR_IMMEDIATE has been specified, the MaxTime parameter is ignored.

m_AnswerCall—Answer incoming call

This function is used to answer an incoming call on the voice channel after an m_OnIncomingCall() event has been received. If not answered within 15 seconds, the call will be transferred to the application's revert DN, and the Meridian Mail session will be dropped (with an m_OnSessionEnd() event).

Header files to include	m_acc.h (Constants, return codes) m_voice.h (Function declarations)
Prerequisites	Registered Acquired
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_NO_INC_CALL No incoming call to answer
See also	m_WaitingForCall m_DisconnectCall
Events	m_OnIncomingCall m_OnSessionEnd

Declaration

```
short m_AnswerCall(rc)
short *rc;  /* returned status code */
```

m_ConferenceCall—Create a conference call

After the m_AddOnCall() function has been used to put the current call on hold and place a new call, the application may make the original call part of a conference by performing m_ConferenceCall(). The original call is taken off hold and is placed into a conference with all of the other active calls.

This function returns TRUE if successful. Otherwise, it returns FALSE and the status code may be checked to determine the nature of the problem.

Header files to include	m_acc.h (Constants, return codes) m_voice.h (Function declarations)
Prerequisites	Registered Acquired Connected Call added on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_OPER_TIMEOUT Operation not completed in given time period MME_DO_ADDONCALL Must do m_AddOnCall() first
See also	m_AddOnCall m_ReconnectCall
Events	m_OnCallProgress

Declaration

```
short m_ConferenceCall(TelephonyReturn, MaxTime, rc)
short TelephonyReturn;          /* when to return from operation */
unsigned short MaxTime;         /* max. time for completion
                                   (secs) */
short *rc;  /* returned status code */
```

TelephonyReturn If TR_IMMEDIATE, the application must check for m_OnCallProgress() events to determine the conference request state. When TR_ON_COMPLETE is specified, the application will not receive Call Progress events until the function is complete since the CallProgress event handler will be temporarily replaced.

MaxTime This specifies the maximum time the function should wait for the operation to complete. Set the value to MIN_OPER_TIME or greater so that the operation has enough time to complete under normal circumstances. If TR_IMMEDIATE has been specified, MaxTime is ignored.

m_DisconnectCall—Disconnect telephone connection (hang up)

A voice channel connection may be disconnected using this function. This is equivalent to hanging up the telephone. If the voice channel is involved in a conference, the other calls in the conference are not affected.

Header files to include	m_acc.h (Constants, return codes) m_voice.h (Function declarations)
Prerequisites	Registered Acquired Connected
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_NO_ACTV_CHNL No active voice channel
See also	m_AnswerCall m_MakeCall m_Release
Events	m_OnCallProgress
Declaration	

```
short m_DisconnectCall(rc)
in *rc        /* returned status code*/
```

The application will always receive a disconnect event in response to this command indicating the switch has confirmed the disconnect and is ready for the next call, either inbound or outbound.

m_GenerateDTMF—Generate DTMF tones

This function allows applications to generate DTMF tones. The tones will not trigger digit events to the calling application.

A call must be established prior to execution of this function, although there must not be any voice operations (play or record, for example) in progress.

Although the call will return before all tones and pauses have been generated, the application will not receive any call progress events. Allow approximately 1/10 second for each tone to be generated.

Header files to include	m_acc.h (Constants, return codes)
Prerequisites	Registered Acquired
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_TIMEOUT Timed out waiting for response from LH MME_BAD_SEQUENCE Invalid command sequence MME_INVALID_DTMF Invalid DTMF string MME_DETECT_INPROG Sound detection module in progress

Declaration

```
short m_GenerateDTMF (Digits, rc)
char *Digits;   /* string of DTMF tones to be generated */
short *rc;      /* returned status code */
```

Digits Valid digits are 0-9, "#", "*", and the military tones A-D. The "," character may be used to generate a two-second pause. Pointer to a null terminated string of maximum length DN_SIZE as defined in the m_acc.h.

m_GetCallInfo—Get detailed information on calls

This function is intended for use by applications requiring detailed information on either incoming or outgoing calls. For example, an application that receives a CP_DNUpdate state change from the m_OnCallProgress event may use this function to retrieve the new DN.

Not all fields are valid at all times. The information available here in SMDI configured systems is quite limited.

Header files to include	m_acc.h (Constants, return codes) m_voice.h (Function declarations)
Prerequisites	Registered Acquired Call in progress
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire"
Events	m_OnCallProgress

Declaration

```
short m_GetCallInfo(CallRec, rc)
struct CallInfo *CallRec                /* returned call information */
short *rc   /* returned status code */
```

where:

```
struct CallInfo {
short CallState;                        /* last CallProgress State
                                           Change*/
short CallInfo;                         /* last CallInfo Info Change */
char  CallingDN[DN_SIZE];               /* DN of caller */
char  CallingDNType;                    /* DNType of caller */
char  CalledDN[DN_SIZE];                /* DN dialed by caller */
char  CalledDNType;                     /* DNType of Called DN */
char  CalledTN[TN_SIZE];                /* TN */
char  OtherDN[DN_SIZE];                 /* The DNIS */
char  CallType;                         /* Type of call */
char  DeviceType;                       /* device type */
char  CallID[ID_SIZE];                  /* CallID for CCR call */
}; Note that Calling DN Type, Called DN Type, Call Type, and Device Type are not
printable ASCII characters, but bytes containing binary information.
```

CallState See m_OnCallProgress() for details.

CallInfo See m_OnCallProgress() for details.

DNtype	**Description**
DN_UNKNOWN	Unknown
DN_INTERNATIONAL	International
DN_NATIONAL	National
DN_SPECIAL	Special
DN_SUBSCRIBER	Subscriber
DN_ESN	ESN call
DN_CDP	CDP call
DN_RESERVED	Reserved
DN_INTERNAL	Internal extension
DN_RAC_MEMBER	Route access code and member number
DN_RAC	Route access code only
DN_ATTNDT_MEMBER	Attendant code and member number
DN_ACD_POS	ACD DN and position ID
DN_ACD_DNIS	ACD position and DNIS
DN_ACD_IANI	IANI ACD DN and position ID
DN_IANI	Inband ANI
DN_ACD	ACD DN

CalledTN Is 4 bytes of numeric data which are the actual Meridian 1 physical address of the set. This field is only set for internal calls that have been answered. m_GetCallInfo returns the TN information in the Meridian 1 internal format. This format can be converted to the TN of the agent's set using the following algorithm:

TN (packed format) = {TN[0] * 256^3 + TN[1] * 256^2 + TN[2] * 256
+ TN[3]}

Note: TN[0] and TN[1] can be ignored since presently they will always be 0 (zero).

CallType	**Description**
CT_DIRECT	Direct
CT_FORWARDED	Call forwarded
CT_FWD_BUSY	Call forwarded on busy
CT_FWD_NOANSWER	Call forwarded, no answer
CT_FWD_DND	Call forwarded on do not disturb

DeviceType	**Description**
DT_EXTERNAL	Internal
DT_INTERNAL	External

m_MakeCall—Connect to telephone

This function places a call from the voice channel to a given telephone number.

This function returns TRUE if successful. Otherwise, it returns FALSE and the status code may be checked to determine the nature of the problem.

When placing outbound calls within the first few seconds after an answer is detected, call failure may occur on non-AML trunks. If this happens, append the "#" character to the dialed DN. This will force the switch to permit the command as soon as the DSP has detected voice.

On systems configured with AML/CSL, internal calls on the switch do not use or need the DSP to detect answer, busy, and so on. It only uses the DSP to detect these when calls go out on trunks regardless of the trunk type. On systems configured with SMDI, the DSP is used on all calls to monitor the call progress.

Header files to include	m_acc.h (Constants, return codes) m_voice.h (Function declarations)
Prerequisites	Registered Acquired
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_CHAN_IN_USE Voice channel is already in use MME_BAD_DN DN is invalid MME_BUSY_DN Phone was busy MME_NOT_ANSWERED Call not answered within given time period MME_CALL_REORDER Call has been rejected
–continued–	

Return codes (continued)	MME_CALL_FAILURE Call connection attempt has failed MME_CALL_COLLISION Call resulted in collision MME_RESTRICTED_DN DN has a restricted prefix MME_NO_CHNL No voice channel was available
See also	m_AddOnCall m_DisconnectCall m_Release
Events	m_OnCallProgress

Declaration

```
short m_MakeCall(DN, TelephonyReturn, MaxTime, rc)
    char *DN;                       /* telephone number to call */
    short TelephonyReturn;          /* when to return from operation */
    unsigned short MaxTime;           /* max. time for completion
                                        (secs)*/
    short *rc;                      /* returned status code */
```

DN This should point to a null-terminated string. Maximum length DN_SIZE as defined in m_acc.h.

Telephony Return If TR_IMMEDIATE, the application must check incoming m_OnCallProgress() events to determine the state of the call. When TR_ON_COMPLETE is specified, the application will not receive Call Progress events until the function is complete since the CallProgress event handler will be temporarily replaced.

MaxTime This specifies the maximum time the function should wait for the operation to complete. Set the value to MIN_RING_TIME or greater to ensure that the called telephone has enough time to ring. If TR_IMMEDIATE has been specified, the MaxTime parameter is ignored.

m_ReconnectCall—Reconnect to call currently on hold

After m_AddOnCall() has been used to put the current call on hold and place a new call, the application may return to the original call with m_ReconnectCall(). This cancels the m_AddOnCall() request.

This function returns TRUE if successful. Otherwise, it returns FALSE and the status code may be checked to determine the nature of the problem.

Header files to include	m_acc.h (Constants, return codes) m_voice.h (Function declarations)
Prerequisites	Registered Acquired Connected Call added on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_OPER_TIMEOUT Operation not completed in given time period MME_DO_ADDONCALL Must do m_AddOnCall() first
See also	m_AddOnCall m_ConferenceCall m_DisconnectCall
Events	m_OnCallProgress

Declaration

```
short m_ReconnectCall(TelephonyReturn, MaxTime, rc)
    short TelephonyReturn; /* when to return from operation */
    unsigned short MaxTime; /* max. time for completion (secs)*/
    short *rc;              /* returned status code */
```

TelephonyReturn If TR_IMMEDIATE, the application must check for m_OnCallProgress() events to determine the reconnection request state. When TR_ON_COMPLETE is specified, the application will not receive Call Progress events until the function is complete, since the CallProgress event handler will be temporarily replaced.

MaxTime This specifies the maximum time the function should wait for the operation to complete. Set the value to MIN_OPER_TIME or greater so that the operation has enough time to complete under normal circumstances. If TR_IMMEDIATE has been specified, MaxTime is ignored.

m_SoundDetect—Detect the presence of sound or silence

This function allows an application to start detecting the presence of sound or silence on a voice channel with an active call. The application specifies whether sound or silence is to be monitored and the time window of interest. Additional characteristics can be specified to a limited extent in the form of minimum/maximum acceptable duration of sound or silence, and in the case of sound for the purpose of monitoring voice, the duration of inter-word silence.

The function m_SoundDetect sets up the environment for monitoring. The actual results come in the form of a “Sound Detect” event which can be captured by installing the appropriate event handler (that is, using OnSoundDetect event handler install function). Whenever an application calls m_SoundDetect, it must have previously installed a “SoundDetect” event handler. The application can expect to receive at most OnSoundDetect event for each call to m_SoundDetect. The results of the monitoring can be expected either on or before the specified time window. Two successive m_SoundDetect calls cannot be made without receiving the results of the first call. Monitoring of sound or silence on a voice channel can be stopped before the time window has expired by calling the m_StopSoundDetect function.

Header files to include	m_acc.h (Constants, return codes) m_voice.h (Function Declarations)
Prerequisites	Registered Acquired Connected OnSoundDetect
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_NO_ACTV_CHNL No active voice channel MME_OTHER_TELEPHONY Other telephony request still in progress

–continued–

Return codes (continued)	MME_BAD_DETECTION Context must be either SOUND or SILENCE MME_BAD_DURATION Duration value out of range MME_INSTL_EVENT Must install SoundDetect event handler MME_DETECT_INPROG Sound detection module in progress MME_BAD_SEQUENCE Invalid command sequence
Events	m_OnSoundDetect

Declaration

```
short m_SoundDetect (Context, Period, MinDur, MaxDur, IWSDur,rc)
AudioSignal Context;              /*audio signal to be detected SOUND or
                                   SILENCE */
long Period; /*duration of monitoring period-milliseconds */
long MinDur; /*minimum continuous audio
                  signal-milliseconds*/
long MaxDur; /*maximum continuous audio
           signal-milliseconds*/
long IWSDur; /*inter word silence duration-milliseconds*/
short*rc;    /*status return code */
```

Context This parameter specifies the type of audio signal to be detected. Either sound or silence can be detected using the definitions SOUND/SILENCE which are included in “m_voice.h”.

Period This parameter specifies the time window (in milliseconds) that the sound detection module remains in operation. If a “Period” of 10000 milliseconds is specified, the sound detection module will monitor a voice channel for an absolute maximum of 10 seconds. The sound detection module will stop monitoring the voice channel before the time window expires if it discovers a line audio signal satisfying the other monitoring parameters first.

MinDur This parameter is used to specify the minimum duration (in milliseconds) of continuous audio signal (either sound or silence) that will satisfy the application. Its purpose is to ignore audio signals of the correct context but of insubstantial duration. Should an audio signal be discovered on the voice channel with a duration smaller than the minimum duration, it will be ignored and monitoring will continue. If the minimum duration of the desired audio signal is never satisfied within the time window, the application receives a Sound Detect event with a duration of 0.

MaxDur This parameter is used to specify the maximum duration (in milliseconds) of continuous audio signal (either sound or silence) that will satisfy the application. Its purpose is to cause the sound detect module to terminate if an audio signal matching the context remains on the voice channel for a period of time matching or exceeding MaxDur. Thus, if an audio signal is detected whose duration exceeds MaxDur the sound detection module immediately halts monitoring and the application receives a Sound Detect event with a duration of MaxDur. If an audio signal is detected whose duration is between MinDur and MaxDur the sound detection module immediately halts monitoring and the application receives a Sound Detect event with the exact duration detected.

IWSDur This parameter is used only in the context of SOUND detection for the purpose of monitoring actual voice. It is used to specify the maximum duration (in milliseconds) of inter-word silence. It is ignored in the context of silence detection. Its purpose is to bridge the gap between spoken words; that is, if a silence period is detected between two spoken words that is less than the “inter-word silence” duration, then that period of silence is considered as sound. If the sound detection module happens to be monitoring inter-word silence when the time window expires, that silence will not be considered as part of the preceding sound duration.

The values of the above parameters should be such that

0<= MinDur <= MaxDur <= Period <= 300,000 milliseconds (5 min.)

0<= IWSDur <= Period <= 300,000 milliseconds (5 min.)

Examples

In order to demonstrate the use of the sound detect module, we present two examples. In the first example, we try to determine whether a human or an answering machine has answered our call. In this case, we are looking for the long silence that the human speaker produces when he or she pauses to listen for a response after having said “Hello?” It is assumed that machines do not pause in their speech within the first five seconds of answering the call. In the second example, we are trying to detect positive voice confirmation within a period of five seconds. The reader should note the installation of the VoiceDetect event handler (m_OnSoundDetect) before actually calling m_Sound Detect. Remember, the results of a voice monitoring session are returned in the form of a VoiceDetect event. Also, note that the parameters used in these examples are purely speculative. They DO NOT represent “magic numbers” that can be applied to any application. Actual parameters will vary from application to application.

```
/*Install VoiceDetect Event Handler */
m_OnSoundDetect(SoundDetectHandler());

/*Clear Global Sound/Silence Variables*/
DetectedVoice=DetectedSilence=FALSE;

/*

*Example1:Determine Whether we are talking to a

* Human or Machine after establishing a call

*/

 /*Check for 5 seconds*/
m_SoundDetect (SILENCE, 5000, 1000, 2000, 0, rc);

pause();                                    /*wait for the SoundDetect Event */

if (DetectedSilence){

            /*We are talking to a Human */

else{

            /*We are talking to an Answering Machine */

}

            /*Clear Global Sound/Silence Variables */

DetectedVoice=DetectedSilence=FALSE;

/*

*Example2: Detecting possibility of voice

*           after establishing a call

*/

/*Check for 5 seconds */
m_ SoundDetect (SOUND, 5000, 250, 3000, 1000, rc);

pause();                                    /*wait for the SoundDetect Event */

if (DetectedVoice){

            /*We have detected presence of voice */

else{

            /*We have not detected the presence of voice */

}


/*This is our VoiceDetect Event Handler*/
```

```
SoundDetectHandler (Context, Duration)
 /*Used by the application to verify requested context*/
AudioSignal Context;
long Duration;                              /*This is the duration of the audio
signal detected */
{
if (Context ==SOUND) && (Duration >0)
        DetectedVoice = TRUE;
if (Context ==SILENCE) && (Duration >0)
        DetectedSilence = TRUE;
}
```

m_StopSoundDetect — Stop detecting the presence of sound or silence

This function allows an application to abort a previous request for sound or silence detection before the specified time window expires. The application will not receive a SoundDetect event.

Header files to include	m_acc.h (Constants, return codes) m_voice.h (Functions, declarations)
Prerequisites	Registered Acquired Connected SoundDetect
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_NO_PREV_DETECT No previous request for sound/silence detection
See also	m_SoundDetect
Events	m_OnSoundDetect

Declaration

```
short m_StopSoundDetect (rc)
    short *rc;                              /*status return code       */
```

m_TransferCall—Transfer call to given telephone number

When a Meridian ACCESS application is running, a call may be connected between a telephone set and a voice channel. This function allows that call to be transferred to another set so that the voice channel is no longer being used.

If the command is successful, this function returns TRUE. Otherwise, the function returns FALSE with a return status code.

When m_TransferCall() succeeds, the voice connection to Meridian Mail no longer exists. The application must reestablish the connection with m_MakeCall() or wait for another incoming call before subsequent voice operations may be performed.

When attempting m_Transfer() within the first few seconds after an answer is detected, call failure may occur on non-AML trunks. If this happens, append the "#" character to the dialed DN. This will force the switch to permit the command as soon as the DSP has detected voice.

Header files to include	m_acc.h (Constants, return codes) m_voice.h (Function declarations)
Prerequisites	Registered Acquired Connected
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_NO_ACTV_CHNL No active voice channel MME_BAD_DN DN is invalid
See also	m_MakeCall m_TransferCallRevert m_OnCallProgress m_AddOnCall

–continued–

Events	m_OnCallProgress

Declaration

```
short m_TransferCall(ToDN, TelephonyReturn, MaxTime, rc)
    char *ToDN;          /* DN to transfer to */
    short TelephonyReturn; /* when to return from operation */
    unsigned short MaxTime; /* max. time for completion (secs)*/
    short *rc;           /* returned status code */
```

ToDN This should point to a null-terminated string. To transfer to the operator, use a ToDN of 0. Maximum length DN_SIZE as defined in m_acc.h.

TelephonyReturn If TR_IMMEDIATE, the application must check the incoming m_OnCallProgress() events to determine the state of the call. When TR_ON_COMPLETE is specified, the application will not receive Call Progress events until the function is complete since the CallProgress event handler will be temporarily replaced.

MaxTime This specifies the maximum time the function should wait for the operation to complete. Set the value to MIN_RING_TIME or greater to ensure that the called telephone has enough time to ring. If TR_IMMEDIATE has been specified, the MaxTime parameter is ignored.

m_TransferCallRevert—Transfer call to revert DN

This function transfers a connected call to the custom revert DN which is configured for each user's DN by the Meridian Mail administrator.

When m_TransferCallRevert() succeeds, the voice connection to Meridian Mail no longer exists. The application must reestablish the connection with m_MakeCall(), or wait for another incoming call before subsequent voice operations may be performed.

Header files to include	m_acc.h (Constants, return codes) m_voice.h (Function declarations)
Prerequisites	Registered Acquired Logged on Connected
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_NO_ACTV_CHNL No active voice channel MME_BAD_DN DN is invalid MME_DO_LOGON Must be logged on to use this command
See also	m_MakeCall m_TransferCall m_AddOnCall
Events	m_OnCallProgress

Declaration

```
short m_TransferCallRevert(TelephonyReturn, MaxTime, rc)
short TelephonyReturn; /* when to return from operation */
unsigned short MaxTime;  /* max. time for completion (secs) */
short *rc;             /* returned status code */
```

TelephonyReturn If TR_IMMEDIATE, the application must check the incoming m_OnCallProgress() events to determine the state of the call.

When TR_ON_COMPLETE is specified, the application will not receive Call Progress events until the function is complete since the CallProgress event handler will be temporarily replaced.

MaxTime This specifies maximum time function should wait for an operation to complete. Set value to MIN_RING_TIME or greater so that the called telephone has enough time to ring. If TR_IMMEDIATE has been specified, MaxTime is ignored.

Chapter 7: File access functions

The Meridian Mail system maintains a typical directory structure of files, except that the contents of the files may contain both voice and text. A disk directory under Meridian Mail is called a “cabinet”.

File access routines provide cabinet operations (finding the names and dates of files), and file-level operations (opening, closing, and deleting entire files).

Cabinet-level access functions	**Description**
m_GetCabinetInfo	Cabinet information summary
m_FilePattern	Retrieve file directory information
m_RetrieveFile	Retrieve file directory information

File-level access functions	**Description**
m_CloseFile	Close a file.
m_CommitFile	Commit file to disk.
m_CopyFile	Copy a file.
m_CreateFile	Create a new file.
m_DeleteFile	Delete a file.
m_FileExistCheck	Check for existence of a file.
m_GetFileInfo	Get information on a file.
m_OpenFile	Open a file.
m_RenameFile	Rename a file.
m_SetFileSubject	Set subject field of a file.
m_UndeleteFile	Undelete a file.

File-level operations can be performed either by name, as on most computers, or by number. In Meridian Mail, file names are not necessarily unique within a cabinet. The file number is unique but its association with a file is temporary as explained below.

Because file names are not unique, functions which create new files or rename existing files may result in duplicate file names. If several files in the cabinet have the same name, functions which take a file name as a parameter will only be performed on the first of those files in the cabinet.

File numbers are intended for use in applications which place a list of files in a menu or which deal with non-unique file names. The position in the list may be used to open the file, for example. Both a file's name and number are returned in the FileInfo structure of the m_RetrieveFile() command. File numbers are integers greater than zero.

File numbers are associated with files by the file directory retrieval functions (m_FilePattern() and m_RetrieveFile()). These functions are described in detail in the following sections. The association between a file number and a file exists only until the next file directory retrieval is performed (that is, m_FilePattern()).

m_FilePattern—Indicate files to be retrieved

File retrieval has two parts—selecting the group of files to be retrieved and performing the retrievals. Selecting the files is done by the m_FilePattern() function which tells Meridian Mail which files should be included in the actual retrieval.

Header files to include	m_acc.h (Constants, return codes) m_file.h (Declarations, constants, macros)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_FNAME_FORMAT Invalid filename format MME_DO_LOGON Must be logged on to use this command
See also	m_GetFileInfo m_RetrieveFile

Declaration

```
short m_FilePattern(FileName, ClassMap, IncludeStates,
            ExcludeStates, rc)
    char *FileName;        /* file to retrieve */
    short ClassMap;        /* bit map of classes to include */
    short IncludeStates;   /* bit map of states to include */
    short ExcludeStates;   /* bit map of states to exclude */
    short *rc;             /* returned status code */
```

The m_FilePattern function succeeds even if no files match the group of files specified for retrieval.

The parameters to m_FilePattern() include a pointer to a file name and three bit maps which specify the types of files to be retrieved. Only files that satisfy all three criteria (FileName, Class, states) are retrieved.

FileName "ANDed" with the classes and states to select the files to be retrieved. A null file name ("") indicates all files in the cabinet. The FileName should point to a null-terminated string. Maximum length equal to the field FNAME_SIZE as defined in m_acc.h.

The bit maps specify file classes and states for which the retrieval is to be performed. Both classes and states can be "ORed" together to produce combinations. An example would be "FS_URGENT | FS_INCOMING", which specifies that both urgent and incoming files are of interest.

ClassMap The file classes which can be specified are the following:

File class	Description
FC_ALL	Retrieve all files.
FC_VOICE	Retrieve simple voice files.
FC_VOICE_MESSAGE	Retrieve voice message files.
FC_VOICE_SEGMENT	Retrieve voice segment files.

IncludeStates/ExcludeStates These may be specified as either included or excluded so that many combinations of states are possible. File states, which are set by Meridian Mail and cannot be changed by Meridian ACCESS applications, consist of the following:

File states	Description
FS_ALL	All states together
FS_NONE	No states
FS_DELETED	Files marked deleted
FS_PRIVATE	Private files (cannot be forwarded)
FS_URGENT	Urgent (voice message) files
FS_INCOMING	Incoming (received) files
FS_ALTERED	Sent or read files
FS_ECONOMY	File sent by inexpensive communications route

m_RetrieveFile—File information

To retrieve file information, call m_RetrieveFile() repeatedly—each call retrieves one of the files matched by m_FilePattern(). The file information is placed into a structure. A pointer to the structure is passed as a parameter.

m_RetrieveFile returns TRUE on a successful retrieval and FALSE on an error or end of file list. The status code “rc” distinguishes an error from a normal end of file list. The status code MMS_OKAY indicates no more file information to be retrieved.

Header files to include	m_acc.h (Constants, return codes) m_file.h (Declarations, constants, macros)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_DO_LOGON Must be logged on to use this cmd MME_DO_FILEPAT Must call m_FilePattern() first
See also	m_GetFileInfo m_FilePattern

–continued–

Declaration

```
short m_RetrieveFile(FileRec, rc)
struct FileInfo *FileRec;                /* returned file data */
short *rc;  /* returned status code */

struct FileInfo {
char FileName[FNAME_SIZE];               /* name of retrieved file*/
short FileNum;                           /* corresponding file no.*/
char Class;                              /* CL_VOICE_MESSAGE, etc.*/
short ReadOnly;                          /* TRUE/FALSE */
short States;                            /* bit map of file states*/
short VoiceKBytes;                       /* # kilobytes of voice */
short TextKBytes;                        /* # kilobytes of text */
struct DATE CreateTime;                  /* creation date and time*/
struct DATE ModifyTime;                  /* last modification date*/
char Subject[SUBJECT_SIZE];              /* subject of file */
char ToFrom[TO_FROM_SIZE];               /* To: or From: name */
};
```

Class These can be set to one of the following values:

Class	Description
CL_VOICE	Simple voice file
CL_VOICE_MESSAGE	Voice message file
CL_VOICE_SEGMENT	Voice segment file

m_GetCabinetInfo—Cabinet information summary

The m_GetCabinetInfo() function provides summary information about the user's file cabinet. A variety of current statistics are returned in a structure filled in by Meridian Mail.

Header files to include	m_acc.h (Constants, return codes) m_file.h (Function declarations)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_DO_LOGON Must be logged on to use this cmd
See also	m_GetFileInfo m_FilePattern/m_RetrieveFiles

Declaration

```
short m_GetCabinetInfo(CabRec, rc)
    struct CabInfo *CabRec;                    /* returned info */
    short *rc;                                 /* returned status code */
```

The structure is given below. A pointer to this structure should be passed to the m_GetCabinetInfo() function.

```
struct CabInfo {
long int TextSpace;  /* text space avail. (1K blocks )*/
long int TextUsed;   /* text space currently in use*/
long int VoiceSpace; /* voice space avail. (8K blocks)*/
long int VoiceUsed;  /* voice space currently in use*/
short TotalFiles;    /* total # of files in cabinet */
short TotalDeleted;  /* total # of files marked deleted */
short TotalVM;       /* total # of Voice Messages */
short UnopVM;        /* # of unopened VMs */
short UnopUrgVM;     /* # of unopened, urgent VMs */
short UnsentVM;      /* # of unsent VMs */
};
```

m_CloseFile—Close a file

A file must be closed after it has been used.

Header files to include	m_acc.h (Constants, return codes) m_file.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_DO_LOGON Must be logged on to use this cmd MME_INVALID_HANDLE Invalid file handle passed to command MME_BAD_HANDLE Unassigned file handle MMW_COMMIT_IGNORED Read-only file: not committed MMW_BAD_COMMIT Invalid commit flag MMW_BAD_COMMAND Command invalid on this file type
See also	m_OpenFile m_Logoff m_Release m_CommitFile m_CreateFile

Declaration

```
short m_CloseFile(FileHandle, Commit, rc)
    short FileHandle;   /* handle of file to close */
    short Commit;       /* save changes?(TRUE/FALSE) */
    short *rc;          /* returned error code */
```

FileHandle The file handle number obtained from m_CreateFile(), m_OpenFile(), m_ForwardMsg(), m_PlayMsg(), m_ReplyMsg(), m_OpenGreeting, or m_OpenPersVerif().

Commit Is a flag indicating whether or not changes made to the file should be saved to disk. If FALSE is specified, any changes will be ignored. If a newly created file is closed without committing it, the file will not be created. A Commit can only be performed on a file which has been newly created or which was opened using m_OpenFile(), or m_OpenFileN in update mode.

m_CommitFile—Save file changes to disk

Changes made to files are normally saved to disk when the file is closed. This is done by the “Commit” parameter of the m_CloseFile() function. It is also possible to save changes made to an open file without closing the file. This is done through the m_CommitFile() function.

In an application where a file will remain open for a long time, it can be useful to commit the file changes to disk periodically.

A commit can only be performed on a file which has been newly created or on a file which was opened using m_OpenFile() in update mode.

Header files to include	m_acc.h (Constants, return codes) m_file.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MMW_COMMIT_IGNORED Read-only file: not committed MME_INVALID_HANDLE Invalid file handle passed to command MME_BAD_HANDLE Unassigned file handle MME_BAD_COMMAND Command invalid on this file type
See also	m_CloseFile

Declaration

```
short m_CommitFile(FileHandle, rc)
short FileHandle;                         /* handle of file to commit */
short *rc;  /* returned error code */
```

FileHandle The file handle number obtained from m_CreateFile(), m_OpenFile(), m_ForwardMsg(), m_PlayMsg(), m_ReplyMsg(), m_OpenGreeting, or m_OpenPersVerif().

m_CopyFile—Copy a file

These functions make a copy of a file. To copy a file by name, the m_CopyFile() function is used. These functions always create a new file of the same type as the original.

Header files to include	m_acc.h (Constants, return codes) m_file.h (Function declarations)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_FILE_DNE File does not exist MME_NO_MEMORY Out of memory MME_FNAME_FORMAT Invalid filename format MME_DO_LOGON Must be logged on to use this command
See also	m_RenameFile m_CreateFile m_FileExistCheck
–continued–	

Declaration

```
short m_CopyFile(From, To, rc)
    char *From;                    /* source name */
    char *To;                      /* destination name */
    short *rc;                     /* returned status code */
```

Files may also be copied by number, in which case the source (but not the destination) file is specified with a file number.

```
short m_CopyFileN(From, To, rc)
    short From;                    /* source number */
    char *To;                      /* destination name */
    short *rc;                     /* returned status code */
```

From This should point to a null-terminated string for m_CopyFile(), maximum length equal to the field FNAME_SIZE as defined in m_acc.h; and to the file number obtained from m_RetrieveFile() for m_CopyFileN().

To If a file with the “To” name already exists, a new file will be created with the same name. (Remember, file names are not necessarily unique in the Meridian Mail file system.) The m_FileExistCheck() function should be used to determine if a file with the “To” name already exists. Maximum length FNAME_SIZE as defined in m_acc.h.

m_CreateFile—Create a new file

This function creates a new file of a specified type and opens it for writing. If a file with the given name already exists, a new file will be created with the same name. The m_FileExistCheck() function should be used to determine if a file with the given name already exists.

Header files to include	m_acc.h (Constants, return codes) m_file.h (Function declarations)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_DO_LOGON Must be logged on to use this cmd MME_MAX_OPEN Maximum open file limit reached MME_FNAME_FORMAT Invalid filename format
See also	m_CopyFile m_RenameFile m_FileExistCheck

Declaration

```
short m_CreateFile(FileName, FileType, FileHandle, rc)
    char *FileName;                          /* name of new file */
    short FileType;                          /* type of file to create */
    short *FileHandle;                       /* returned file handle no. */
    short *rc;                               /* returned status code */
```

FileName This should point to a null-terminated string containing the name of the file to create. For the purpose of voice operations, the position of a newly created/opened voice file is at the beginning. Maximum length equal to the field FNAME_SIZE as defined in m_acc.h.

FileType The FileType may be one of the following:

File Type	**Description**
CL_VOICE	simple voice file
CL_VOICE_MESSAGE	voice message file
CL_VOICE_SEGMENT	voice segment file

m_DeleteFile—Delete a file

A file can be deleted by name or by number. If the file is currently open, an attempt to delete it will fail.

Header files to include	m_acc.h (Constants, return codes) m_file.h (Function declarations)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_FILE_OPEN File is open MME_DO_LOGON Must be logged on to use this command MME_BAD_FLAG Invalid flag MME_FNAME_FORMAT Invalid filename format
See also	m_UndeleteFile m_CloseFile m_RetrieveFile

Declaration

```
short m_DeleteFile(FileName, Immediate, rc)
    char *FileName;    /* name of file to delete */
    short Immediate;   /* physical deletion? (TRUE/FALSE) */
    short *rc;         /* returned status code */

short m_DeleteFileN(FileNum, Immediate, rc)
    short FileNum;     /* number of file to delete */
    short Immediate;   /* physical deletion? (TRUE/FALSE) */
    short *rc;         /* returned status code */
```

FileName This should point to a null-terminated string containing the name of the file to delete. Maximum length equal to the field FNAME_SIZE as defined in m_acc.h.

FileNum (of m_DeleteFileN()) is the file number obtained from m_RetrieveFile().

Immediate Deletion becomes effective when the Meridian Mail account is logged off unless the “Immediate” parameter is specified as TRUE. Until then, the m_UndeleteFile() function may be used to recover it. A file which has been marked deleted (but has not been physically deleted) can still have file operations performed on it.

m_FileExistCheck—Check if a File Exists

This function will check if a file with a given name exists in the current cabinet. Since several files can have the same name, this function will indicate if there is at least one such file.

> ***Note:*** The return value from the function indicates whether or not the function succeeded, not whether or not the file exists. To check for the existence of the file, use the returned “Exists” parameter.

Header files to include	m_acc.h (Constants, return codes) m_file.h (Function declarations)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_DO_LOGON Must be logged on to use this cmd

Declaration

```
short m_FileExistCheck(FileName, Exists, rc)
    char *FileName;    /* name of file to check for */
    short *Exists;     /* returned - named file exists? */
    short *rc;         /* returned status code */
```

FileName This should point to a null-terminated string. Maximum length equal to the field FNAME_SIZE as defined in m_acc.h.

Exists (returned) is set to TRUE if a file with the given name exists and FALSE otherwise.

m_GetFileInfo—Information on a single file

This function retrieves the file information for a single specified file. The information is placed into a structure, a pointer to which is passed as a parameter. See the m_FilePattern() and m_RetrieveFile() functions for a description of the returned structure.

To retrieve the information for a file by name, the m_GetFileInfo() function will return information on the first file found with the given name. When retrieving file information by name, the field in the returned FileInfo structure pertaining to the file number is meaningless and is always set to 0.

Header files to include	m_acc.h (Constants, return codes) m_file.h (Function declarations)
Prerequisites	Registered Acquired Logged on File pattern
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_DO_LOGON Must be logged on to use this cmd MME_DO_FILEPAT Must call m_FilePattern first MME_FILE_DNE File does not exist MME_FNAME_FORMAY Invalid filename format
See also	m_GetCabinetInfo m_FilePattern/m_RetrieveFile
–continued–	

Declaration

```
short m_GetFileInfo(FileName, FileRec, rc)
    char *FileName;                 /* name of file to retrieve */
    struct FileInfo *FileRec;       /* returned file info */
    short *rc;                      /* returned status code */
```

Information on a file can also be retrieved by number:

```
short m_GetFileInfoN(FileNum, FileRec, rc)
    short FileNum;                  /* no. of file to retrieve */
    struct FileInfo *FileRec;       /* returned file info */
    short *rc;                      /* returned status code */
```

FileName This should point to a null-terminated string. Maximum length equal to the field FNAME_SIZE as defined in m_acc.h.

FileNum The FileNum is the file number obtained from m_RetrieveFile() for m_GetFileInfoN().

For the purpose of voice operations, the position of a newly created/opened voice file is at the beginning.

m_OpenFile—Open a file

Files must be opened before they can be read, written, played, or have any other operation performed on them. A file must already exist for it to be opened by this function. To create a new file, the m_CreateFile() function should be used.

Header files to include	m_acc.h (Constants, return codes) m_file.h (Function declarations)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_READ_MODE Cannot open Read file in Update mode MME_FILE_OPEN File is already open MME_MAX_OPEN Maximum open file limit reached MME_FILE_DNE File does not exist MME_BAD_MODE Invalid file access mode used MME_DO_LOGON Must be logged on to use this command
See also	m_CloseFile m_CreateFile m_CommitFile m_RetrieveFile

–continued–

Declaration

```
short m_OpenFile(FileName, Mode, FileHandle, rc)
char *FileName;                    /* name of file to open */
char *Mode;                        /* type of open: “r” or “u” */
short *FileHandle;                 /* returned file handle no. */
short *rc;  /* returned status code */
```

To open a file by number, a separate function is used.

```
short m_OpenFileN(FileNum, Mode, FileHandle, rc)
short FileNum;                     /* number of file to open */
char *Mode;                        /* type of open: “r” or “u” */
short *FileHandle;                 /* returned file handle no. */
short *rc;  /* returned status code */
```

A limited number of files may be open simultaneously based on the amount of Meridian Mail server resources available. Typically, two or three files may be open at the same time.

For the purposes of voice operations, the position in a newly opened voice file is the beginning.

FileNum the file number obtained from m_RetrieveFile().

Mode used to specify if the file is to be opened for reading or writing. A file may have many readers but only one writer at any given time.

File mode	Description
r	read-only
u	read or write (update)

FileHandle This is the returned value which is used by all other file functions when referring to an opened file.

FileName This should point to a null-terminated string. Maximum length equal to the field FNAME_SIZE as defined in m_acc.h.

m_RenameFile—Rename a file

A file can be renamed by the RenameFile() function. If the file is currently open, an attempt to rename it will fail. A file number does not change when a file is renamed.

Header files to include	m_acc.h (Constants, return codes) m_file.h (Function declarations)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_FILE_OPEN File is already open MME_FILE_DNE File does not exist MME_FNAME_FORMAT Invalid filename format MME_DO_LOGON Must be logged on to use this command
See also	m_CopyFile m_CreateFile m_FileExistCheck m_RetrieveFile

Declaration

```
short m_RenameFile(From, To, rc)
    char *From;                            /* original name */
    char *To;                              /* new name */
    short *rc;                             /* returned status code */
```

Files can also be renamed by number, in which case the original (but not the new) file is specified with a file number.

```
short m_RenameFileN(From, To, rc)
    short From;                            /* original number */
    char *To;                              /* new name */
    short *rc;                             /* returned status code */
```

From For m_RenameFileN, the file number obtained from m_RetrieveFile(). “From” should point to a null-terminated string. Maximum length equal to the field FNAME_SIZE as defined in m_acc.h.

To If a file with the “To” name already exists, a file with a duplicate name will result. The m_FileExistCheck() function should be used to check if a file with the “To” name already exists. “To” should point to a null-terminated string. Maximum length equal to the field FNAME_SIZE as defined in m_acc.h.

m_SetFileSubject—Add a Subject Field to a File

Each file may have attached to it a field that describes the subject of the file. The subject is added by using this function.

Header files to include	m_acc.h (Constants, return codes) m_file.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_INVALID_HANDLE Invalid file handle passed to command MME_BAD_HANDLE Unassigned file handle MME_FILE_DNE File does not exist MME_READ_ONLY Cannot do command, read-only flag MME_BAD_SUBJECT Invalid subject string MME_BAD_COMMAND Command invalid on this file type MME_DO_LOGON Must be logged on to use this command
See also	m_RetrieveFile m_GetFileInfo m_OpenFile m_CreateFile

Declaration

```
short m_SetFileSubject(FileHandle, Subject, rc)
    short FileHandle;    /* file to attach subject to */
    char *Subject;       /* subject of file */
    short *rc;           /* returned status code */
```

FileHandle The file handle number obtained from m_CreateFile(), m_OpenFile(), m_ForwardMsg(), m_PlayMsg(), m_ReplyMsg(), m_OpenGreeting(), or m_OpenPersVerif().

Subject The maximum length is defined by the constant SUBJECT_SIZE.

m_UndeleteFile—Recover a File

This function recovers a file which has been marked as deleted. If the file is currently open, an attempt to undelete it will fail.

If a file has been physically deleted, it cannot be recovered. See the m_DeleteFile() function for information on physical deletion of a file.

Header files to include	m_acc.h (Constants, return codes) m_file.h (Function declarations)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_FILE_OPEN File is open MME_FILE_DNE File does not exist MME_FNAME_FORMAT Invalid filename format MME_DO_LOGON Must be logged on to use this command
See also	m_DeleteFile

Declaration

```
short m_UndeleteFile(FileName, rc)
    char *FileName;    /* name of file to recover */
    short *rc;         /* returned status code */

short m_UndeleteFileN(FileNum, rc)
    short FileNum;     /* number of file to recover */
    short *rc;         /* returned status code */
```

FileNum The file number obtained from m_RetrieveFile().

FileName This should point to a null-terminated string. Maximum length equal to the field FNAME_SIZE as defined in m_acc.h.

Chapter 8: Voice operation functions

Voice operations are functions which are used to manipulate a voice channel. They allow voice files to be played or recorded to/from a telephone set. This section describes these operations.

Function	**Description**
m_PlayVoice	Play a voice file
m_RecordVoice	Record into a voice file
m_SkipVoice	Skip within a file
m_StopVoice	Stop playing or recording voice

m_PlayVoice—Play a file

An open voice file can be played by calling the m_PlayVoice() function on the file. In the basic version, the file is played either from the beginning or from the current position onwards. The extended function provides more control over where the playing will begin and end.

This function should be used for playing all voice files except voice segment files (which use the m_PlaySegs() function).

Unless the m_StopVoice() function is executed, the position in the file when the m_PlayVoice() is complete will be at the end of the file.

Header files to include	m_acc.h (Constants, return codes) m_voice.h (Function declarations, constants)
Prerequisites	Registered Acquired Logged on File open Connected
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_NO_ACTV_CHNL No active voice channel MME_BAD_POSITION Bad FromPos MMS_AT_EOF End of playing reached MME_PLAYING Playing already in progress MME_BAD_COMMAND Command invalid on this file type

–continued–

Return codes (continued)	MME_BAD_TO_POS Bad ToPos MME_CHAN_IN_USE Voice channel already in use MME–BAD_SEQUENCE Invalid command sequence MMS_NO_VOICE No voice in segment to play MME_DO_LOGON Must be logged on to use this command
See also	m_StopVoice m_SkipVoice m_PlayMsg m_PlaySegs
Events	m_OnPlayEnd

Declaration

```
short m_PlayVoice(FileHandle, Restart, rc)
    short FileHandle;  /* file to play */
    short Restart;     /* play from beginning? (TRUE/FALSE) */
    short *rc;         /* returned status code */
```

The extended function:

```
short m_PlayVoiceX(FileHandle, FromPos, ToPos, rc)
    short FileHandle;               /* file to play */
    short FromPos;                  /* pos'n to start playback */
    short ToPos;                    /* position to end playback */
    short *rc;                      /* returned status code */
```

FileHandle The file handle number obtained from m_CreateFile() m_OpenFile(), m_ForwardMsg(), m_PlayMsg(), m_ReplyMsg(), m_OpenGreeting(), or mOpenPersVerif().

FromPos FromPos can have the following values:

FromPos parameter	**Description**
PV_FROM_BOF	Play from beginning-of-file
PV_FROM_BOS	Play from beginning-of-segment (currently not supported)
PV_FROM_CUR	Play from current position in file

ToPos ToPos can have the following values:

ToPos parameter	**Description**
PV_TO_EOS	Play to end-of-segment (currently not supported)
PV_TO_EOF	Play to end-of-file

These functions initiate the playing of a file and return immediately. When the playback of a file is complete, either because it has played to the end or because the m_SkipVoice() function has skipped to the beginning or end of the file, the m_OnPlayEnd() event is generated.

m_RecordVoice—Record into a file

Voice can be recorded by calling the m_RecordVoice() function on an open file. The file must have been opened for writing (updating).

The basic function provides a simple interface for recording a voice file. Such a file has a maximum size which is set by the Meridian Mail administrator. An audible beep is heard over the phone when recording starts. Also, an end-of-recording warning beep is heard during recording when 80% of the maximum size has been recorded.

An extended function provides many additional options for controlling the recording.

Recording can be stopped by m_StopVoice() and may be continued by issuing another m_RecordVoice(). This function uses the RV_APPEND option as the recording position—see below.

An m_OnRecordEnd event is generated when the recording is complete.

Header files to include	m_acc.h (Constants, return codes) m_voice.h (Function declarations, constants)
Prerequisites	Registered Acquired Logged on File open Connected
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_NO_ACTV_CHNL No active voice channel MME_BAD_RECORD_POS Invalid recording position MME_RECORDING Recording already in progress
–continued–	

Return codes (continued)	MME_BAD_POSITION Invalid voice start position MME_BAD_RECORDING_POS Invalid recording position MME_BAD_HANDLE Unassigned file handler MME_READ_ONLY Can't do command on read only file MME_DO_LOGON Must be logged on to use this command
See also	m_StopVoice m_SkipVoice
Events	m_OnRecordEnd

Declaration

```
short m_RecordVoice(FileHandle, Restart, rc)
    short FileHandle;  /* file to record to */
    short Restart;  /* record from beginning? (TRUE/FALSE) */
    short *rc;      /* returned status code */
```

Extended function:

```
short m_RecordVoiceX(FileHandle, Restart, RecordPos,
MaxSize, rc)
    short FileHandle;  /* file to record to */
    short Restart;  /* begin @ start of segment? TRUE/FALSE */
    short RecordPos;   /* recording position: RV_xxx */
    short MaxSize;     /* max. size of item being recorded
                       (secs)*/
    short *rc;         /* returned status code */
```

FileHandle The file handle number is obtained from m_CreateFile() or m_OpenFile().

RecordPos—can have the following value:

RecordPos parameter	Description
RV_APPEND	Insert within segment; delete trailing voice

MaxSize If MaxSize is larger than the maximum set by the Meridian Mail administrator, then the latter will be used without warning. A MaxSize of zero (0) indicates that the system maximum should be used.

m_SkipVoice—Skip within segment

During the playback of a voice file the current position within the file can be changed. This allows the application to skip part of the playback or to replay a part of the file which has just been played.

Header files to include	m_acc.h (Constants, return codes) m_voice.h (Function declarations, constants)
Prerequisites	Registered Acquired Logged on File open Connected Playing
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_NO_ACTV_CHNL No active voice channel MME_BAD_SEQUENCE Invalid command sequence: must be playing
See also	m_PlayVoice m_StopVoice
Events	m_OnPlayEnd

Declaration

```
short m_SkipVoice(Time, Forward, rc)
    unsigned short Time;    /* centiseconds to skip */
    short Forward;       /* skip forward? (TRUE/FALSE) */
    short *rc;           /* returned status result */
```

Forward This determines the direction of the skip—forward or backward. Skipping past the beginning or end of a file is the same as having completed the playback. (Playback halts, and an m_OnPlayEnd() event is sent to the application.)

m_StopVoice—Stop playing/recording

This function halts a voice operation (any playing or recording of voice). The current position is left unchanged so that a subsequent play or record command can continue where the previous one ended. Since the playback has not completed, no m_OnPlayEnd() event will be generated.

When voice segments are being played (m_PlaySegs()), m_StopVoice() will abort the playback and clear out any unplayed voice segment IDs from the play queue. In this case the playback cannot be resumed from the position where it was stopped. No m_OnPlayEnd() event will be generated.

Header files to include	m_acc.h (Constants, return codes) m_voice.h (Function declarations, constants)
Prerequisites	Registered Acquired Logged on File open Connected Playing/recording
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_NO_ACTV_CHNL No active voice channel MME_BAD_SEQUENCE No play/record currently in progress
See also	m_PlayVoice m_RecordVoice m_SkipVoice
Events	m_OnPlayEnd m_OnRecordEnd

Declaration

```
short m_StopVoice(rc)
    short *rc;                                /* returned status result */
```

Chapter 9: Messaging functions

The general term “messaging” refers to the sending of information from one user to another. In Meridian Mail, this is divided into the following two categories:

- **Voice messaging**– is the core function of the Meridian Mail system.
- **External messaging** –is the ability of the Meridian Mail software to keep track of messages which are external to Meridian Mail. Such messages include text messages from a third-party computing system, FAX messages, and other message types where the message is not actually stored in Meridian Mail.

Voice messaging

Voice Messaging is the sending of messages recorded as voice message files. A voice message is a file in a special format. Functions are provided to manage this format so that the detailed internal format need not be known by an application.

Function	Description
m_AddBoxToAddr	Address message
m_AddNameToAddr	Address message
m_AddrPattern	Get list of receivers
m_RetrieveAddr	Get list of receivers
m_CallMsgSender	Call sender (via the phone)
m_DeleteFromAddr	Delete receiver from list

–continued–

Function	**Description**
m_ForwardMsg	Forward message
m_PlayMsg	Play message
m_ReplyMsg	Reply to sender (via a message)
m_SenderAddr	Retrieve the sender of a message
m_SendMsg	Send a message

A new message file can be created by the m_CreateFile() function. An existing message can be opened and played using the m_PlayMsg() function. The subject of the file (and message) is set by the m_SetFileSubject() function.

m_AddBoxToAddr—Address a message

A message can be addressed by any combination of receiver names or mailbox numbers. In both cases, the addressee's full name and mailbox number are returned in a standard format.

This function should be called after creating a new message with m_CreateFile(), but before sending the message with m_SendMsg(). It may be called repeatedly to address the message to several people.

Header files to include	m_acc.h (Constants, return codes) m_msg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_BAD_RCVR The given MailBox was not found MME_MULTIMATCH MailBox matches several users MME_BAD_HANDLER Invalid file handle MME_BAD_COMMAND Command invalid on this file type MME_MAX_PDL_ENTRIES Exceeded number of maximum PDL entries MME_BAD_BOX Invalid box number MME_NOT_NUMERIC Non-numeric in numeric field MME_DO_LOGON Must be logged on to use this command

–continued–

Declaration

```
short m_AddBoxToAddr(FileHandle, MailBox, FullName, FullBox, rc)
    short FileHandle;                /* file to address */
    char *MailBox;                   /* mailbox to send to */
    char *FullName;                  /* returned full user name */
    char *FullBox;                   /* returned mailbox number */
    short *rc;                       /* returned status code */
```

MailBox If this matches more than one user, the function will fail with a returned status code of MME_MULTIMATCH. Should be large enough to accept strings up to BOX_SIZE in length as defined in m_acc.h.

FullName/FullBox This should be large enough to accept strings up to FULLNAME_SIZE and BOX_SIZE in length, respectively as defined in m_acc.h.

m_AddNameToAddr—Address a message

A message can be addressed by any combination of receiver names or mailbox numbers. In both cases, the addressee's full name and mailbox number are returned in a standard format.

This function should be called after creating a new message with m_CreateFile(), but before sending the message with m_SendMsg(). It may be called repeatedly to address the message to several people.

Header files to include	m_acc.h (Constants, return codes) m_msg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_BAD_RCVR The given Name was not found MME_MULTIMATCH Name matches several users MME_BAD_HANDLE Invalid file handle MME_BAD_COMMAND Command invalid on this file type MME_MAX_PDL_ENTRIES Exceeded maximum number of PDL entries MME_BAD_GIVEN Invalid firstname MME_BAD_SURNAME Invalid lastname MME_DO_LOGON Must be logged on to use this command
–continued–	

Declaration

```
short m_AddNameToAddr(FileHandle, Name, FullName, FullBox, rc)
    short FileHandle;                /* file to address */
    char *Name;                      /* user name to send to */
    char *FullName;                  /* returned full user name */
    char *FullBox;                   /* returned mailbox number */
    short *rc;                       /* returned status code */
```

Name Name can be specified in a variety of formats and may contain wildcard characters. See Chapter 2: “Meridian Mail Facilities” for details. If the specified Name matches more than one user, the function will fail with a returned status code of MME_MULTIMATCH fullname. Should be large enough to accept strings up to FULLNAME_SIZE in length as defined in m_acc.h

FullName/FullBox This should be large enough to accept strings up to FULLNAME_SIZE and BOX_SIZE in length, respectively as defined in m_acc.h.

m_AddrPattern—List receivers

The receiver list which is created with the m_AddBoxToAddr() and m_AddNameToAddr() functions can be queried using m_AddrPattern() and m_RetrieveAddr(). To start the retrieval, the m_AddrPattern() function should be called.

Header files to include	m_acc.h (Constants, return codes) m_msg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_BAD_HANDLE Invalid file handle MME_BAD_COMMAND Command invalid on this file type MME_DO_LOGON Must be logged on to use this command
See also	m_AddBoxToAddr m_AddNameToAddr m_RetrieveAddr

Declaration

```
short m_AddrPattern(FileHandle, rc)
    short FileHandle; /* file to get addresses from */
    short *rc;        /* returned status code */
```

FileHandle The file handle number obtained from m_CreateFile() or m_OpenFile().

m_RetrieveAddr—List receivers

To retrieve all of the addresses, the m_RetrieveAddr() function should be called repeatedly. Each invocation retrieves one message receiver.

m_RetrieveAddr() returns TRUE on a successful retrieval and FALSE on an error or at end-of-list. The status code “rc” distinguishes an error from a normal end-of-list (status code of MMS_OKAY indicates end-of-list).

Header files to include	m_acc.h (Constants, return codes) m_msg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_DO_ADDRPAT Must call m_AddrPattern() first MME_BAD_HANDLE Invalid file handle MME_BAD_COMMAND Command invalid on this file type MME_DO_LOGON Must be logged on to use this command
See also	m_AddBoxToAddr m_AddNameToAddr m_AddrPattern

Declaration

```
short m_RetrieveAddr(FileHandle, FullName, MailBox, rc)
    short FileHandle;   /* file to get addresses from */
    char *FullName;     /* returned user’s name */
    char *MailBox;      /* returned user’s mailbox */
    short *rc;          /* returned status code */
```

FileHandle The file handle number obtained from m_CreateFile() or m_OpenFile().

FullName The name should be large enough to accept strings up to FULLNAME_SIZE in length.

FullBox Thisshould be large enough to accept strings up to BOX_SIZE in length.

m_CallMsgSender—Call sender of message

The sender of a voice message will be called (on the telephone) by this function. This provides the ability to “call back” the person who left a message.

The function takes a file handle rather than a file name, and so operates only on an open voice message file (such as one that has just been played).

This function returns TRUE if successful. Otherwise, it returns FALSE and the status code may be checked to determine the nature of the problem.

When m_CallMsgSender() succeeds, the voice connection to Meridian Mail no longer exists. The application must re-establish the connection with m_MakeCall(), or wait for another incoming call, before subsequent voice operations may be performed.

Header files to include	m_acc.h (Constants, return codes) m_msg.h (Function declarations) m_voice.h (Constants)
Prerequisites	Registered Acquired Logged on File open Connected
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_NO_ACTV_CHNL No active voice channel MME_BAD_DN DN is invalid MME_NOT_RECEIVED Cannot call sender of an outgoing message

–continued–

Return codes (continued)	MME_OTHER_TELEPHONY Other telephony operation in progress MME_DETECT_INPROG Voice/silence detection in progress MME_BAD_HANDLE Invalid file handle MME_BAD_COMMAND Command invalid on this file type MME_NOT_RECEIVED Cannot call sender of an outgoing message MME_SYS_MSG Operations invalid on system messages
See also	m_MakeCall m_ReplyMsg m_SenderAddr m_TransferCall
Events	m_OnCallProgress

Declaration

```
short m_CallMsgSender(FileHandle, TelephonyReturn, MaxTime, rc)
short FileHandle;      /* file to get DN from */
short TelephonyReturn; /* when to return from this
                          operation*/
unsigned short MaxTime;  /* max. time for completion (secs) */
short *rc;             /* returned status code */
```

FileHandle The file handle number obtained from m_CreateFile(), m_OpenFile(), m_ForwardMsg(), m_PlayMsg(), or m_ReplyMsg().

TelephonyReturn The possible values are discussed in the introduction to Chapter 6: “Telephony Functions”. If it specifies TR_IMMEDIATE, the application must check for m_OnCallProgress() events to determine the state of the call. When TR_ON_COMPLETE is specified, the application will not receive Call Progress events until the function is complete, since the CallProgress event handler will be temporarily replaced.

MaxTime The MaxTime specifies the maximum time the function should wait for the operation to complete. The value should be MIN_RING_TIME or greater to ensure that the called telephone set has enough time to ring. If TR_IMMEDIATE has been specified, the MaxTime parameter is ignored.

m_DeleteFromAddr—Remove receiver of message

A person listed as a receiver of a voice message will be deleted from the receiver list with this function. This allows an unsent message to have its distribution list reduced before it is sent.

This function deletes only the first occurrence of MailBox in the receiver list.

Header files to include	m_acc.h (Constants, return codes) m_msg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_BAD_RCVR The given MailBox was not found MME_BAD_HANDLE Invalid file handle MME_READ_ONLY Command cannot be performed on read-only file MME_BAD_COMMAND Command invalid on this file type MME_BAD_BOX Invalid box number
See also	m_AddBoxToAddr m_AddNameToAddr m_AddrPattern m_RetrieveAddr

Declaration

```
short m_DeleteFromAddr(FileHandle, MailBox, rc)
    short FileHandle;                    /* file containing message */
    char *MailBox;                       /* address to delete */
    short *rc;                           /* returned status code */
```

FileHandle The file handle number obtained from m_CreateFile(), m_OpenFile(), m_ForwardMsg(), m_PlayMsg(), or m_Reply_Msg.

MailBox This should be large enough to accept strings up to BOX_SIZE in length.

m_ForwardMsg—Forward received message

This function creates a new message file (with the name specified in NewFileName) and a copy of the original message appended to it. It then opens this message file and returns a file handle that can be used with the other messaging functions such as m_AddBoxToAddr(), m_SendMsg(), and m_RecordVoice(). The appended original message cannot be modified.

> ***Note:*** A forwarded message is always a copy of an existing message—the original is left unchanged in the user's cabinet.

Messages may also be forwarded using the FileNum file number obtained from m_RetrieveFile().

Header files to include	m_acc.h (Constants, return codes) m_msg.h (Function declarations)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_FORWARD_PRIVATE Cannot forward a private message MME_DO_LOGON Must be logged on to use this command MME_FNAME_FORMAT Invalid filename format MME_MAX_OPEN Maximum open file limit reached MME_FILE_DNE File does not exist MME_FILE_NOT_MSG File is not a message file MME_SYS_MSG Operations invalid on system messages
–continued–	

See also	m_CreateFile m_ReplyMsg

Declaration

```
short m_ForwardMsg(FileName, NewFileName, FileHandle, rc)
char *FileName;        /* name of file to forward */
char *NewFileName;     /* new message file to create */
short *FileHandle;     /* returned file handle */
short *rc;             /* returned status code */

short m_ForwardMsgN(FileNum, NewFileName, FileHandle, rc)
short FileNum;         /* number of file to forward */
char *NewFileName;     /* new message file to create */
short *FileHandle;     /* returned file handle */
short *rc;             /* returned status code */
```

FileName, **NewFileName** These should point to null-terminated strings. of maximum length FNAME_SIZE as defined in m_acc.h.

m_PlayMsg—Play a voice message

This is a specialized version of the m_PlayVoice() function. The m_PlayMsg function opens a voice message file, plays the message, and optionally deletes the file. If you specify “delete”, the file is flagged for deletion. To close a file, whether or not it is flagged for deletion, you must perform the m_CloseFile function. The m_PlayMsg function performs the four most common voice operations (m_OpenFile(), m_PlayVoice(), m_CloseFile(), and m_DeleteFile()) in a single transaction. These functions return a file handle for use by the playback editing functions.

Playback editing functions such as m_SkipVoice() and m_StopVoice() can be used during the playback of the voice message. An m_OnPlayEnd() event will be generated when the playback is complete.

Only one m_PlayMsg() file can be active at a time. If another m_PlayMsg() command is issued while an existing command is outstanding, the existing command is terminated and the new one begins. Only one event will be generated in this case when the playback of the new command is complete.

Header files to include	m_acc.h (Constants, return codes) m_msg.h (Function declarations)
Prerequisites	Registered Acquired Logged on Connected
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_NO_ACTV_CHNL No active voice channel MME_DO_LOGON Must be logged on to use this command MME_BAD_FLAG Invalid flag
–continued–	

Return codes	MME_CHAN_IN_USE Voice channel already in use MME_PLAYING Play command already in progress MME_BAD_SEQUENCE Invalid command sequence MMS_NO_VOICE No voice segment to play MMS_AT_EOS At end of voice segment MME_DETECT_IN_PROGRESS Sound detect already in progress MMS_AT_EOF Reached end of file MME_FILE_DNE File does not exist MME_FILE_NOT_MSG File is not a message file
See also	m_PlayVoice m_DeleteFile m_StopVoice m_SkipVoice
Events	m_OnPlayEnd

Declaration

```
short m_PlayMsg(FileName, Delete, FileHandle, rc)
    char *FileName;      /* name of file to play */
    short Delete;        /* delete after play? (TRUE/FALSE) */
    short *FileHandle;   /* returned file handle */
    short *rc;           /* returned status code */
```

There is also a numbered version of m_PlayMsg()

```
short m_PlayMsgN(FileNum, Delete, FileHandle, rc)
    short FileNum;       /* number of file to play */
    short Delete;        /* delete after play? (TRUE/FALSE) */
    short *FileHandle;   /* returned file handle */
    short *rc;           /* returned status code */
```

FileNum The file number obtained from m_RetrieveFile().

FileName The file name should point to a null-terminated string of maximum length FNAME_SIZE as defined in m_acc.h.

m_ReplyMsg—Reply to a received message

A received voice message may be replied to by another voice message. (To reply by telephone, see the m_CallMsgSender() function.) A new message file is created, which is automatically addressed to the sender of the original message. The new file is opened and a file handle is returned so that the reply may be recorded into the voice message file.

Since this function operates on closed files, a numbered version is also provided.

Header files to include	m_acc.h (Constants, return codes) m_msg.h (Function declarations)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_NOT_RECEIVED Cannot reply to an outgoing message MME_EXTERNAL Cannot reply to an external message MME_REMOTE Unknown remote site MME_BROADCAST Cannot reply to a broadcast message MME_DO_LOGON Must be logged on to use this command MME_BAD_FLAG Invalid flag MME_FNAME_FORMAT Invalid filename format MME_MAX_OPEN Maximum open file limit reached
–continued–	

Return codes	MME_FILE_NOT_MSG File is not a message file MME_BAD_RCVR Invalid receiver in address list MME_NOT_RECEIVED Not a received message MME_SYS_MSG Operation invalid or system message MME_AMIS_REPLY Cannot reply all on AMIS message
See also	m_CreateFile m_ForwardMsg m_CallMsgSender

Declaration

```
short m_ReplyMsg(FileName, NewFileName, All, FileHandle, rc)
    char *FileName;        /* name of file to reply to */
    char *NewFileName;     /* new message file to create */
    short All;    /* reply to all receivers? (TRUE/FALSE) */
    short *FileHandle;     /* returned file handle */
    short *rc;             /* returned status code */

short m_ReplyMsgN(FileNum, NewFileName, All, FileHandle, rc)
    short FileNum;         /* number of file to forward */
    char *NewFileName;     /* new message file to create */
    short All;    /* reply to all receivers? (TRUE/FALSE) */
    short *FileHandle;     /* returned file handle */
    short *rc;             /* returned status code */
```

FileName, NewFileName These should point to null-terminated strings, maximum length FNAME_SIZE as defined in m_acc.h.

All If TRUE, then all receivers of the original message will also receive the reply.

FileNum The file number obtained by m_RetrieveFile().

m_SenderAddr—Get sender of message

The sender of a message can be extracted from the message itself by this function. The sender and the sender's mailbox number are returned to the calling process.

Header files to include	m_acc.h (Constants, return codes) m_msg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_NO_ACTV_CHNL No active voice channel MME_BAD_HANDLE Unassigned file handle MME_BAD_COMMAND Command invalid on this file type
See also	m_CallMsgSender m_ReplyMsg m_RetrieveFile m_GetFileInfo

Declaration

```
short m_SenderAddr(FileHandle, FullName, MailBox, rc)
    short FileHandle;    /* file to retrieve name from */
    char *FullName;      /* returned sender's name */
    char *MailBox;       /* returned sender's mailbox */
    short *rc;           /* returned status code */
```

FileHandle The file handle number obtained from m_CreateFile(), m_OpenFile(), m_ForwardMsg(), m_PlayMsg(), or m_ReplyMsg().

FullName This should be large enough to accept strings up to FULLNAME_SIZE in length.

MailBox This should be large enough to accept strings up to BOX_SIZE in length.

m_SendMsg—Send a message

After a message has been created, addressed, recorded, and perhaps given a subject, it must be sent. Sending a message closes the file and marks the file deleted. An extended version of the m_SendMsg() function provides for additional control over the sending process.

Header files to include	m_acc.h (Constants, return codes) m_msg.h (Function declarations, constants)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_EMPTY_MSG Cannot send an empty message MME_INCOMING Cannot send incoming message MME_NEED_RCVR Message not addressed MME_MAX_DELAY Delay delivery time too long MME_BAD_HANDLE Unassigned file handle MME_BAD_COMMAND Command invalid on this file type

Declaration

```
short m_SendMsg(FileHandle, rc)
    short FileHandle;    /* file to send */
    short *rc;           /* returned status code */

short m_SendMsgX(FileHandle, Priority, Tags, Date, rc)
    short FileHandle;    /* file to send */
    short Priority;      /* priority of msg delivery */
    short Tags;          /* auxiliary mail features */
    struct DATE *Date;   /* earliest allowable delivery */
    short *rc;           /* returned status code */
```

FileHandle The file handle number obtained from m_CreateFile(), m_OpenFile(), m_ForwardMsg(), m_PlayMsg(), or m_ReplyMsg().

Priority The priority determines how quickly the message is to be delivered. The states (“urgent”, for example) associated with a file are returned by the m_RetrieveFile() and m_GetFileInfo() functions. Only one delivery priority can be specified.

Priority	Description
SP_NORMAL	Normal delivery; specified if neither Urgent or Economy apply
SP_URGENT	Urgent (high priority) delivery across a network
SP_ECONOMY	Messages held for delivery at lower-cost time across a network

Tags Tags allow special mailing features to be attached to the message. These tags cause some special processing to be performed at the point of delivery. The tags are as follows:

Tag	Description
ST_NONE	No special tags
ST_ACKNOWLEDGE	Sender receives an acknowledge-ment message when the message is first opened by each recipient
ST_PRIVATE	Indicates that the contents are confidential; the message cannot be forwarded

The acknowledgement message will be an empty message from the recipient, with the same subject as the original message but prefixed with “Ack:”.

A message may have several tags. For example, a message may be both private and produce an acknowledgment of receipt by specifying “ST_PRIVATE | ST_ACKNOWLEDGE” as the message tag.

Date This specifies the earliest allowable delivery date and time for the message. The maximum interval allowed for a delayed delivery is set by the Meridian Mail administrator. If the date specified exceeds this interval, then the function will fail with MME_MAX_DELAY. If the current date is to be used, the m_NullDate() function can be specified. For example:

```
struct DATE *m_NullDate(void)
```

External Messaging

External messaging provides Meridian ACCESS application programs with the ability to keep track of text, FAX, and other messages which neither originate nor terminate on the Meridian Mail system. This is done by setting and retrieving counters for various external message types, and by requesting notification (on the phone) of the arrival of the corresponding external messages.

The Meridian Mail system will ensure that the user’s Message Waiting Indicator (MWI) is turned on while the total of all message counters for enabled message types is non-zero. Likewise, the MWI is turned off when the total of all enabled message counters becomes zero.

If notification is enabled for a particular message type then Meridian Mail Voice Messaging will announce to the user, at logon time, if there are any such external messages waiting for the user. If NOTIFY_CLEAR is specified, the corresponding message counter is set to zero. If NOTIFY_KEEP is specified the counter will not be modified.

The following functions are available:

Function	**Description**
m_GetMsgCounter	Get external message counter
m_GetMsgNotification	Get external message notification
m_SetMsgCounter	Set external message counter
m_SetMsgNotification	Set external message notification

In each of the functions, the following message types (MsgTypes) are valid:

Message type	**Description**
MT_FAX	FAX messages
MT_TEXT	Text messages

The valid Enabled settings are as follows:

Enable settings	**Description**
OFF	No notification
NOTIFY_CLEAR	Notify, but clear counter on log on to voice messaging
NOTIFY_KEEP	Notify, but keep counter on log on to voice messaging.

m_GetMsgCounter—Get external message counter

This function will return the current value of a specified user's external message counter for a given message type.

The function m_SetMsgCounter() can be used to change the value of this counter.

Header files to include	m_acc.h (Constants, return codes) m_msg.h (Function declarations, constants)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_BAD_BOX Invalid mailbox number MME_BAD_MSG_TYPE Invalid external message type MME_DO_LOGON Must be logged on to use this command
See also	m_SetMsgCounter m_GetMsgNotification m_SetMsgNotification m_GetCabinetInfo

Declaration

```
short m_GetMsgCounter(Mailbox, MsgType, Count, rc)
    char *Mailbox;        /* mailbox number of user */
    short MsgType;  /* message type to retrieve counter for */
    unsigned short *Count;  /* returned value of counter */
    short *rc;            /* returned status code */
```

MailBox This should be large enough to accept strings up to BOX_SIZE in length as defined by m_acc.h.

m_GetMsgNotification—Get message notification

This function will return the current notification setting (enabled or disabled) of a specified user's external messages of a given message type.

The function m_SetMsgNotification() can be used to enable or disable the notification of a user's external messages for a particular message type.

Header files to include	m_acc.h (Constants, return codes) m_msg.h (Function declarations, constants)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_BAD_BOX Invalid mailbox number MME_BAD_MSG_TYPE Invalid external message type MME_DO_LOGON Must be logged on to use this command
See also	m_SetMsgNotification m_GetMsgCounter m_SetMsgCounter

Declaration

```
short m_GetMsgNotification(Mailbox, MsgType, Enabled, rc)
    char *Mailbox;   /* mailbox number of user */
    short MsgType;   /* msg. type to check notification for */
    short *Enabled;  /* returned - notification setting?
                        OFF/NOTIFY_CLEAR/NOTIFY_KEEP */
    short *rc;       /* returned status code */
```

MailBox This should be large enough to accept strings up to BOX_SIZE in length as defined in m_acc.h.

m_SetMsgCounter—Set external message counter

This function will modify a specified user's external message counter for a given message type.

The function m_GetMsgCounter() can be used to query the current value of this counter.

Header files to include	m_acc.h (Constants, return codes) m_msg.h (Function declarations, constants)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_BAD_BOX Invalid mailbox number MME_DO_LOGON Must be logged on to use this command MME_BAD_MSG_TYPE Invalid external message type
See also	m_GetMsgCounter m_GetMsgNotification m_SetMsgNotification

Declaration

```
short m_SetMsgCounter(Mailbox, MsgType, Count, rc)
    char *Mailbox;      /* mailbox number of user */
    short MsgType;      /* msg. type of counter to modify */
    unsigned short Count; /*value to give to message counter*/
    short *rc;          /* returned status code */
```

MailBox This should be large enough to accept strings up to BOX_SIZE in length as defined in m_acc.h.

m_SetMsgNotification—Set message notification

This function will enable or disable a specified user's external message notification for a given message type.

The m_GetMsgNotification() function can be used to query the current notification setting.

Header files to include	m_acc.h (Constants, return codes) m_msg.h (Function declarations, constants)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_BAD_BOX Invalid mailbox number MME_BAD_MSG_TYPE Invalid external message type MME_BAD_FLAG Invalid flag
See also	m_GetMsgNotification m_GetMsgCounter m_SetMsgCounter

Declaration

```
short m_SetMsgNotification(Mailbox, MsgType, Enable, rc)
    char *Mailbox;  /* mailbox number of user */
    short MsgType;  /* message type to set notification for */
    short Enable;   /* notification settings OFF/NOTIFY_CLEAR/
                       NOTIFY_KEEP */
    short *rc;      /* returned status code */
```

MailBox This should be large enough to accept strings up to BOX_SIZE in length as defined in m_acc.h.

Chapter 10: Voice segment file functions

Voice segment files are Meridian Mail files which contain short sequences of voice (typically single words) in separate segments. These segments can be concatenated together to form a voice prompt—a phrase which has a smooth transition between words.

The functions described in this chapter provide for the creation, manipulation, and segment deletion of user voice segment files. Many of these functions are for use by the Voice Prompt Editor and may not be used by many application programs.

For more information on the Voice Prompt Editor, see the *Voice Prompt Editor User's Guide* (555-7001-318).

Function	**Description**
m_AddSeg	Add a voice segment
m_AddToSeg	Add silence to a voice segment
m_DeleteFromSeg	Delete voice/silence from a voice segment
m_DeleteSeg	Delete a voice segment
m_GetSegInfo	Get segment information
m_GetNumSegs	Get number of segments in file
m_GetSegScript	Get segment script
m_NormalizeSeg	Normalize a voice segment
m_PlaySegs	Play a list of segments
m_PositionToSeg	Move to a specified segment

m_SegPattern/M_RetrieveSeg	Retrieve segment information from files
m_SetSegInfo	Set segment information
m_SetSegScript	Set segment script
m_UndeleteSeg	Undelete a voice segment

Voice operations on the segments are performed by the m_RecordVoice(), m_StopVoice(), and m_PlaySegs() functions. In particular, the m_PlaySegs() function is used to concatenate voice segments and play them back without delays.

A new voice segment file is created by m_CreateFile(). An existing voice segment file can be opened and closed by the m_OpenFile() and m_CloseFile() functions. Other filing functions, such as m_SetFileSubject() and m_CopyFile(), work on these files as well.

m_AddSeg—Add a voice segment

Use the m_AddSeg() function to add a new voice segment to an open voice segment file.

Header files to include	m_acc.h (Constants, return codes) m_seg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_MAX_SEGS Reached maximum number of segments allowed in file MME_READ_SEG_FILE File must be opened in update mode MME_BAD_HANDLE Unassigned file handle MME_DO_LOGON Must be logged on to use this command MME_BAD_COMMAND Command invalid on this file type
See also	m_DeleteSeg m_SegPattern/m_RetrieveSeg

Declaration

```
short m_AddSeg(FileHandle, SegmentID, rc)
    short FileHandle;    /* file to add segment to */
    short *SegmentID;    /* returned segment ID number */
    short *rc;           /* returned status code */
```

FileHandle The file handle number obtained by m_CreateFile() or m_OpenFile().

SegmentID This is returned to indicate the segment number of the new voice segment. The current position in the voice segment file is updated to the beginning of the new voice segment.

m_AddToSeg—Add silence to a voice segment

This function adds a specified amount of silence to the beginning or end of a specified voice segment.

Header files to include	m_acc.h (Constants, return codes) m_seg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_BAD_SEG_ID Segment ID not found MME_BAD_HANDLE Unassigned file handle MME_READ_ONLY Cannot do command on read-only file MME_BAD_COMMAND Command invalid on this file type MME_BAD_EDIT_POS Invalid segment editing position MME_BAD_OPERATOR Invalid segment editing operator MME_BAD_AMOUNT Invalid amount specified
See also	m_DeleteFromSeg m_NormalizeSeg

Declaration

```
short m_AddToSeg(FileHandle, SegmentID, Position, Amount, rc)
    short FileHandle; /* file containing seg  */
    short SegmentID;  /* seg to add silence to (0=current) */
    short Position;   /* position in segment to add silence */
    long Amount;      /* amount of silence to add (milli
                         secs)*/
    short *rc;        /* returned status code */
```

FileHandle The file handle number obtained by m_CreateFile() or m_OpenFile().

Position The possible position values are:

Position	**Description**
SP_BEGIN	Beginning of segment
SP_END	End of segment

m_DeleteFromSeg—Delete voice/silence from a segment

This function deletes a specified amount of voice and/or silence from the beginning or end of a specified voice segment.

Header files to include	m_acc.h (Constants, return codes) m_seg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_BAD_SEG Segment ID not found in file MME_BAD_HANDLE Unassigned file handle MME_READ_ONLY Cannot do command on read-only file MME_BAD_COMMAND Command invalid on this file type MME_BAD_EDIT_POS Invalid segment editing position MME_BAD_OPERATOR Invalid segment editing operator MME_BAD_AMOUNT Invalid amount specified
See also	m_AddToSeg m_NormalizeSeg

Declaration

```
short m_DeleteFromSeg(FileHandle, SegmentID, Position, Amount, rc)
    short FileHandle; /* file containing seg to delete from */
    short SegmentID;  /* segment to delete from (0=current) */
    short Position;   /* position to delete from  */
    long Amount;      /* amount to delete (millisecs) */
    short *rc;        /* returned status code */
```

FileHandle The file handle number obtained by m_CreateFile() or m_OpenFile().

Position The possible position values are:

Position	Description
SP_BEGIN	Beginning of segment
SP_END	End of segment

Amount This can be either the number of milliseconds to delete from either end of the segment, or “DS_ALL_SILENCE”. DS_ALL_SILENCE may be used to delete all leading or trailing silence from a voice segment.

m_DeleteSeg—Delete a given voice segment

This function marks a voice segment for deletion. All segment operations (such as m_PlaySegs(), etc.) are allowed on voice segments which have been marked for deletion. When the file is committed, the deletion becomes effective.

If segment deletions have been made, then the segment IDs of the voice segments contained in that file will change when the file is committed. The m_SegPattern() and m_RetrieveSeg() functions must be used to re-retrieve the voice segments to determine the new segment IDs.

Header files to include	m_acc.h (Constants, return codes) m_seg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_BAD_SEG_ID Segment ID not found in file MME_READ_SEG_FILE File must be opened in update mode MME_BAD_HANDLE Unassigned file handle MME_READ_ONLY Cannot do command on read-only file MME_BAD_COMMAND Command invalid on this file type
See also	m_UndeleteSeg m_AddSeg m_SegPattern/m_RetrieveSeg m_PositionToSeg

–continued–

Declaration

```
short m_DeleteSeg(FileHandle, SegmentID, rc)
     short FileHandle;  /* file to delete segment from */
     short SegmentID;   /* segment to delete (0=current) */
     short *rc;         /* returned status code */
```

FileHandle The file handle number obtained by m_CreateFile() or m_OpenFile().

m_GetNumSegs—Get the total number of segments in a file

The total number of voice segments in a voice segment file may be determined by calling this function.

Header files to include	m_acc.h (Constants, return codes) m_seg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_BAD_HANDLE Unassigned file handle MME_BAD_COMMAND Command invalid on this file type
See also	m_GetFileInfo

Declaration

```
short m_GetNumSegs(FileHandle, NumSegments, rc)
    short FileHandle;    /* voice segment file to query */
    short *NumSegments;  /* returned no. of segs in file */
    short *rc;           /* returned status code */
```

FileHandle The file handle number obtained by m_CreateFile() or m_OpenFile().

m_GetSegInfo—Retrieve voice segment information

This function retrieves status information on a single voice segment from an open voice segment file, including the voice segment name, title, length (in milliseconds) and whether or not the segment has been marked deleted.

To retrieve the associated text script for the voice segment, the function m_GetSegScript() should be used.

Header files to include	m_acc.h (Constants, return codes) m_seg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_BAD_SEG_ID Segment ID not found in file MME_BAD_HANDLE Unassigned file handle MME_BAD_COMMAND Command invalid on this file type
See also	m_GetSetInfo m_GetSegScript m_PositionToSeg

Declaration

```
short m_GetSegInfo(FileHandle, SegmentID, Name, Title, Length, Deleted, rc)
    short FileHandle;   /* file containing seg to retrieve */
    short SegmentID;    /* segment to retrieve (0=current) */
    char *Name;         /* returned name of the segment */
    char *Title;        /* returned title of the segment */
    long *Length;       /* returned voice length (millisecs)*/
    short *Deleted;     /* returned marked deleted status
                        (TRUE/FALSE) */
    short *rc;          /* returned status code */
```

Name This should be large enough to accept strings of up to SEGNAME_SIZE in length.

Title This should be large enough to accept strings of up to SEGTITLE_SIZE in length.

FileHandle The file handle number obtained by m_CreateFile() or m_OpenFile().

m_GetSegScript—Retrieve text script of a segment

This function retrieves the script associated with a voice segment. The script may be used to store, in text format, the words which have been recorded in the voice segment.

To retrieve the name, title, and length (in milliseconds) of the voice segment, use the function m_GetSegInfo().

Header files to include	m_acc.h (Constants, return codes) m_seg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_BAD_SEG_ID Segment ID not found in file MME_BAD_HANDLE Unassigned file handle MME_BAD_COMMAND Command invalid on this file type
See also	m_GetSegInfo m_GetSegScript m_PositionToSeg

Declaration

```
short m_GetSegScript(FileHandle, SegmentID, Script, rc)
    short FileHandle;  /* file containing seg to retrieve */
    short SegmentID;   /* segment to retrieve (0=current) */
    char *Script;      /* returned script for the segment */
    short *rc;         /* returned status code */
```

FileHandle The file handle number obtained by m_CreateFile() or m_OpenFile().

Script This should be large enough to accept a string up to SEGSCRIPT_SIZE in length.

m_NormalizeSeg—Normalize a voice segment

This function normalizes a given voice segment by removing *all* silence from the beginning and end of the voice segment, then adding a specified amount of silence back to the beginning and end of the voice segment.

By performing this function on all related voice segments in a voice segment file, the segments will be "normalized"—a fixed period of silence will be noticed between segments during playback. In particular, where no silence is desired between segments, this function can be used with 0 (zero) specified as both the BegSilence and EndSilence amounts.

The Meridian Mail system must have silence compression enabled for this function to work properly. See *Meridian Mail System Administration Tools* (NTP 555-7001-305) for more information.

Header files to include	m_acc.h (Constants, return codes) m_seg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_BAD_SEG_ID Segment ID not found in file MME_BAD_HANDLE Unassigned file handle MME_READ_ONLY Cannot do command on read-only file MME_BAD_COMMAND Command invalid on this file type
See also	m_AddToSeg m_DeleteFromSeg

–continued–

Declaration

```
short m_NormalizeSeg(FileHandle, SegmentID, BegSilence,
EndSilence, rc)
    short FileHandle; /* file containing seg to normalize */
    short SegmentID;  /* segment to normalize (0=current) */
    long BegSilence;  /* amount of silence to leave at
                         beginning (milliseconds) */
    long EndSilence;  /* amt. to leave at end (millisec) */
    short *rc;        /* returned status code */
```

FileHandle The file handle number obtained by m_CreateFile() or m_OpenFile().

m_PlaySegs—Play list of voice segments

This function takes a list of segment IDs from a single open voice segment file and plays them in sequence. The segments specified are concatenated together in the specified order and played back without delays.

More than one m_PlaySegs() function can be called to queue lists of voice segment IDs to be played. This prevents delays between the playback of voice segments by enqueueing lists of voice segment IDs, possibly from different voice segment files, into a “play queue” while previously queued voice segments are being played.

A maximum of two voice segment files may be open at the same time while being used for playing voice segments. The voice segment files to be used for playback should be opened before calling this function, and should remain open while playing is in progress. Opening or closing an active voice segment file while segments are playing, results in the playback being stopped.

If the play queue fills up as a result of this request, then an m_OnError() asynchronous error event is generated with error ER_QUEUE_FULL and the extra segments are not played.

Header files to include	m_acc.h (Constants, return codes) m_seg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open Connected
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_MAX_SEG_FILES Too many open voice segment files for play MME_NO_ACTV_CHNL No active voice channel

–continued–

Return codes (continued)	MME_BAD_SEQUENCE Previous play still in progress MMS_NO_VOICE No voice in segment to play MME_BAD_NUM_SEGS Bad number of segments specified MME_SEG_Q_FULL Segment play queue is full MME_CHAN_IN_USE Voice channel already in use MME_DETECT_INPROG Sound detect already in progress MME_BAD_HANDLE Unassigned file handle MME_BAD_COMMAND Command invalid on this file type MME_BAD_SEG_ID Segment ID not found in file
See also	m_StopVoice m_PlayVoice m_PlayMsg
Events	m_OnPlayEnd m_OnError

Declaration

```
short m_PlaySegs(FileHandle, Segments, ReturnPlayEnd, rc)
    short FileHandle;     /* file containing segs to play */
    short Segments[];     /* segment IDs to play (in order) */
    short ReturnPlayEnd;  /* return end of playback event?
                             (TRUE/FALSE) */
    short *rc;            /* returned status code */
```

FileHandle The file handle number obtained by m_CreateFile() or m_OpenFile().

Segments This is an array of integer voice segment IDs (maximum of SEGLIST_SIZE) terminated with a segment ID of zero (0) to indicate the end of the list.

ReturnPlayEnd This indicates whether an m_OnPlayEnd() event should be sent back to the application after the last voice segment in the play queue has completed playing. If TRUE, then no more m_PlaySegs() calls can be made until the m_OnPlayEnd() event has been received (that is, the queue is empty). The m_StopVoice() function can be used to end the playback prematurely, in which case the m_OnPlayEnd() event will not be generated.

m_PositionToSeg—Position to given voice segment

This function changes the current position in the specified open voice segment file to a given voice segment.

This function will fail if a segment ID is specified which does not exist in the given file. To position to the first voice segment in the file, specify a segment ID of 1.

Header files to include	m_acc.h (Constants, return codes) m_seg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_BAD_SEG_ID Segment ID not found in file MME_BAD_HANDLE Unassigned file handle MME_BAD_COMMAND Command invalid on this file type

Declaration

```
short m_PositionToSeg(FileHandle, SegmentID, rc)
    short FileHandle;                       /* file to position within */
    short SegmentID;                        /* segment to position to */
    short *rc;                              /* returned status code */
```

FileHandle The file handle number obtained by m_CreateFile() or m_OpenFile().

m_SegPattern—Retrieve voice segments

The voice segments in an open voice segment file can be retrieved using the following functions. m_SegPattern() is used to initialize the retrieval, after which the m_RetrieveSeg() function can be called repeatedly to retrieve successive voice segments.

Header files to include	m_acc.h (Constants, return codes) m_seg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_BAD_SEG_ID Segment ID not found in file MME_BAD_HANDLE Unassigned file handle MME_BAD_COMMAND Command invalid on this file type
See also	m_GetSegInfo m_GetSegScript m_AddSeg m_PositionToSeg m_RetrieveSeg

Declaration

```
short m_SegPattern(FileHandle, StartSegment, rc)
    short FileHandle;   /* file containing segs to retrieve */
    short StartSegment; /* segment to start retrieval from
                           (0=current) */
    short *rc;          /* returned status code */
```

StartSegment This is used to specify where in the voice segment file the retrieval should commence. Specify a value of 1 to start retrieval from the beginning of the file, zero (0) to start from the current position, or any other segment ID value to start retrieval from a specific position.

m_RetrieveSeg—Retrieve voice segments

m_RetrieveSeg() returns TRUE on a successful retrieval and FALSE on an error or at end-of-retrieval—a return value of FALSE with an “rc” status code of MMS_OKAY indicates an end-of-retrieval. After the retrieval has been completed, the current position in the voice segment file is changed to the segment ID specified in StartSegment.

Header files to include	m_acc.h (Constants, return codes) m_seg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_DO_SEGPAT Must call m_SegPattern() first MME_BAD_HANDLE Unassigned file handle
See also	m_GetSegInfo m_GetSegScript m_AddSeg m_PositionToSeg m_SegPattern

Declaration

```
short m_RetrieveSeg(FileHandle, SegmentID, Name, Title, Length, Deleted, rc)
    short FileHandle; /* file containing segs to retrieve */
    short *SegmentID; /* returned segment ID number */
    char *Name;       /* returned name of the segment */
    char *Title;      /* returned title of the segment */
    long *Length;     /* returned voice length in millisec. */
    short *Deleted;   /* returned marked deleted status
                         (TRUE/FALSE)*/
    short *rc;        /* returned status code */
```

FileHandle The file handle number obtained by m_CreateFile() or m_OpenFile().

Name This should be large enough to accept strings at least SEGNAME_SIZE in length.

Title This should be large enough to accept strings at least SEGTITLE_SIZE in length.

m_SetSegInfo—Update name and title of voice segment

Each voice segment in a voice segment file may have a name and a title attached to it. These fields can be added, modified, or deleted by using this function.

Header files to include	m_acc.h (Constants, return codes) m_seg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_BAD_SEG_ID Segment ID not found in file MME_BAD_HANDLE Unassigned file handle MME_READ_ONLY Cannot do command on read-only file MME_BAD_COMMAND Command invalid on this file type MME_TITLE_LENGTH Invalid length in field
See also	m_GetSegInfo m_SetSegScript m_PositionToSeg

Declaration

```
short m_SetSegInfo(FileHandle, SegmentID, Name, Title, rc)
    short FileHandle;    /* file containing seg to update */
    short SegmentID;     /* segment to update (0=current) */
    char *Name;          /* name of the segment */
    char *Title;         /* title of the segment */
    short *rc;           /* returned status code */
```

FileHandle The file handle number obtained by m_CreateFile() or m_OpenFile().

Name The maximum size is SEGNAME_SIZE, as defined in the header files.

Title The maximum size is SEGTITLE_SIZE, as defined in the header files.

m_SetSegScript—Update text script of a segment

Each voice segment in a voice segment file may have a text script attached to it. This concept is useful for identifying the contents of a voice segment in text form. The script can be added, modified or deleted using this function.

Header files to include	m_acc.h (Constants, return codes) m_seg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_BAD_SEG_ID Segment ID not found in file MME_MAX_SCRIPT_SIZE Script for voice segment too long MME_BAD_HANDLE Unassigned file handle MME_READ_ONLY Cannot do command on read-only file MME_BAD_COMMAND Command invalid on this file type MME_SCRIPT_LENGTH Invalid script length
See also	m_GetSegScript m_SetSegInfo m_PositionToSeg

Declaration

```
short m_SetSegScript(FileHandle, SegmentID, Script, rc)
    short FileHandle;   /* file containing seg to update */
    short SegmentID;    /* segment to update (0=current) */
    char *Script;       /* script for the segment */
    short *rc;          /* returned status code */
```

FileHandle The file handle number obtained by m_CreateFile() or m_OpenFile().

Script The maximum size is SEGSCRIPT_SIZE, as defined in the header files.

m_UndeleteSeg—Recover a deleted voice segment

This function recovers a voice segment which has been marked for deletion from an open voice segment file.

If the voice segment file has been committed since the segment deletion, then the voice segment cannot be recovered.

Header files to include	m_acc.h (Constants, return codes) m_seg.h (Function declarations)
Prerequisites	Registered Acquired Logged on File open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_BAD_SEG_ID Segment ID not found in file MME_BAD_HANDLE Unassigned file handle MME_READ_ONLY Cannot do command on read-only file MME_BAD_COMMAND Command invalid on this file type
See also	m_DeleteSeg m_PositionToSeg

Declaration

```
short m_UndeleteSeg(FileHandle, SegmentID, rc)
    short FileHandle;   /* file to undelete segment from */
    short SegmentID;    /* segment to undelete (0=current) */
    short *rc;          /* returned status code */
```

FileHandle The file handle number obtained by m_CreateFile() or m_OpenFile().

Chapter 11: External Notification Services

This chapter describes the additions to the Meridian Access API Library (known as ‘m_acc.lib’) which will support External Notification Services (ENS). These additions are intended to accommodate those applications that perform service or administrative related tasks on the Meridian Mail system. Unlike your typical “IVR” system, these applications do not require a voice channel to perform their activities. As such, the functions described here are unlike other ACCESS APIs in that the application MUST NOT have acquired a voice channel before invoking them.

ENS is an additional service contained in the library. ENS will allow developers the ability to build applications that integrate voice mail and e-mail systems. The concept behind ENS is to permit an application establishing itself as the External Notification Client to track the status of designated mailboxes on a particular Meridian Mail system. There can be only one ENS application per Meridian Mail system at any given time. Any ACCESS application can mark a user’s mailbox for external notification. An event is then passed to the application every time a message is deposited in a user’s mailbox. In addition, at the end of every user session, an event will be passed to the application indicating the current view of the mailbox contents.

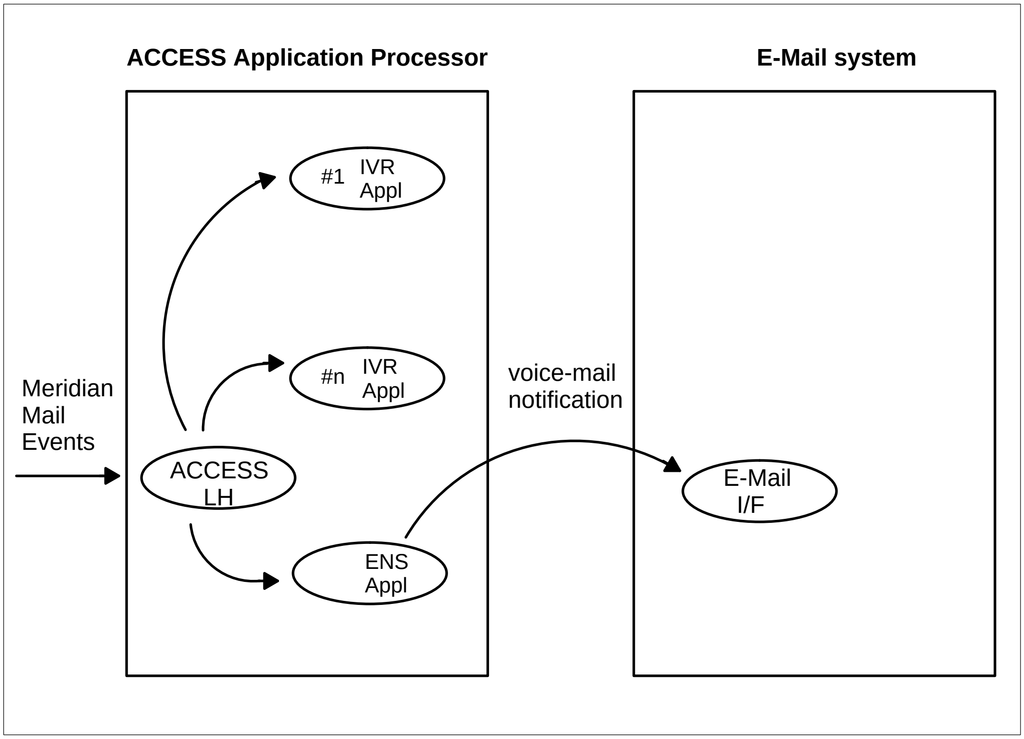
The ENS ‘C’ library functions include the following:

- **m_AcquireENS**   Acquire External Message Notification Service on the Meridian Mail system and establish the calling process as the ENS application.
- **m_ReleaseENS**   Release External Message Notification Service on the Meridian Mail system, and relinquish the calling process as being the ENS application.

- **m_OnMBox Status** This is an event indicating that the status of a mailbox being monitored by ENS may have changed.
- **m_SetMboxEHN** Set a Meridian Mail User Mailbox's external system notification to be either ON or OFF.
- **m_GetMboxStat** Get a Meridian Mail User Mailbox's current status.

Figure 11-1 illustrates how an ENS application could be used to notify an e-mail system of outstanding voice messages. Note the existence of a voice application (#1 IVR Appl...#nIVR Appl) running concurrently with the ENS application. An ENS application does not require the use of a voice channel and, thus, will not affect an existing voice application. The only requirement for the application is that it establish itself as the designated External Notification Client on Meridian Mail (this is accomplished through the m_AcquireENS API).

Figure 11-1
Notification to E-Mail system of outstanding voice messages



It is very important that the ACCESS ENS application remain in an active state at all times. A temporary outage between the application and Meridian Mail or the application and the e-mail system may result in the status of mailboxes being out of sync. To curtail this problem, Meridian Mail will send a “Time Out” event if it has not heard from the ENS application after 30 seconds. The ENS application must in turn respond to the TimeOut event within 30 seconds. Otherwise, Meridian Mail will conclude it is not operational (that is, ENS application does not appear to be active) and will send a “Session End” event. The ENS application must then reestablish itself as a server to receive further ENS events.

The following restrictions apply to ACCESS applications using ENS executing on the ACCESS Applications Processor:

- There can only be one designated External Notification Client application for every Meridian Mail system at any given time.
- Once the ENS application has established itself as the External Notification Client on Meridian Mail, it cannot also acquire a voice channel. That is to say, it is restricted to using the API's defined here.
- Other applications can retrieve a user mailbox status or enable a user mailbox's external notification as long as they have not previously acquired a voice channel.

ACCESS will enforce the above restrictions.

m_AcquireENS—Acquire External Notification Service on Meridian Mail system

In order to become an ENS application, a process must invoke the m_AcquireENS function. The application can then expect to receive "MailBox Status" events provided it has installed the appropriate event handler (see m_OnMBox Status). These events are specifically defined to keep the application informed of the designated mailboxes it is monitoring. A given user mailbox on Meridian Mail can be designated for monitoring by enabling its Host Notification flag (see m_SetMboxEHN). The service can be aborted by calling the m_ReleaseENS API.

Once a process has established itself as the ENS application it should regularly inform the Meridian Mail system that it is "alive." The application can have this done automatically by ACCESS by simply invoking m_TimeoutOff function. Alternatively, the application could have the Meridian Mail system send it TimeOut events every 30 seconds. In this scenario, the application would respond appropriately to the TimeOut event with an m_TimeoutContinue. In either case, if the Meridian Mail system concludes that the ENS application is not active (that is, it has not heard from the application in 60 seconds), a SessionEnd event is sent and the process has to reestablish itself as the ENS application.

The application MUST not have acquired a voice channel before requesting this service. In addition, only one application can request the External Message Notification service on Meridian Mail at any given time. This also implicitly restricts the application from logging into a Meridian Mail account, playing/recording of voice files, or performing any telephony-based API. The only prerequisite for using this administrative API is that the application must have been previously registered.

Header files to include	m_acc.h	(Constants, return codes)
	m_ens.h	(Function declaration, constants)
Prerequisites	Registered	
	Not Acquired	
	No other ENS application currently executing	

–continued–

Return Codes	MME_NOT_REGISTERED Application has not registered MME_ALREADY_ACQUIRED Application has acquired a voice channel MME_ENS_EXISTS An Application has already acquired ENS
Events	MailBox Status Timeout

Declaration

```
short m_AcquireENS (rc)
    short*rc;                                   /*status return code      */
```

m_ReleaseENS—Release External Message Notification Service on Meridian Mail

An ENS application can stop receiving “Mailbox Status” events and release the service on the Meridian Mail system by invoking the m_ReleaseENS function. The designated ENS application is the only process that has the authority to discontinue the External Message Notification on the Meridian Mail system. There is a subtle difference between issuing the release function and allowing Meridian Mail to conclude that the ENS application is no longer active (that is, not responding to a Timeout event). Not responding to a Timeout event results in a SEER message indicating that the ENS application is not operational. Once External Message Notification has been released on Meridian Mail, the application is free to acquire a voice channel, log into a Meridian Mail account, play/record voice files, or perform any telephony-based API. The only prerequisite for using this administrative API is that the application must have been previously registered and is currently the ENS application.

Header files to include	m_acc.h (Constants, return codes) m_ens.h (Function declarations, constants)
Prerequisites	Registered ENS application currently executing
Return Codes	MME_NOT_REGISTERED Application has not registered MME_NOT_ENS Must be ENS Application to invoke this command

Declaration

```
short m_ReleaseENS (rc)
   short*rc;                              /*status return code     */
```

m_SetMboxEHN—Set Mailbox’s External Host Notification

A given user mailbox on Meridian Mail can be designated for monitoring by ENS simply by enabling the mailbox’s External Host Notification (EHN) flag. The m_SetMboxEHN command sets a user mailbox’s EHN flag to either ON or OFF. Once a user process has set the EHN flag to yes, ENS sends a ‘‘Mailbox Status” event to the ENS application if a new message is put in the mailbox. As well, at the end of every user session, an event will pass to the server indicating the current view of the mailbox contents.

The application MUST not have acquired a voice channel before requesting this service. The command also implicitly restricts the application from logging into a Meridian Mail account, playing/recording voice files, or performing any telephony-based API. The only prerequisite for using this administrative API is that the application must have been previously registered.

Header files to include	m_acc.h (Constants, return codes) m_ens.h (Function declarations, constants)
Prerequisites	Registered Not Acquired
Return Codes	MME_NOT_REGISTERED Application has not registered MME_ALREADY_ACQUIRED Application has acquired a voice channel MME_BAD_BOX Invalid mailbox on Meridian Mail MME_INVALID_CUST Invalid Customer ID number on Meridian Mail

Declaration

```
short m_SetMboxEHN(CustNum,MailBox,EHN,rc)
    short CustNum;  /*Customer ID number where mailbox resides
                        on Meridian Mail*/
    char MailBox[BOX_SIZE];             /*User Mailbox DN-may include
                        network address*/
    short EHN;          /*External Host Notification flag-either
                        ON or OFF*/
    short*rc            /*status return code  */
```

CustNum This parameter specifies the logical Customer ID number on the Meridian Mail system where the user mailbox resides. It is a short integer which is always greater than or equal to 0, or an application which can use the definition for the default Customer ID which is -1.

Mailbox This parameter specifies the User Mailbox DN on the Meridian Mail system which will have its EHN flag changed. The User Mailbox is a numeric string that may also include a networking and/or NMS prefix and can be no longer than 18 digits in length (excluding NULL).

EHN This parameter is used to specify the state of the User Mailbox's External Host Notification flag. A Mailbox can either turn ON or turn OFF the External Host Notification.

> ***Note:*** This application can use the definitions ON or OFF for this purpose.

m_GetMboxStat—Get a Mailbox’s current status

This command allows any application to probe a given user mailbox on Meridian Mail for a current view of its voice messaging contents. The information returned from this command is equivalent to what is returned from a “Mailbox Status” event. The differences are a) this command is solicited rather than coming in asynchronously like an event, and b) only an ENS application can receive “Mailbox Update” events whereas any application can invoke this command. In order to invoke the command, the application simply specifies the designated Customer ID and Mailbox in the “MboxStat” structure.

The application MUST not have acquired a voice channel before requesting this service. This also implicitly restricts the application from logging into a Meridian Mail account, playing/recording voice files, or performing any telephony-based API. The only prerequisite for using this administrative API is that the application must have been previously registered.

Header files to include	m_acc.h (Constants, return codes) m_ens.h (Function declarations, constants)
Prerequisites	Registered Not Acquired
Return Codes	MME_NOT_REGISTERED Application has not registered MME_ALREADY_ACQUIRED Application has acquired a voice channel MME_INVALID_CUST Invalid Customer ID number on Meridian Mail MME_BAD_BOX Invalid mailbox on Meridian Mail
–continued–	

Declaration

```
short m_GetMboxStat (MboxData,rc)
    struct MboxStat*MboxData;    /*User Mailbox status update
                                          info structure */
    short*rc;                             /*status return code */

struct MboxStat{
    short CustNum;  /*Meridian Mail Customer ID number where
                             user mailbox resides*/
    char Mailbox[BOX_SIZE];          /*Mailbox number which was
                    updated <=18 chars */
    short NumVM;         /*Current number of unread Voice Messages*/
    short NumUVM;        /*Current number of unread Urgent Voice
                         Messages */
    short NumText;       /*Current number of unread Text Messages */
    short NumFax;        /*Current number of unread Fax Messages */
  short MWI;             /*Message Waiting indicator status-either
                         *ON or OFF*/
    short  EHN;          /*External Host Notification flag-either
                         /*ON or OFF*/
    short EText;         /*External Text-OFF,NOTIFY_CLEAR,
                         NOTIFY_KEEP*/
    short EFax;          /*OFF,NOTIFY_CLEAR,NOTIFY_KEEP*/
}
```

CustNum This parameter is used to specify the logical Customer ID number on the Meridian Mail system where the user mailbox resides. It is a short integer which is always greater than or equal to 0, or an application can use the definition for the default Customer ID which is -1.

> ***Note:*** This is an input parameter and must be specified before invoking m_GetMboxStat API.

Mailbox This parameter is used to specify the User Mailbox number on the Meridian Mail system. The User Mailbox is a numeric string that also may include a networking and/or NMS prefix and can be no longer than 18 digits in length (excluding NULL).

> ***Note:*** This is an input parameter and must be specified before invoking m_GetMboxStat API.

NumVM This field contains the current number of unread voice messages in the User Mailbox. It is a short integer which is always greater than or equal to 0.

NumUVM This field contains the current number of unread URGENT voice messages in the User Mailbox. It is a short integer which is always greater than or equal to 0.

NumText This field contains the current number of unread TEXT messages in the User Mailbox. It is a short integer which is always greater than or equal to 0.

NumFax This field contains the current number of unread FAX messages in the User Mailbox. It is a short integer which is always greater than or equal to 0.

MWI This field contains the current status of the User Mailbox “Message Waiting Indicator” (MWI). The MWI can either be ON or OFF.

EHN This field contains the state of the User Mailbox’s External Host Notification flag. The EHN flag can be turned on or off using the m_SetMboxEHN function.

EText This field contains the state of the User Mailbox’s External Text Notification flag. A Mailbox can have it’s External Text flag set to OFF, NOTIFY_CLEAR (for enable and clear the count), or NOTIFY_KEEP (for enable and retain present count).

EFax This field contains the state of the User Mailbox’s External Fax Notification flag. A Mailbox can have its External Text flag set to OFF, NOTIFY_CLEAR (for enable and clear the count), or NOTIFY_KEEP (for enable and retain present count).

Chapter 12: User administration functions

User Administration allows various features of the Meridian Mail system to be enabled, disabled, or modified. These functions affect features which are maintained across sessions rather than features of short duration.

Personal Distribution Lists function	**Description**
m_AddBoxToPDL	Add entries to list.
m_AddNameToPDL	Add entries to list.
m_DeleteFromPDL	Delete entries from list.
m_DeletePDL	Delete list.
m_OpenPDL	Open list file.
m_PDLPattern	Retrieve list entries.
m_RetrievePDL	Retrieve list entries.
Greetings function	**Description**
m_DeleteGreeting	Delete greeting.
m_OpenGreeting	Open greeting file.
Personal Verification function	**Description**
m_DeletePersVerif	Delete personal verification.
m_OpenPersVerif	Open personal verification f.ile.
Other function	**Description**
m_UserPassword	Change password.

Personal Distribution Lists

A Personal Distribution List (PDL) is a private list of mailbox numbers stored together under a single-digit designation. A message can be sent to all mailboxes in the list by addressing the message to all entries in the list. The list can be maintained and used by either a Meridian ACCESS application (using the following functions) or by using Meridian Mail Voice Messaging from a telephone set.

Personal Distribution Lists function	**Description**
m_AddBoxToPDL	Add entries to list.
m_AddNameToPDL	Add entries to list.
m_DeleteFromPDL	Delete entries from list.
m_DeletePDL	Delete list.
m_OpenPDL	Open list file.
m_PDLPattern	Retrieve list entries.
m_RetrievePDL	Retrieve list entries.

m_AddBoxToPDL—Add entry to PDL

A mailbox number can be added to a PDL using this function.

Header files to include	m_acc.h (Constants, return codes) m_admin.h (Function declarations)
Prerequisites	Registered Acquired Logged on PDL open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_BAD_RCVR The given Mailbox was not found MME_MULTIMATCH Box/Name matches several users MME_BAD_HANDLE Unassigned file handle MME_BAD_COMMAND Command invalid on this file type MME_MAX_PDL_ENTRIES Exceeded maximum entries MME_BAD_BOX Invalid box number MME_NOT_NUMERIC Non-numeric in numeric field
See also	m_openPDL m_DeleteFromPDL

Declaration

```
short m_AddBoxToPDL(FileHandle, MailBox, FullName, FullBox, rc)
    short FileHandle;                    /* handle of list to add to */
    char *MailBox;                       /* mailbox to add */
    char *FullName;                      /* returned full user name */
    char *FullBox;                       /* returned full box number */
    short *rc;                           /* returned status code */
```

FileHandle The file handle number obtained by m_OpenPDL().

Mailbox If it matches more than one user, the function will fail with a returned status code of MME_MULTIMATCH. It should be large enough to accept strings up to BOX_SIZE in length.

FullName This shoukd be large enough to accept strings up to FULLNAME_SIZE in length.

FullBox This should be large enough to accept strings up to BOX_SIZE in length.

m_AddNameToPDL—Add entry to PDL

A name can be added to a PDL using this function.

Header files to include	m_acc.h (Constants, return codes) m_admin.h (Function declarations)
Prerequisites	Registered Acquired Logged on PDL open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_BAD_RCVR The given Mailbox was not found MME_MULTIMATCH Box/Name matches several users MME_BAD_HANDLE Unassigned file handle MME_BAD_COMMAND Command invalid on this file type MME_MAX_PDL_ENTRIES Exceeded maximum entries MME_BAD_SURNAME Invalid lastname MME_BAD_GIVEN Invalid firstname
See also	m_openPDL m_DeleteFromPDL

Declaration

```
short m_AddNameToPDL(FileHandle, Name, FullName, FullBox,rc)
    short FileHandle;                /* handle of list to add to */
    char *Name;                      /* user name to add */
    char *FullName;                  /* returned full user name */
    char *FullBox;                   /* returned full box number */
    short *rc;                       /* returned status code */
```

FileHandle The file handle number obtained by m_OpenPDL().

Name This can be specified in a variety of formats and may contain wildcard characters. See Chapter 2 “Meridian Mail Facilities,” for details. If the Name matches more than one user, the function will fail with a returned status code of MME_MULTIMATCH. It should be large enough to accept strings up to FULLNAME_SIZE in length.

FullName This should be large enough to accept strings up to FULLNAME_SIZE in length.

FullBox This should be large enough to accept strings up to BOX_SIZE in length.

m_DeleteFromPDL—Delete entry from PDL

This function deletes an entry from a personal distribution list. Entries may only be deleted from an open PDL, and only by mailbox number—not by name.

If the MailBox occurs more than once in the PDL, this function deletes only the first occurrence.

Header files to include	m_acc.h (Constants, return codes) m_admin.h (Function declarations)
Prerequisites	Registered Acquired Logged on PDL open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_BAD_RCVR The given Mailbox was not found MME_BAD_HANDLE Unassigned file handle MME_BAD_COMMAND Command invalid on this file type MME_BAD_BOX Invalid box number MME_NOT_NUMERIC Non-numeric in numeric field MME_NO_MATCHING_BOX No matching box address in PDL
See also	m_AddBoxToPDL/m_AddNameToPDL

Declaration

```
short m_DeleteFromPDL(FileHandle, MailBox, rc)
short FileHandle;  /* handle of list to delete from */
char *MailBox;     /* box number to remove */
short *rc;         /* returned status code */
```

FileHandle The file handle number obtained by m_OpenPDL().

MailBox The maximum size is BOX_SIZE, as defined in the header files.

m_DeletePDL—Delete personal distribution list

This function deletes a personal distribution list. The list number to be deleted must be specified.

If the PDL is currently open, an attempt to delete it will fail.

Header files to include	m_acc.h (Constants, return codes) m_admin.h (Function declarations)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_FILE_OPEN File is open MME_BAD_PDL_NUM Invalid Personal Distribution List number MME_PDL_DNE List does not currently exist MME_BAD_HANDLE Unassigned file handle MME_MAX_OPEN Maximum open file limit reached MME_BAD_COMMAND Command invalid on this file type
See also	m_CloseFile

Declaration

```
short m_DeletePDL(ListNum, rc)
short ListNum;                              /* list to delete (1..9) */
short *rc;  /* returned status code */
```

ListNum The list number as specified in m_OpenPDL().

m_OpenPDL—Open a personal distribution list

A personal distribution list is a file which must be opened in order to be able to access the entries in the list, and closed (with commit) to update the list.

Other file operations (such as m_PlayVoice()) cannot be performed on a PDL file. The m_OpenPDL() function creates a Personal Distribution List with the given ListNum if one does not exist.

> ***Note:*** If the PDL file is closed without any names or box numbers in the file, the file is deleted.

Header files to include	m_acc.h (Constants, return codes) m_admin.h (Function declarations)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_BAD_PDL_NUM Invalid PDL number MME_BAD_HANDLE Unassigned file handle MME_FILE_OPEN File is already opened MME_MAX_OPEN Maximum open file limit reached MME_BAD_COMMAND Command invalid on this file type
See also	m_PDLPattern/m_RetrievePDL m_CloseFile m_CommitFile

Declaration

```
short m_OpenPDL(ListNum, FileHandle, rc)
short ListNum;        /* list to open (1..9) */
short *FileHandle;    /* returned handle to the PDL */
short *rc;            /* returned status code */
```

m_PDLPattern—Retrieve PDL entries

A personal distribution list can be retrieved using m_PDLPattern and m_RetrievePDL.

Header files to include	m_acc.h m_admin.h	(Constants, return codes) (Function declarations)
Prerequisites	Registered Acquired Logged on PDL open	
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_BAD_HANDLE Unassigned file handle MME_BAD_COMMAND Command invalid on this file type	
See also	m_AddBoxToPDL m_AddNameToPDL m_RetrievePDL	

Declaration

```
short m_PDLPattern(FileHandle, rc)
    short FileHandle;  /* handle of list to retrieve */
    short *rc;         /* returned status code */
```

FileHandle The file handle number obtained by m_OpenPDL().

m_RetrievePDL—Retrieve PDL entries

To start the retrieval of a Personal Distribution List, m_PDLPattern() should be called. Then, to retrieve all of the entries, m_RetrievePDL() should be called repeatedly. Each invocation retrieves one PDL entry.

This function returns TRUE on a successful retrieval and FALSE on an error or at end-of-list. The status code "rc" distinguishes an error from a normal end-of-list (MMS_OKAY indicates end-of-list).

Header files to include	m_acc.h (Constants, return codes) m_admin.h (Function declarations)
Prerequisites	Registered Acquired Logged on PDL open
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_DO_PDLPAT Must call m_PDLPattern() first MME_BAD_HANDLE Unassigned file handle MME_BAD_COMMAND Command invalid on this file type
See also	m_AddBoxToPDL m_AddNameToPDL m_PDLPattern

Declaration

```
short m_RetrievePDL(FileHandle, FullName, FullBox, rc)
    short FileHandle;  /* handle of list to retrieve */
    char *FullName;    /* returned full user name */
    char *FullBox;     /* returned box number */
    short *rc;         /* returned status code */
```

FileHandle The file handle number obtained by m_OpenPDL().

FullName This should be large enough to accept strings up to FULLNAME_SIZE in length.

FullBox This should be large enough to accept strings up to BOX_SIZE in length.

Greetings

Normally, when a person calls a DN which is a Meridian Mail user and the DN rings but is not answered, the call is answered by Meridian Mail. Meridian Mail then plays that user’s greeting to the caller.

If the call came from an outside line (an “external” call), a different greeting is played than if the call came from a telephone directly attached to the PBX (an “internal” call).

In the following functions, the GreetType parameter can have the following values:

GreetType parameter	**Description**
GR_INTERNAL	Internal greeting
GR_EXTERNAL	External greeting

Function	**Description**
m_DeleteGreeting	Delete greeting
m_OpenGreeting	Open greeting file

m_DeleteGreeting—Delete greeting file

The internal or external greeting can be deleted by this function. If there is no greeting when a call is answered, the Meridian Mail machine plays a standard recorded announcement to the caller.

If the greeting has been opened by the m_OpenGreeting() function, it should be closed with m_CloseFile() before it is deleted.

Header files to include	m_acc.h (Constants, return codes) m_admin.h (Function declarations)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_OPEN_GREETING Greeting is open MME_BAD_PARAMETER Invalid personal greeting type
See also	m_CloseFile m_OpenGreeting

Declaration

```
short m_DeleteGreeting(GreetType, rc)
short GreetType;                              /* greeting type */
short *rc;  /* returned status code */
```

m_OpenGreeting—Open greeting file

A greeting can be played or recorded by opening the greeting as if it were a standard voice file. If the greeting does not exist, it is created.

The m_PlayVoice(), m_RecordVoice(), and other Voice Operations functions can then be used on the file. When the modifications have been completed, the file should be closed with the m_CloseFile() function.

The file is opened with position at beginning of file. If the greeting file is closed without any voice in the file, the file is deleted.

The maximum length of a greeting is set by the system administrator.

Header files to include	m_acc.h (Constants, return codes) m_admin.h (Function declarations)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_OPEN_GREETING Greeting is open MME_MAX_OPEN Maximum open file limit reached MME_BAD_PARAMETER Invalid personal greeting type
See also	m_CloseFile m_CommitFile m_PlayVoice m_RecordVoice

Declaration

```
short m_OpenGreeting(GreetType, FileHandle, Created, rc)
    short GreetType;    /* greeting type */
    short *FileHandle;  /* returned handle for play/record */
    short *Created;     /* returned flag: greeting created?*/
    short *rc;          /* returned status code */
```

Created This indicates whether or not the greeting was just created. Its value is TRUE or FALSE. The file is opened in update mode (see the m_OpenFile() function for a description of file modes).

Personal Verification

Every Meridian Mail user’s name can be recorded as a personal verification. When a user receives a voice message and uses Meridian Mail Voice Messaging to play the message, the name of the sender is played to the receiver as part of the introduction to the message. These functions allow a personal verification to be played, recorded, or deleted.

Tag	**Description**
m_DeletePersVerif	Delete personal verification.
m_OpenPersVerif	Open personal verification file.

m_DeletePersVerif—Delete personal verification

This function deletes a personal verification recorded previously.

If the personal verification has been opened by the m_OpenPersVerif() function, it should be closed with m_CloseFile() before it is deleted.

Header files to include	m_acc.h (Constants, return codes) m_admin.h (Function declarations)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_OPEN_PERS_VERIF Personal verification is open MME_RESTRICTED Personal verification modification restricted to administrator access only
See also	m_CloseFile m_OpenPersVerif

Declaration

```
short m_DeletePersVerif(rc)
short *rc;  /* returned status code */
```

m_OpenPersVerif—Open Personal Verification

This function allows a personal verification to be opened as a voice file. If the personal verification does not exist, it is created. The file can then be recorded into or played. The file is opened at the beginning of file.

The m_PlayVoice(), m_RecordVoice(), and other Voice Operations functions can then be used on the file. When the modifications have been completed, the file should be closed with the m_CloseFile() function.

A personal verification has a maximum length of 10 seconds. If this limit is reached, recording will stop and an m_OnRecordEnd() event will be generated. If the file is closed without any voice in the file, the file is deleted. The user's personal verification is updated when the file is closed.

Personal verification modification can be restricted to administrator access only. The administrator can then make the function available to users through the Voice Messaging Option in the Meridian Mail system.

Header files to include	m_acc.h (Constants, return codes) m_admin.h (Function declarations)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_OPEN_PERS_VERIF Personal verification is already open MME_RESTRICTED Personal verification modification restricted to administrator access only
See also	m_CloseFile m_CommitFile m_PlayVoice m_RecordVoice

–continued–

Declaration

```
short m_OpenPersVerif(FileHandle, Created, rc)
     short *FileHandle; /* returned handle for play/record */
     short *Created;   /* returned flag - PersVerif created? */
     short *rc;        /* returned status code */
```

Created This indicates whether or not the personal verification was just created. Its value is TRUE or FALSE.

Other Functions

One other administration function is available:

Function	Description
m_UserPassword	Change user password

m_UserPassword—Change current password

The m_UserPassword() function can be used to update the user's password.

The maximum length of a password is given by the constant PSWD_SIZE in the header file. The Meridian Mail administrator may impose a restriction on the minimum length of user passwords in the system and on the number of unique passwords which must be used before old passwords can be reused. Such restrictions will be imposed by this function.

Header files to include	m_acc.h (Constants, return codes) m_admin.h (Function declarations)
Prerequisites	Registered Acquired Logged on
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before "Acquire" MME_PSWD_OLD Cannot reuse old passwords MME_PSWD_TOO_SHORT Use a longer password MME_BAD_PSWD Invalid password MME_NOT_NUMERIC Non-numeric in numeric field

Declaration

```
short m_UserPassword(OldPswd, NewPswd, rc)
char *OldPswd;                          /* current password */
char *NewPswd;                          /* password to be set */
short *rc;  /* returned status code */
```

NewPswd, OldPswd These should point to null-terminated strings. Maximum length equal to PSWD_SIZE as defined in m_acc.h.

Chapter 13: Event handling functions

Meridian Mail signals applications of many asynchronous, unsolicited events including arrival of a new voice message or end of a playback request. User-written event handler functions may be specified for each kind of event.

There are two ways in which event handler functions may be invoked:

- ***Event Interruption*** When an event arrives, the application process is interrupted by a UNIX “SIGUSR2” software signal which causes a signal handler routine (part of the API library) to execute and invoke the corresponding event handler function (if installed). Using this type of event notification, system calls and API functions may be interrupted (and fail) due to the arrival of an event.
- ***Polling*** When an event arrives, the application is not interrupted or notified of the arrival. Instead, the event is placed into the process’s receive message queue. Events are dequeued and processed in FIFO order when a call to m_EventCheck is made. Applications sitting idle or waiting for an event to occur must call m_EventCheck to be notified of an event arrival.

  An application may switch between these two modes by calling the API functions m_AutoEventON and m_AutoEventOFF. By default, event interruption is enabled (auto-event ON).

  Note: Application processes should not manipulate the SIGUSR2 UNIX signal regardless of the mode of event handling used. This will cause event notification and delivery to fail and may cause other API functions to fail.

Function	**Description**
m_AutoEventOff	Disable auto-event notification.
m_AutoEventOn	Enable auto-event notification.
m_EventCheck	Check for pending events.
m_OnBRWarn	Temporary storage full warning
m_OnCallProgress	Call progress information
m_OnDigit	Dual-Tone Multi-Frequency (DTMF; touch-tone) digit received.
m_OnError	Meridian Mail error occurred.
m_OnIncomingCall	Incoming call received.
m_OnLhEvent	Link handler event or error
m_OnNewMessage	New message received.
m_OnPlayEnd	End of voice playback
m_OnRecordEnd	End of recording
m_OnSessionEnd	Session disconnected
m_OnTimeout	Timeout

An event handler may be installed for each type of event and may be deinstalled by passing a NULL pointer to the installation routine. All event handler installation functions return a pointer to the previously installed event handler, if any, for the corresponding event. This is useful for applications which temporarily replace an existing event handler and later reinstall the original event handler.

An event handler should never call any Meridian ACCESS API library function. These functions may not work properly if they are used within event handlers, and unpredictable problems may be encountered.

m_AutoEventOff—Disable Auto Event notification

This function allows an application to block receipt of Meridian Mail events until it is ready to receive them.

Any events that arrive while auto event notification is off can only be received using the m_EventCheck function.

Header files to include	m_acc.h (Constants, return codes)
Prerequisites	Registered Acquired
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH
See also	m_EventCheck m_AutoEventOn
Declaration short AutoEventOff(rc) short *rc; /* returned status code */	

m_AutoEventOn—Enable Auto Event notification

This function allows applications to restart auto event notification after it has been turned off (using m_AutoEventOff).

This function triggers the receipt of any events that were queued while auto event notification was turned off and were not received using m_EventCheck.

Header files to include	m_acc.h (Constants, return codes)
Prerequisites	Registered Acquired
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH
Declaration	

```
short_m_AutoEventOn(rc)
short *rc    /* returned status code */
```

m_EventCheck—Check for pending events

This function triggers the receipt of all events (if any) that have occurred since auto event checking was turned off, or since the last call to m_EventCheck. A “set” bit in “Events” indicates that the corresponding event occurred at least once.

While auto event notification is ON, arriving events are automatically received by the appropriate event handler. While auto event notification is OFF, events are only received if the application calls m_EventCheck().

Header files to include	m_acc.h (Constants, return codes) m_event.h (Function declarations, constants)
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_AUTOEVENTON m_EventCheck should only be used when auto event notification is off MME_NOT_ACQUIRED Meridian Mail channel not acquired MME_API_INTERRUPTED A UNIX signal occurred while waiting for events
See also	m_AutoEventOn m_AutoEventOff

Declaration

```
short m_EventCheck(MaxTime,Events,rc)
short *MaxTime   /* maximum time to wait for events */
long *events     /* returned bit map of events received */
short *rc        /* returned status code */
```

MaxTime This is updated to reflect the number of seconds remaining if the function is interrupted or one or more events are received.

Events These are set to either EV_XXXXX to indicate which event(s) occurred or to EV_NONE to indicate that no events occurred in MaxTime seconds (that is, timed out), if the function returns TRUE and “rc” is MMS_OKAY. EV_XXXXX represents the value that would be returned for either a single event or events.

If more than one event occurred, the value returned is the logical ‘‘OR” of the values (defined in m_event.h) which correspond to the occurred events.

m_OnBRWarn—Temporary storage full warning

This event is sent to an application when the temporary storage on Meridian Mail, used to save changes made to files until they are committed or closed, is full.

Invoking either a commit (m_CommitFile) or close (m_CloseFile) operation on the specified FileHandle reclaims the file system resources. If the application does not commit or close the specified file within four minutes of receiving this event, the file is automatically committed, and another BRWarn event is sent to the application with the Committed parameter set to TRUE.

The Meridian Mail file system can save approximately 30 minutes of uncommitted voice editing changes.

Applications should be designed to avoid this event (by periodically calling m_CommitFile or m_CloseFile), rather than simply reacting to its receipt. Typically, this event only occurs when editing a voice segment file since this type of operation is likely to cause many updates to the file.

Header files to include	m_acc.h	(Constants, return codes)
	m_event.h	(Function declarations, constants)

To install the Event Handler:

```
void (*m_OnBRWarn(void (*)(short, short)))(short, short)
```

Event handler declaration

```
void BRWarnHandler(FileHandle, Committed)
short FileHandle  /* file handle of file out of resources */
short Committed   /* TRUE when file has been committed */
```

m_OnCallProgress—Call progress information

Changes in the state of a telephone call can be checked by specifying a call progress event handler. When any change occurs, this event handler is invoked with parameters indicating the nature of the change.

The event handler is automatically invoked when a change in state occurs for an existing telephone call, after the following call is executed:

m_OnCallProgress(CallProgressHandler)

Note: If the link between Meridian Mail and the telephone switch is not an AML/CSL link, certain Call Progress events may not be returned.

Header files to include	m_acc.h (Constants, return codes) m_event.h (Function declarations, constants)
See also	m_MakeCall m_DisconnectCall m_TransferCall m_TransferCallRevert m_CallMsgSender

To install the Event Handler:

```
void (*m_OnCallProgress(void (*)(short, short)))(short, short);
```

Event handler declaration

```
void CallProgressHandler(StateChange, StateInfo)
short StateChange;  /* new state which has been entered */
short StateInfo;    /* additional info on state change */
```

StateChange The following state changes are detected:

State change	Description
CP_ESTABLISHED**	Call has been established that is, answered.
CP_RINGING**	Called party has been rung.
CP_BUSY**	Called party is busy.
CP_REORDER**	Call has been rejected.

CP_FAILURE	Call connection attempt has failed.
CP_COLLISION	Call resulted in collision.
CP_COMPLETED	Call Transfer/Conference /Reconnect successful.
CP_DISCONNECT	Set has gone on-hook.
CP_DNUPDATE	Called/calling DN has changed.

StateInfo Additional call information is provided with some state changes. The following are possible values returned:

State information	**Description**
CI_NO_INFO	No additional information available

State information for CP_RINGING	**Description**
CI_ACD_QUEUED	Waiting in ACD queue
CI_ATT_QUEUED	Waiting in attendant queue
CI_ESN_QUEUED	Waiting in ESN queue
CI_ACD_RINGING	Idle ACD agent found, being rung
CI_ATT_RINGING	Idle attendant found, being rung
CI_TRUNK_SEIZED	A trunk has been seized.
CI_PARKED	The call is parked.

State information for CP_REORDER	**Description**
CI_BLOCKED	Call blocked due to no resources
CI_RESTRICTED	Access restricted
CI_SIT_TONE	SIT-based tone detected

State information for CP_FAILURE	**Description**
CI_BAD_ORIG_DN	Bad originating DN
CI_BAD_CALLED_DN	Bad called DN
CI_INC_ORIG_DN	Incomplete originating DN
CI_INC_CALLED_DN	Incomplete called DN
CI_SWITCH_ERROR	Internal switch error
CI_ORIG_BUSY	Originating party is busy.
CI_MTE_BUSY	Originating party is maintenance busy.
CI_OTHER_BUSY	Another user is using the DN.
CI_ON_HOOK	500/2500 set is on hook.
CI_ORIG_QUIT	Originating party disconnected
CI_INVALID_TN	Invalid TN
CI_BAD_CUSTOMER	Incorrect customer number
CI_BAD_IXFER	Initialization of transfer failed.
CI_BAD_CXFER	Complete of transfer failed.
CI_BAD_RECON	Reconnect failed.
CI_BAD_CONF	Conference failed.
CI_INT_ERROR	Internal error
CI_UNKNOWN_TONE**	Tone detected, type not known

State information for CP_COLLISION	**Description**
CI_SAME_SERVICE	Collision with the same service
CI_OTHER_SERVICE	Collision with a different service

State information for CP_ESTABLISHED and CP_COMPLETED	**Description**
CI_VOICE_DETECTED**	Voice detected on remote end
CI_SHORT_SILENCE**	Ringing was detected, call answered but no voice detected
CI_LONG_SILENCE**	Call answered but voice not detected
CI_PAGER**	Pager tone detected

Note: States marked with '**' are TONES detected by DSP; all others are for those on internal Meridian 1 switch calls.

Call Progress Tones There are three tones.

- ***Pager tone*** This is a 1400 Hz tone lasting for 1.5 seconds. It signifies that the call is connected.
- ***SIT tone (Special Information Tone)*** This is a standard three-tone sequence heard when a number dialed cannot be reached or is unobtainable. The call is disconnected.
- ***Unknown tone*** This sounds if a tone was heard, but not recognized. The call is disconnected.

m_OnDigit—Digit received

When a digit is received from the DTMF keypad of a telephone set, the digit is forwarded to the application.

The event handler is automatically invoked after the following call is executed and a digit key is pressed on the telephone set:

m_OnDigit(DigitHandler)

Header files to include	m_acc.h (Constants, return codes) m_event.h (Function declarations, constants)

To install the Event Handler:

```
void (*m_OnDigit(void (*)(short)))(short);
```

Event handler declaration

```
void DigitHandler(RecDigit)
short RecDigit;                              /* received digit value */
```

RecDigit The value is one of the following:

- 0 - 9
- *
- #
- A -D

m_OnError—Error notification

Several types of asynchronous errors may occur during execution. These errors are returned to the application by this event.

The error handler is automatically invoked after the following call is executed and an asynchronous error occurs:

m_OnError(ErrorHandler)

Header files to include	m_acc.h (Constants, return codes) m_event.h (Function declarations, constants)

To install the Event Handler:

```
void (*m_OnError(void (*)(short)))(short);
```

Event handler declaration

```
void ErrorHandler(Type)
short Type;                                  /* type of error */
```

Various asynchronous errors are reported by this event. The most common ones are the following:

Asynchronous error	**Description**
ER_CMD_FAILURE	Meridian Mail internal command has failed.
ER_VOICE_FAILURE	Voice operation failed.
ER_QUEUE_FULL	m_PlaySegs() queue is full.

m_OnIncomingCall—Incoming phone call

If a call is placed to the voice channel associated with an active Meridian ACCESS session, this event will notify the session of the call.

The IncomingCall handler is automatically invoked after the following call is executed and a new call has been received:

m_OnIncomingCall(IncomingCallHandler)

The m_AnswerCall() function must be used to answer the incoming call within 15 seconds, or the call will be transferred automatically to the revert DN for the application, and the Meridian Mail session for the application will be dropped.

In AML/CSL configurations, the calling DN (FromDN) and the called DN (ToDN) are only available if AML (PRA) trunks are used for the incoming call. Even with AML trunks, the FromDN may not always be set depending on where the call originated.

If your AML/CSL-configured system uses DID trunks configured with DNIS, the Meridian 1 will pass the DNIS value received to ACCESS applications.

Header files to include	m_acc.h	(Constants, return codes)
	m_event.h	(Function declarations, constants)

To install the Event Handler:

```
void (*m_OnIncomingCall(void (*)(char *, char *)))(char *, char *);
```

Event handler declaration

```
void IncomingCallHandler(FromDN, ToDN)
char *FromDN;                          /* DN of caller (if known) */
char *ToDN;                            /* DN of local set */
```

ToDN, FromDN These are on a temporary stack when the event handler is executed. If these parameters must be referenced after the handler has returned, they must be copied to a static data area. ToDn and FromDN should be large enough to accept strings of up to DN_SIZE in length.

m_OnLhEvent—Link Handler event or error

Fatal errors, recoverable errors, and other events can occur while the Link Handler is running. LH error/events will only be sent to the registered monitor process (if any). A monitor process always has “auto-event” notification ON while it is registered and cannot turn it OFF.

Header files to include	m_acc.h (Constants, return codes) M_LH.H (Function declarations)
See also	m_RegisterAsMonitor

Declaration

```
void (m_OnLhEvent(void (*)(short)))(short);
```

Event Handler Declaration:

```
void LhEventHandler(Event)
short Event        /* Link Handler event that occurred */
```

Recoverable and fatal errors and events are listed in the following tables.

Recoverable error/events	Description
LH_TOO_MANY_RETRIES	Retry limit exceeded on sending packet
LH_LINK_RCV_TO	Link Receiver timeout: on process restart
LH_PORT_ERROR	Couldn’t send a char out serial port
LH_PORT_CLEARED	Serial port error has cleared
LH_LOST_SYNCH	Lost synchronization with MM
LH_IN_SYNCH	Synchronization with MM achieved
LH_TERM_PROTOCOL	MM request to terminate protocol
LH_REG_TIMEOUT	Client process inactivity; deregistered
LH_BAD_API	Unknown LH command received from a client
LH_BAD_MSG_TYPE	Unknown message type received from API queue

Fatal error/events	**Description**
LH_BAD_VERSION	TC running incompatible protocol software
LH_SYS_ERR	TC internal system error
LH_UNKNOWN_SIG	Received unknown signal from TC
LH_BAD_API_QUEUE	Could not access (or create) message queues
LH_BAD_EV_QUEUE	Could not access (or create) event queues
LH_BAD_SEMAPHORE	Could not create semaphore
LH_SEMA_INIT_FAILURE	Could not initialize semaphore
LH_BAD_LOG_FILE	Could not open the log file
LH_BAD_CNF_FILE	Could not open configuration file
LH_NO_LHRX	Link Receiver executable file not found
LH_FORK_ERROR	Could not (re)fork the Link Receiver process

m_OnMboxStatus—Mailbox Status Change

Once a process has established itself as the ENS application, it can expect to receive “Mailbox Status” events whenever important mailbox changes take place on the Meridian Mail system. These changes will be propagated to the ENS application in the form of a “Mailbox Status” event if the user mailbox in question has it’s ENS flag turned ON (see m_SetMboxEHN API). This is the only method for the ENS application to receive status information on a user mailbox as it happens. Thus, it is recommended that a process install this event handler before establishing itself as the ENS application (that is, before calling m_AcquireENS). For example

m_OnMboxStatus (MboxStatusHandler)

m_AcquireENS(rc);

In the above example, the installed function *MboxStatusHandler* is automatically invoked as “Mailbox”events come in from Meridian Mail.

Header files to include	m_acc.h	(Constants, return codes)
	m_event.h	(Function declarations, constants)

To install the Event Handler:

```
void (*m_OnMboxStatus(void(*)(MboxInfo))(MboxInfo);
```

Event handler declaration

```
void MboxStatusHandler (MboxUpd)
    /*User Mailbox status update info structure */
    struct MboxInfo *MboxUpd;
struct MboxInfo{
    /*Mlbox event-either VM_NEW/VM_SESSION_END*/
    short MboxEvent;

    /*MM Customer ID number where user mailbox resides */
    short CustNum;

    /*Mailbox DN which was updated <= 18 chars */
    char Mailbox [BOX_SIZE];

    /*Current number of unread Voice Messages */
    short NumVM;

    /*Current number of inread Urgent Voice Messages */
    short NumUVM;

    /*Message Waiting indicator status-either ON or OFF */
    short MWI;

    /*Sender ID (Last Name) <= 19 chars */
    char Sender [SENDER_SIZE];
}
```

MboxEvent This field is used to specify the type of Voice Messaging event that has just taken place. An event can be a NEW message or VM_SESSION_END which only occurs when a user logs out of a voice messaging session.

CustNum This field contains the logical Customer ID number on the Meridian Mail system where the user mailbox resides. It is a short integer which is always greater than or equal to 0.

Mailbox This field contains the User Mailbox DN that has just a Voice Messaging status change on the Meridian Mail system. The User Mailbox is a numeric string that may also include a networking and/or NMS prefix and can be no longer than 18 digits in length (excluding NULL).

NumVM This field contains the current number of unread voice messages in the User Mailbox. It is a short integer which is always greater than or equal to 0.

NumUVM This field contains the current number of unread URGENT voice messages in the User Mailbox. It is a short term integer which is always greater than or equal to 0.

MWI This field contains the current status of the User Mailbox Message Waiting Indicator (MWI). The MWI can either be ON or OFF.

Sender This field contains the Sender’s Identification (Sender’s Surname). The Sender’s ID is a numeric string which can be no longer that 19 characters in length (excluding NULL).

> ***Note:*** This field is not necessarily always filled and may be NULL.

m_OnNewMessage—New message arrival

If a user is logged on and a new voice message is received for that user, this event is generated. This allows a voice messaging application to be informed of additions to the user’s cabinet.

The following call installs a handler:

m_OnNewMessage(NewMessageHandler)

The specified handler is automatically invoked after a new message arrives.

Header files to include	m_acc.h (Constants, return codes) m_event.h (Function declarations, constants) m_file.h (Structure)
See also	m_FilePattern m_RetrieveFile m_PlayMsg m_ReplyMsg m_ForwardMsg

To install the Event Handler:

```
void (*m_OnNewMessage(void (*)(struct FileInfo *)))(struct FileInfo *);
```

Event handler declaration

```
void NewMessageHandler(FileRec)
struct FileInfo *FileRec;                    /* new msg file info */
```

FileRec On a temporary stack when the event handler is executed. If this structure must be referenced after the handler has returned, it must be copied to a static data area.

m_OnPlayEnd—End of playback

When the playing of a voice file is stopped by Meridian Mail (rather than by m_StopVoice() from the application), an end-of-playback event is generated. This allows an application to know when the playback of a voice file is completed. Meridian Mail stops the playback at the beginning of a file if m_SkipVoice() has skipped backwards to the start of the file.

Header files to include	m_acc.h (Constants, return codes) m_event.h (Function declarations, constants)
See also	m_PlayVoice m_PlayMsg m_PlaySegs m_StopVoice m_SkipVoice

To install the Event Handler:

```
void (*m_OnPlayEnd(void (*)(short)))(short);
```

Event handler declaration

```
void PlayEndHandler(Location)
short Location;      /* position in file when stopped */
```

Location The location is returned as one of the following:

Location	Description
PE_BOF	Stopped at beginning of file
PE_EOF	Stopped at end of file

The event handler is invoked automatically after the following call is executed and playback is stopped by Meridian Mail:

m_OnPlayEnd(PlayEndHandler)

m_OnRecordEnd—End of recording

When the recording of a voice segment ends abnormally, an event is sent to the application. To receive such an event, the corresponding event handler must be installed.

The handler is automatically invoked after the following call is executed and recording is stopped by Meridian Mail:

m_OnRecordEnd(RecordEndHandler)

Header files to include	m_acc.h (Constants, return codes) m_event.h (Function declarations, constants)
See also	m_RecordVoice m_StopVoice

To install the Event Handler:

```
void (*m_OnRecordEnd(void (*)(short)))(short);
```

Event handler declaration

```
void RecordEndHandler(Reason)
    short Reason;                              /* reason recording ended */
```

Reason The following are possible values:

Possible reason	Description
RE_SILENCE	Silence timeout
RE_TOOLONG	Recording too long
RE_DISKFULL	Disk drive is full.
RE_CABFULL	User’s cabinet is full.

m_OnSessionEnd—Session disconnect

If sessions and voice channels are released by Meridian Mail instead of by the application, a Session Disconnect event is sent to the application. The application does not lose its registration with Meridian ACCESS and may try to reacquire a new Meridian Mail session.

The handler is automatically invoked after the following call is executed and the active session has been released by Meridian Mail:

m_OnSessionEnd(SessionEndHandler)

Header files to include	m_acc.h (Constants, return codes) m_event.h (Function declarations, constants)
See also	m_OnTimeout m_AnswerCall m_AcquireOnIncomingCall

To install the Event Handler:

```
void (*m_OnSessionEnd(void (*)(short)))(short);
```

Event handler declaration

```
void SessionEndHandler(Reason)
short Reason;                                 /* reason for disconnect */
```

Reason The following are possible values:

Reason for disconnection	Description
SE_TIMEOUT	The session has been inactive and has timed out, or m_AcquireOnIncomingCall request has timed out.
SE_SHUTDOWN	Meridian Mail was shut down by administrator.
SE_SYSERR	System error
SE_NOANSWER	Incoming call was not answered with m_AnswerCall within 15 sec.

–continued–

SE_BADLOGON	Too many bad logon attempts
SE_AOIC_DISC	Call disconnect received in SMDI configuration when using m_AcquireOnIncomingCall

m_OnSoundDetect—Stop Detecting Sound or silence

If a request is made to monitor sound or silence on an active voice channel using m_SoundDetect, this event notifies the application of the results of the monitoring period. This is the only method for an application to receive the results of sound or silence monitoring. Thus, the application must install this event handler before calling m_SoundDetect. The installed function is automatically invoked after the results of the monitoring session are known.

m_OnSoundDetect(SoundDetectHandler)

m_SoundDetect(SOUND,PERIOD,MINDUR,MAXDUR, IWSDUR,rc)

Header files to include	m_acc.h (constants, return codes) m_event.h (function declaration, constants)

To install the Event Handler:

```
void (*m_OnSoundDetect(void(*)(AudioSignal,long)))(AudioSignal,long);
```

Event handler declaration

```
void SoundDetectHandler(Context, Duration);
    AudioSignal Context;/*requested detection either SOUND or SILENCE  */
    long Duration;      /*duration of the audio signal detected-milliseconds */
```

Context This parameter specifies the type of audio signal originally requested from the m_SoundDetect call. This will be either sound or silence as defined by the literals SOUND/SILENCE included in “m_voice.h”.

Duration This parameter specifies the actual duration (in milliseconds) of the audio signal detected. A duration of zero indicates that the minimum duration was not met within the desired period.

m_OnTimeout—Timeout notification

Each command sent to the Meridian Mail system resets a “watchdog” timer. If the timer runs down as a result of inactivity from the application, a Timeout event is sent to the application. If the event is not responded to, Meridian Mail assumes that the application has failed. It then sends a Session Disconnect event. This frees the session and voice channel so that other applications can use them.

The Timeout event is sent if no non-local Meridian ACCESS command has been issued for one minute. A further wait of SecsToEnd seconds (usually 30 seconds) occurs while pausing for a response to the Timeout event.

This function is automatically invoked after the following call is executed and a timeout notification is sent by Meridian Mail:

m_OnTimeout(TimeoutHandler)

Header files to include	m_acc.h (Constants, return codes) m_event.h (Function declarations, constants)
See also	m_OnSessionEnd m_SetTimeout m_TimeoutOFF m_TimeoutContinue

To install the Event Handler:

```
void (*m_OnTimeout(void (*)(short, short)))(short, short);
```

Event handler declaration

```
void TimeoutHandler(Reason, SecsToEnd)
short Reason;      /* cause of timeout */
short SecsToEnd;   /* seconds until session ends */
```

Reason One is currently defined:

Reason	Description
TO_INACTIVE	Application is inactive

Chapter 14: Link Handler functions

Link Handler functions do not communicate with Meridian Mail. Data is only exchanged between the calling process and the Link Manager Process (LMP).

If a process registers with the Link Handler via the m_RegisterAsMonitor function, it can only use the functions described below. If it registers using the m_Register function, then it can use all functions except m_RegisterAsMonitor, m_DeregisterAsMonitor and m_OnLhEvent.

Link Handler General functions

These functions may be used by any UNIX process that is registered (using m_Register or m_RegisterAsMonitor) with a Meridian ACCESS LMP.

- m_GetLinkOM
- m_ResetLinkOM
- m_LinkSanity()

Link Handler Monitor functions

Only UNIX processes registered with the Link Handler via the m_RegisterAsMonitor function may use these functions. Use by any other process will fail (return FALSE), and “rc” will be set to indicate the error.

- m_RegisterAsMonitor()
- m_DeregisterAsMonitor()
- m_OnLhEvent().

m_GetLinkOM—Collect Operational Measurements

The m_GetLinkOM function allows any registered process to retrieve operational measurements associated with the link.

Time fields in the LH_OM structure are defined by the number of seconds since 00:00:00 GMT January 1, 1970 (according to local UNIX OS time).

Header files to include	m_acc.h (Constants, return codes) m_lh.h (Function declarations)
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH

Declaration

```
short m_GetLinkOM(data, rc);
struct LH_OM *data;
short *rc;
```

OM information is placed into the LH_OM structure as follows:

```
struct LH_OM {
    unsigned long ReportPeriod; /* seconds */
    struct {              /* all packet types */
        unsigned long Syncs;     /* synchronization pkts */
        unsigned long Datas;     /* user data pkts */
        unsigned long Polls;     /* poll pkts */
        unsigned long Terms;     /* termination pkts */
        unsigned long Acks;      /* acknowledgement pkts */
        unsigned long Naks;      /* error ack. pkts */
    } RcvCnt, SndCnt;       /* received & sent pkt counts */
    struct {              /* pkt error counts */
        unsigned long Format;  /* no ”stop”, too short, etc. */
        unsigned long Checksum;   /* bad checksum */
        unsigned long SeqAck;    /* bad Ack sequence number */
        unsigned long SeqNak;    /* bad Nak sequence number */
        unsigned long SeqData;   /* bad Data sequence n. */
        unsigned long Type;      /* bad packet type */
    } RcvErr;      /* counts of received pkt errors */
    unsigned long peakReg; /* max simul. reg. processes */
    unsigned long currReg; /* curr. processes registered */
    unsigned long totReg;  /* curr. total process registers */
    unsigned long lastReset; /* start time of current count */
    unsigned long lastSync;  /* time of last link synch */
};
```

m_ResetLinkOM—Reset Operational Measurements counts

This function allows any registered process to reset the operational measurement counts to zero.

In addition, the “lastReset” field of the LH_OM data will be updated to the current time, to reflect this call.

Header files to include	m_acc.h	(Constants, return codes)
	m_lh.h	(Function declarations)
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH	
Declaration short m_ResetLinkOM(rc) short *rc;		

m_LinkSanity—Check Link Handler operational status

This function allows any registered process to check the operational status of the Link Handler.

Header files to include	m_acc.h (Constants, return codes) m_lh.h (Function declarations)
Prerequisites	Registered
Status codes	MMS_LH_IN_SYNCH Link Handler is in synch with Meridian Mail MMS_LH_NOT_SYNCH Link Handler is not in synch with Meridian Mail
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH

Declaration

```
short m_LinkSanity(Status, rc)
short *Status;      /* LH status, if operation successful*/
short *rc;          /* returned error code */
```

Status The returned "status" code is only valid when the operation is successful (m_LinkSanity returns TRUE).

m_RegisterAsMonitor—Register process as a monitor

The m_RegisterAsMonitor function allows a process to register with the LMP as a monitor process. The calling process will have “auto event notification” turned on automatically by this function (it cannot be turned off by a monitor process), so the process should also call the m_OnLhEvent MaxTime Amountfunction, to install an appropriate event handler routine. Only one process may be registered with the LH as a monitor process.

This function is similar to the m_Register function in that they both register the calling process with the LMP. However, a process registered using the m_Register function may use regular functions that communicate with Meridian Mail. A monitor process can only use other Link Handler functions to monitor the link, and cannot communicate with Meridian Mail.

The three minute “inactivity” timeout that applies to other registered processes does not apply to a process registered as a monitor process.

Link handler functions will use the LogicalLink number specified with the m_SetEnv function.

Header files to include	m_acc.h (Constants, return codes) m_lh.h (Function declarations)
Return codes	MME_MONITOR_EXISTS An LH monitor process already MME_EVENT_QUEUE_DOWN System error accessing event message queue MME_BAD_SEM_KEY Could not access / open a semaphore MME_ALREADY_REGISTERED Calling process is already registered
See also	m_SetEnv

Declaration

```
short m_RegisterAsMonitor(rc)
short *rc;
```

m_DeregisterAsMonitor—De-register process as a monitor

This function allows a process to de-register (with the LMP) as a monitor process. In the same manner as m_RegisterAsMonitor is analogous to m_Register, this function is analogous to the m_Deregister function.

Header files to include	m_acc.h (Constants, return codes) m_lh.h (Function declarations)
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_MONITOR Calling process is not registered as a monitor MME_EVENT_QUEUE_DOWN System error accessing event message queue MME_LH_NOT_SYNCH Link Handler is not in synch with Meridian Mail
See also	m_RegisterAsMonitor

Declaration

```
short m_DeregisterAsMonitor(rc)
short *rc;
```

Chapter 15: Starting and stopping the Link Handler

Any UNIX process may start/stop the Meridian 1 ACCESS Link Handler software using the m_StartLink or m_StopLink API function. Since these application programming interfaces (APIs) start and stop the Link Handler software, the calling process does not have to be “registered”, as is the case with most other ACCESS APIs.

Details regarding the starting and stopping of the ACCESS Link Handler software may be found in the ACCESS *Developer's Guide* (NTP 555-7001-316).

m_StartLink—Start the Link Handler

This function allows a process to start up the LMP, and become its parent. The process does not have to be “registered”. It is recommended that the calling process call the m_LinkSanity function, after successfully starting the Link Handler software, to verify that it (the LH software) is running.

After calling the m_StartLink function, all messages from the LMP are directed to the system console’s screen, while the output from the calling process continues unaltered. The output functionality of the Link Handler logfile is not affected by the m_StartLink function.

The m_StartLink function starts the ACCESS link associated with the environment variable LogicalLink (default = 1). To change the default LogicalLink, call the m_SetEnv function.

Header files to include	m_acc.h	(Constants, return codes)
	m_lh.h	(Function declarations)
Return codes	MME_BAD_PATH File specified by “path” not found MME_FORK_ERROR Could not fork process MME_NO_CONFIG Could not find LH configuration file	
See also	m_StopLink	

Declaration

```
short m_StartLink(path, rc)
char *path;         /* directory containing the LH files*/
short *rc;          /* returned status code */
```

Path This should point to a string containing the name of a directory that contains the LMP and LHRX executable files, and the LH configuration file “lh.config”.

m_StopLink—Stop the Link Handler

This function allows the LMP's parent process to gracefully shutdown the LH software. Only the UNIX process that started the LH software, via m_StartLink(), is allowed to call m_StopLink. The process does not have to be "registered".

After sending a signal to the LMP, m_StopLink executes a "wait" system call. This call causes the calling process to block until a child process dies, and returns the process ID of the dead child. If the PID returned (from "wait") is that of a process other than the LMP, the function will return TRUE, and "rc" will be set to MMW_DEAD_CHILD, to indicate that one of the parent's other children has died.

This function should be used with extreme caution since applications cannot communicate with Meridian Mail when the Link Handler is not running.

Header files to include	m_acc.h (Constants, return codes) m_lh.h (Function declarations)
Return codes	MME_NOT_PARENT Calling process did not start LH (via m_Start-Link) MMW_ALREADY_DEAD The LMP does not exist MMW_DEAD_CHILD Calling process had a dead child other than LMP MME_LH_DEFUNCT LMP took too long to die, not notified of its death
See also	m_StartLink

Declaration

```
short m_StopLink(rc);
    short *rc;           /* returned error code */
```

Chapter 16: High-level functions

High-level Meridian ACCESS API functions can be used by application programs to carry out frequently performed operations. In general, high-level API functions are implemented using the existing lower level API functions. By packaging them into higher level routines, the user is not required to reinvent the wheel whenever a particular common functionality is needed.

The following high-level functions are provided:

Function	**Description**
m_CollectDigits	Collect digits from keypad into a string.
m_GetEnv	Get environment parameters for high-level API commands.
m_PlayPrompt	Play a given voice prompt.
m_SetEnv	Set environment parameters for high-level API commands.
m_WaitingForCall	Wait for an incoming call.

m_CollectDigits—Collect digits from keypad into a string

This function collects a series of digits from the telephone keypad and returns a string containing the digits collected.

Input is always taken from the key buffer, not directly from the telephone keypad. Pressing keys on the telephone keypad merely adds digits to the end of the key buffer.

Header files to include	m_acc.h (Constants, return codes)
Prerequisites	Registered Acquired HiLevel On
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_API_NOT_INIT Set HiLevel flag before invoking API MME_BAD_EXITDIGIT Invalid digit key was specified in ExitDigits MME_INTER_KEY_TO InterDigitTimeout has occurred MME_KEY_OVERFLOW Key buffer overflow has occurred MME_API_INTERRUPTED API interrupted by a Meridian Mail event MME_CALL_DISCONNECTED Call has disconnected

–continued–

Declaration

```
short m_CollectDigits(NumDigits, ExitKeys, KeyBuff, LastKey, ClearKeyBuf, rc)
    short NumDigits;    /* max. number of digits to collect */
    char *ExitKeys;     /* array of digits causing immed.
                        exit*/
    char *KeyBuff;      /* returned string of digits
                        collected */
    char *LastKey;      /* returned exit digit key pressed */
    short ClearKeyBuf;  /* clear key buffer  (TRUE/FALSE) */
    short *rc;          /* returned status code */
```

NumDigits This is the maximum number of digits to collect into the string. The return string must be large enough to accommodate user input of at least this size.

By specifying NumDigits=1, this routine can be used to prompt for a menu selection from the telephone keypad where the ExitKeys array contains a NULL list.

ExitKeys It contains the list of digit keys which, when one is pressed, result in an immediate exit from the routing.

KeyBuff The digits collected up to exit point are collected in this string.

LastKey It contains the termination key, or the last key which was pressed. This functionality can be used to specify an input terminator, so that digits are collected until the terminator key is pressed.

ClearKeyBuf The ClearKeyBuf is set to TRUE if you wish to ignore any key presses which have not yet been processed when this function is called.

This routine will only exit when one of the following happens (whichever event occurs first):

1 “NumDigits” digits have been collected.

2 A digit key specified in “ExitKeys” is hit.

3 “InterDigitTimeout” has occurred (see m_SetEnv).

4 A Meridian ACCESS error was encountered.

In all cases, the function will exit immediately and return the string of digits which have been collected so far. In the first two cases, the function will return TRUE, and, in the remaining cases, the function will return FALSE with a non-zero return code.

m_GetEnv—Set environment parameters for high-level API commands

This function returns the current environment parameters which are used by the high-level API functions.

Header files to include	m_acc.h (Constants, return codes)
See also	m_SetEnv

Declaration

```
short m_GetEnv(Env,rc)
    struct EnvInfo *Env;   /* advice parameters retrieved */
    short *rc;             /* returned status code     */

struct EnvInfo {
    short HiLevel;  /* flag indicating Hi Level APIs in use */
    short KeyBufferSize;      /* size of the ”key buffer” */
    short InterDigitTimeout;   /* timeout for entering a
                                         digit, in secs*/
    char GenericSegs[FNAME_SIZE]; /* filename containing
                                         generic segments */
    short LogicalLink;   /* logical ACCESS link */
};
```

m_PlayPrompt—Play a given voice prompt

This function plays a list of voice segment IDs from a single open voice segment file and waits for the playback to end.

Header files to include	m_acc.h (Constants, return codes)
Prerequisites	Registered Acquired Logged On File Open Connected HiLevel On
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_API_NOT_INIT Set HiLevel flag before invoking API MME_KEY_OVERFLOW Key buffer overflow has occurred MME_API_INTERRUPTED API interrupted by a Meridian Mail event MME_PLAY_TIMEOUT Did not receive PLAYEND event (expected PlayTime expired) MME_CALL_DISCONNECTED Call has disconnected MME_NO_ACTV_CHNL No active voice channel MME_BAD_HANDLE Unassigned file handle MME_BAD_COMMAND Command invalid on this file type MME_BAD_SEG_ID Segment ID not found in file MME_BAD_FLAG Invalid flag

–continued–

Return codes (continued)	MME_OTHER_TELEPHONY Other telephony command in progress MME_PLAYING Play command already in progress MME_NO_SEGS No voice segment in the file MME_DETECT_INPROG Sound detect already in progress

Declaration

```
short m_PlayPrompt(FileHandle, SegmentList, PlayTime, Interruptible, ClearKey-
Buf, rc)
    short FileHandle; /* file containing segments to play */
    short SegmentList[];  /* segment IDs to play (in order) */
    short PlayTime;       /* expected play time (in sec) */
    short Interruptible;  /* prompt Interruptible?
                            (TRUE/FALSE) */
    short ClearKeyBuf;    /* clear key buffer before play?
                            (TRUE/FALSE) */
    short *rc;            /* returned status code */
```

SegmentList The SegmentList is an array of integer voice segment IDs from the voice segment file with the given FileHandle. You may give a maximum number of 50 segment IDs in the SegmentList, and you must terminate the list with a segment ID of 0 to indicate the end of the list.

PlayTime The expected play time for the prompt in seconds. If this time expires and the application does not receive a PlayEnd event, the function will return with an MME_PLAY_TIMEOUT.

Interruptible It indicates whether the prompt may be interrupted by a digit key being pressed on the telephone keypad.

ClearKeyBuf If set to TRUE, the key buffer will be cleared before the playback has finished. If the prompt is interrupted, the key which was pressed to interrupt the playback will be the only item in the key buffer when the playback has finished.

m_SetEnv—Set environment parameters for high-level API commands

This function sets the environment parameters which are used by the high-level API functions.

Header files to include	m_acc.h (Constants, return codes)
Return codes	MME_BAD_PARAMETER Invalid parameter value within Env structure
See also	m_GetEnv

Declaration

```
short m_SetEnv(Env,rc)
    struct EnvInfo *Env;     /*          advice parameters to set */
    short *rc;                            /*  returned status code    */

struct EnvInfo {
    short HiLevel;                        /*  flag indicating Hi Level
                                              APIs in use */
    short KeyBufferSize;                  /*  size of the ”key buffer” */
    short InterDigitTimeout;              /*  timeout for entering a
                                              digit, in secs*/
    char GenericSegs[FNAME_SIZE];         /*  filename containing
                                                  generic segments */
    short LogicalLink;                    /*  LogicalLink to register
                                              with */
};
```

HiLevel A TRUE value for this flag indicates that the high-level API commands will be in use and will override event handlers for the following Meridian Mail events:

- OnDigit
- OnPlayEnd

When the HiLevel flag gets reset to FALSE 9the default state), the original event handlers for the above events will be restored.

KeyBufferSize The size of the key buffer for storing digits which have been entered by a user on the telephone keypad, but have not yet been processed by the application. THe value is used by m_CollectDigits and m_PlayPrompt. The default is 20 the minimum is 1.

InterDigitTimeout (Default=3, minimum >= 1) The number of seconds the API command will wait for a user to enter a digit on the telephone keypad before timing out. The value is used by m_CollectDigits.

GenericSegs (Default=GS_SYSTEM_SEGMENTS). The name of the file which contains the recorded generic segments (for example, “one,” “two,” “dollars,” “percent”).

If the size of the key buffer is changed and the current buffer is not empty, the contents of the current buffer are copied into the new buffer. If the new buffer is too small to hold all of the contents of the current buffer, some digits will be lost. If this function is not called, the default parameter values given above will be used. In addition, the environment parameters may be reset at any time back to their default values by calling m_SetEnv(NULL,rc).

LogicalLink (Default = 1) This is the ACCESS link number that will be used in a configuration with multiple ACCESS links. The m_SetEnv function should be called to establish which link the application will use before it calls the m_Register function, or to set the link number that will be started when calling the m_StartLink function.

> ***Note:*** To use and configure multiple ACCESS links, you can change the ‘LogicalLink’ default to the link number that an application will use by issuing the m_SetEnv function. For more information about multiple links, see the *Meridian ACCESS Developer’s Guide* (NTP 555-7001-316).

m_WaitingForCall—Check for an incoming call

This routine is provided to simplify the process required for handling incoming calls. The m_WaitingForCall() function should be called when an application wants to wait for an incoming call. An m_Acquire() or m_AcquireOnIncomingCall() request must already have been performed.

If m_Acquire has been used to acquire a voice channel and the AcqNewChannel parameter is set to FALSE, an m_AcceptCall() request is issued prior to waiting for an incoming call. Otherwise, m_AcquireOnIncomingCall() request are automatically reissued as required.

If an incoming call arrives within the specified MaxTime, m_WaitingForCall() returns TRUE. Otherwise, it returns FALSE, and ”rc” is set to indicate the error.

Header files to include	m_acc.h (Constants, return codes) m_rm.h (Function declarations) m_event.h (Function declarations, constants)
Prerequisites	Registered Acquired
Return codes	MME_NOT_REGISTERED Calling process is not registered with the LH MME_NOT_ACQUIRED Command invalid before “Acquire” MME_MAX_REQUESTS Max. number of outstanding acquires reached MME_OPER_TIMEOUT Timeout performing operation
See also	m_AcquireOnIncomingCall m_Acquire
Events	m_OnIncomingCall

Declaration

```
short m_WaitingForCall(MaxTime, AcqNewChannel, rc)
    short MaxTime;                    /* Max. time to wait (secs) */
    short AcqNewChannel;              /* Acquire a new channel?
                                      TRUE/FALSE */
    short *rc;                        /* returned status code */
```

MaxTime The MaxTime must be at least MIN_WAIT_TIME or greater.

AcqNewChannel It indicates whether or not the function should release the current channel (if any) and acquire a new channel for the next incoming call.

For more information, refer to the *Developer's Guide* (NTP 555-7001-316).

Appendix A: Meridian ACCESS return codes

This appendix lists all of the symbolic constants(return codes) returned by Meridian ACCESS API functions. Symbolic constants begin with one of the following prefixes:

Prefix	**Description**
MMS	Status
MME	Error
MMW	Warning

Meridian ACCESS Return Codes

Return Code	**Symbolic constant**	**Description**
0	MMS_OKAY	Success
1	MME_BAD_PARAMETER	Bad parameter passed to function
2	MMS_NOT_READY	No result available yet
3	MME_TIMEOUT	No result—command timed out
4	MME_NO_LOCAL_MEMORY	Out of memory (local)
5	MME_INVALID_CLASS	Invalid application class
6	MME_NOT_ACQUIRED	Command invalid before “Acquire”
7	MME_NOT_REGISTERED	Calling process is not registered with the LH
8	MME_ALREADY_REGISTERED	Calling process is already registered with the LH
9	MME_BUSY_DN	DN is busy
10	MME_NOT_ANSWERED	No answer at DN

Return Code	Symbolic constant	Description
11	MME_CALL_REORDER	Call has been rejected
12	MME_CALL_FAILURE	Call connection attempt has failed
13	MME_CALL_COLLISION	Call resulted in collision
14	MME_OPER_TIMEOUT	Timeout performing operation
15	MME_CALL_DISCONNECTED	Call has disconnected
16	MME_NO_QUEUE_SPACE	Msg send failed: no queue space
17	MME_BAD_PROCESS_TYPE	Invalid process type
18	MME_API_QUEUE_DOWN	System error accessing API queue
19	MME_EVENT_QUEUE_DOWN	System error accessing Event queue
20	MME_MONITOR_EXISTS	Monitor function already installed
21	MME_NOT_MONITOR	Client is not the monitor process
22	MME_FUNCTION_NOT_AVAIL	API not usable: wrong ACCESS ver.
23	MME_BAD_SEM_KEY	Could not access/open a semaphore
24	MME_BAD_PATH	No file at path specified
25	MME_FORK_ERROR	Couldn’t fork process at path
26	MMW_ALREADY_DEAD	Link Manager was already dead
27	MME_NOT_PARENT	Did not spawn LMP via m_StartLink
28	MMW_DEAD_CHILD	Caller had dead child besides LMP
29	MME_LH_DEFUNCT	LMP took too long to die
30	MME_LH_NOT_SYNCH	LH not synch with MM cmmd failed
31	MMS_LH_NOT_SYNCH	LH not synch with MM cmmd succeeded
32	MMS_LH_IN_SYNCH	LH is synchronized with MM
33	MME_LH_SICK	LH returned an unexpected value
34	MME_MON_RESTRICTED	API is restricted from monitor
35	MME_NO_CONFIG	no LH configuration file found
99	MME_NOT_SUPPORTED	Operation not currently supported
102	MME_BAD_PSWD	Invalid Password
103	MME_NO_TASK	No MM ACCESS Toolkit available
104	MME_FULL_SERVER	No free blocks, server is full
105	MME_FULL_CABINET	No free disk space in User Cabinet

Return Code	Symbolic constant	Description
106	MME_DO_LOGON	Must be logged on to use this cmd
109	MME_ACCESS_DENIED	Access to account denied
111	MME_COMMAND_FAILED	Command Failed, check SEER console
115	MME_ALREADY_ACQUIRED	Already Acquired
117	MME_MAX_LOGONS	Too many failed m_Logon attempts
120	MME_INVALID_FUNCTION	API function not supported
122	MME_NO_MEMORY	Out of memory
126	MME_BAD_ID	Bad userid or mailbox number
128	MME_BAD_FLAG	Invalid flag (0 or 1 are valid)
129	MMW_DUP_LOGON	Warning: Logged on elsewhere
131	MME_BAD_VERSION	API library being used not supported by Meridian Mail
133	MME_INVALID_CUST	Invalid Customer number specified
134	MME_ALREADY_LOGON	Command not valid while logged in
135	MME_ENS_EXISTS	An application has already acquired ENS
136	MME_NOT_ENS	Must be an ENS app to use this command
150	MME_OPTION_NOT_AVAIL	Option not available to customer
151	MME_MAX_REQUESTS	Max. # of acquire requests reached
152	MMW_ALREADY_RELEASED	Session already released by system
200	MME_NO_ACTV_CHNL	No active voice channel
203	MME_BAD_POSITION	Invalid voice start position
204	MME_BAD_TO_POS	Invalid play position
205	MME_BAD_RECORD_POS	Invalid recording position
208	MME_BAD_DIRECTION	Invalid direction (parameter)
211	MME_CHAN_IN_USE	Voice channel already in use
212	MME_NO_ACQUIRED_CHNL	No voice channel has been acquired
213	MME_NO_INC_CALL	No incoming call to answer
214	MME_DO_ADDONCALL	Must call m_AddOnCall first
215	MME_CHANNEL_READY	m_AcceptCall (already) issued
217	MME_OTHER_TELEPHONY	Other telephony command in progress

Return Code	Symbolic constant	Description
223	MME_PLAYING	Play command already in progress
224	MME_BAD_SEQUENCE	Invalid command sequence
225	MME_RECORDING	Record command already in progress
227	MME_VOICE_FAILURE	Voice operation failure
228	MMS_NO_VOICE	No voice in segment to play
229	MMS_AT_EOS	At end of voice segment
231	MME_SILENCE_TIMEOUT	Ended because too much silence
232	MME_RECORD_LIMIT	Recording limit reached
233	MME_BAD_NUM_SEGS	Bad number of segments specified
235	MME_SEG_Q_FULL	Segment play queue is full
236	MME_INVALID_DTMF	Invalid DTMF string
237	MME_BAD_DETECTION	Context must be SOUND/SILENCE
238	MME_BAD_DURATION	Duration must be <= 5 mins.
239	MME_NO_PREV_DETECT	No Previous Detect in progress
240	MME_DETECT_INPROG	Sound Detect already in progress
250	MME_INSTL_EVENT	Must install event handler first
309	MME_NO_ENTRY_FOUND	No such entry found in directory
400	MME_CABINET	Unable to access user’s cabinet
401	MME_INVALID_HANDLE	Invalid file handle passed to command
402	MME_BAD_HANDLE	Unassigned file handle
403	MME_BAD_COMMIT	Invalid commit flag (parameter)
405	MMS_AT_BOF	Reached the beginning of file
406	MME_READ_MODE	Cannot open Read file in Write mode
407	MMS_AT_EOF	Reached the end of file
409	MME_FILE_OPEN	File is already open
410	MMW_COMMIT_IGNORED	Read-only file: Not committed
411	MME_READ_ONLY	Cannot do command on Read-only file
415	MME_FNAME_FORMAT	Invalid filename format
416	MME_MAX_OPEN	Maximum open file limit reached
419	MME_DO_FILEPAT	Must call m_FilePattern first

Return Code	Symbolic constant	Description
420	MME_FILE_DNE	File does not exist
425	MME_BAD_NEW_FLAG	Invalid new flag passed
426	MME_BAD_MODE	Invalid file access mode used
431	MME_BAD_IMMED	Invalid delete parameter
432	MME_BAD_COMMAND	Command invalid on this file type
433	MME_BAD_SEG_ID	Segment ID not found in file
434	MME_TITLE_LENGTH	Invalid length in field
436	MME_DO_SEGPAT	Must call m_SegPattern first
437	MME_SCRIPT_LENGTH	Invalid script length
438	MME_SCRIPT_RETV	Issue retrieve script cmd first
439	MME_NO_SEGS	No voice segments in the file
441	MME_MAX_SEG_FILES	Too many open seg. files for play
442	MME_MAX_SCRIPT_SIZE	Script for voice segment too long
444	MME_MAX_SEGS	Reached max # segs allowed in file
445	MME_BAD_SEG_TYPE	Bad voice segment file type
446	MME_BAD_LANGUAGE	Invalid language specified
448	MME_BAD_EDIT_POS	Invalid segment editing position
449	MME_BAD_OPERATOR	Invalid segment editing operator
450	MME_BAD_AMOUNT	Invalid amount specified
500	MME_FILE_NOT_MSG	File is not a message file
508	MME_BAD_RCVR	Invalid receiver in address list
509	MME_MAX_RCVRS	Exceeded max. # of msg recipients
511	MME_BAD_SUBJECT	Invalid subject string
512	MME_EMPTY_MSG	Cannot send an empty message
513	MME_NOT_RECEIVED	CallSendr/Reply only on recvd msgs
515	MME_DO_ADDRPAT	Must call m_AddrPattern first
519	MME_EXTERNAL	Cannot reply to external messages
520	MME_FORWARD_PRIVATE	Cannot forward a private message
522	MME_NEED_RCVR	Need 1 or more receivers to send
523	MME_MULTIMATCH	Multiple names matched, specify

Return Code	**Symbolic constant**	**Description**
524	MME_INCOMING	Cannot be used on this message type
525	MME_MAX_DELAY	Delay delivery time too long
526	MME_REMOTE	Remote site not recognized
527	MME_SYS_MSG	Operations invalid on system msgs
528	MME_BROADCAST	Cannot ReplyAll to Broadcast msg
529	MME_AMIS_REPLY	Cannot reply all on AMIS message
600	MME_PDL_DNE	List number not found
601	MME_BAD_PDL_NUM	Invalid PDL list number
602	MME_MAX_PDL_ENTRIES	Exceeded number of entries in PDL
603	MME_USER_PROFILE	Unable to access user profile
622	MME_RESTRICTED	Restricted to admin access only
623	MME_BAD_BOX	Invalid box number
625	MME_BAD_SURNAME	Invalid last name
626	MME_BAD_GIVEN	Invalid first name
627	MME_BAD_LIST	Invalid list number
628	MME_PSWD_TOO_SHORT	Password too short
629	MME_BAD_GREET	Invalid personal greeting type
630	MME_DUP_OLD	Old password and logged on elsewhere
631	MME_PSWD_OLD	(for m_Logon) User’s password has expired (for m_UserPassword) Old passwords cannot be reused
632	MME_OPEN_PERS_VERIF	Personal Verification already open
633	MME_OPEN_GREETING	Greeting already open
634	MME_NOT_NUMERIC	Non-numeric in numeric field
636	MME_NO_MATCHING_BOX	No matching box address in PDL
637	MME_DO_PDLPAT	Must call m_PDLPattern first
638	MME_NOT_PDL	Not a PDL file
639	MME_BAD_MSG_TYPE	Invalid external message type
700	MME_API_NOT_INIT	Set HiLev flag before invoking API
701	MME_BAD_EXIT_DIGIT	Invalid digit in ExitDigits
702	MME_INTER_KEY_TO	Inter Digit Timeout occurred

Return Code	Symbolic constant	Description
703	MME_KEY_OVERFLOW	Key buffer overflow occurred
704	MME_API_INTERRUPTED	API interrupted by MM event
705	MME_BAD_ITEMTOPLAY	ItemToPlay in invalid format
706	MME_BAD_PLAYTYPE	Invalid PlayType specified
707	MME_PLAY_TIMEOUT	PlayEnd event not received
806	MME_BAD_DN	Invalid Directory Number passed
808	MME_BAD_ANSWER	Invalid answer flag
811	MME_RESTRICTED_DN	DN has a restricted prefix
900	MME_LH_TABLE_FULL	LH Register Table full
910	MME_TRANS_TABLE_FULL	LH Trans Table full
1000	MME_ECHO_FAIL	Echo test failed: corrupted string
1005	MME_AUTOEVENTON	m_EventCheck with autoeventon

Appendix B: Header file cross-reference

Header files Functions	m_acc.h	m_local.h	m_rm.h	m_file.h	m_voice.h	m_msg.h	m_seg.h	m_admin.h	m_event.h	m_lh.h	m_ens.h
m_AcceptCall	♦										
m_Acquire	♦		♦								
m_AcquireENS	♦										♦
m_AcquireOnIncomingCall	♦		♦								
m_AddBoxToAddr	♦					♦					
m_AddBoxToPDL	♦							♦			
m_AddNameToAddr	♦					♦					
m_AddNameToPDL	♦							♦			
m_AddOnCall	♦				♦						
m_AddrPattern	♦					♦					
m_AddSeg	♦						♦				
m_AddToSeg	♦						♦				
m_AnswerCall	♦				♦						
m_AutoEventOff	♦										
m_AutoEventOn	♦										
m_CallMsgSender	♦				♦	♦					
m_CloseFile	♦			♦							
m_CollectDigits	♦										

Header files / **Functions**	**m_acc.h**	**m_local.h**	**m_rm.h**	**m_file.h**	**m_voice.h**	**m_msg.h**	**m_seg.h**	**m_admin.h**	**m_event.h**	**m_lh.h**	**m_ens.h**
m_CommitFile	♦			♦							
m_ConferenceCall	♦				♦						
m_CopyFile	♦			♦							
m_CreateFile	♦			♦							
m_DeleteFile	♦			♦							
m_DeleteFromAddr	♦					♦					
m_DeleteFromPDL	♦							♦			
m_DeleteFromSeg	♦						♦				
m_DeleteGreeting	♦							♦			
m_DeletePDL	♦							♦			
m_DeletePersVerif	♦							♦			
m_DeleteSeg	♦						♦				
m_Deregister	♦	♦									
m_DeregisterAsMonitor	♦									♦	
m_DisconnectCall	♦				♦						
m_EventCheck	♦								♦		
m_FileExistCheck	♦			♦							
m_FilePattern	♦			♦							
m_ForwardMsg	♦					♦					
m_GenerateDTMF	♦										
m_GetCabinetInfo	♦			♦							
m_GetCallInfo	♦										
m_GetEnv	♦										
m_GetFileInfo	♦			♦							
m_GetLinkOM	♦									♦	
m_GetMboxStat	♦										♦

Header files / **Functions**	**m_acc.h**	**m_local.h**	**m_rm.h**	**m_file.h**	**m_voice.h**	**m_msg.h**	**m_seg.h**	**m_admin.h**	**m_event.h**	**m_lh.h**	**m_ens.h**
m_GetMsgCounter	♦					♦					
m_GetMsgNotification	♦					♦					
m_GetNumSegs	♦						♦				
m_GetSegInfo	♦						♦				
m_GetSegScript	♦						♦				
m_GetSysDate	♦		♦								
m_GetSysVersion	♦		♦								
m_GetVersion	♦	♦									
m_LinkSanity	♦									♦	
m_Logoff	♦		♦								
m_Logon	♦		♦								
m_MakeCall	♦				♦						
m_NormalizeSeg	♦						♦				
m_OnBRWarn	♦								♦		
m_OnCallProgress	♦								♦		
m_OnDigit	♦								♦		
m_OnError	♦								♦		
m_OnIncomingCall	♦								♦		
m_OnLhEvent	♦										
m_OnNewMessage	♦					♦			♦		
m_OnPlayEnd	♦								♦		
m_OnRecordEnd	♦								♦		
m_OnSessionEnd	♦								♦		
m_OnTimeout	♦								♦		
m_OpenFile	♦			♦							
m_OpenGreeting	♦							♦			

Functions \ Header files	m_acc.h	m_local.h	m_rm.h	m_file.h	m_voice.h	m_msg.h	m_seg.h	m_admin.h	m_event.h	m_lh.h	m_ens.h
m_OpenPDL	♦							♦			
m_OpenPersVerif	♦							♦			
m_PDLPattern	♦							♦			
m_PlayMsg	♦					♦					
m_PlayPrompt	♦										
m_PlaySegs	♦						♦				
m_PlayVoice	♦				♦						
m_PositionToSeg	♦						♦				
m_ReconnectCall	♦				♦						
m_RecordVoice	♦				♦						
m_Register	♦	♦									
m_RegisterAsMonitor	♦									♦	
m_Release	♦		♦								
m_ReleaseENS	♦										♦
m_RenameFile	♦			♦							
m_ReplyMsg	♦					♦					
m_ResetLinkOM	♦									♦	
m_RetrieveAddr	♦					♦					
m_RetrieveFile	♦			♦							
m_RetrievePDL	♦							♦			
m_RetrieveSeg	♦						♦				
m_SegPattern	♦						♦				
m_SenderAddr	♦					♦					
m_SendMsg	♦					♦					
m_SetEnv	♦										
m_SetFileSubject	♦			♦							

Header files / **Functions**	**m_acc.h**	**m_local.h**	**m_rm.h**	**m_file.h**	**m_voice.h**	**m_msg.h**	**m_seg.h**	**m_admin.h**	**m_event.h**	**m_lh.h**	**m_ens.h**
m_SetMboxEHN	♦										♦
m_SetMsgCounter	♦					♦					
m_SetMsgNotification	♦					♦					
m_SetSegInfo	♦						♦				
m_SetSegScript	♦						♦				
m_SetTimeout	♦	♦									
m_SkipVoice	♦				♦						
m_StartLink	♦									♦	
m_StopLink	♦									♦	
m_StopVoice	♦				♦						
m_TimeoutContinue	♦										
m_TimeoutOff	♦										
m_TimeoutOn	♦										
m_TransferCall	♦				♦						
m_TransferCallRevert	♦				♦						
m_UndeleteFile	♦			♦							
m_UndeleteSeg	♦						♦				
m_UserPassword	♦							♦			
m_WaitingForCall	♦		♦						♦		

Index

NORTEL

Reader's Response Form for

Meridian ACCESS
***Application Programming Interface (API) Reference Manual* (NTP 555-7001-317)**
August 1995

Tell us about yourself:

Name:	______________________	**Date:**	__________
Company:	______________________		
Address:	______________________		
	______________________		
Occupation:	______________________	**Phone:**	__________

1. What is your level of experience with this product?

   ❑ New user ❑ Intermediate ❑ Experienced ❑ Programmer

2. How do you use this book?

   ❑ Learning ❑ Procedural ❑ Problem solving ❑ Reference

3. Did this book meet all of your needs?

   ❑ Yes ❑ No

   If you answered **No** to this question, please answer the following questions.

4. What chapters, sections, or procedures did you find hard to understand?

   ____

   ____

   ____

5. What information (if any) was missing from this book?

   ____

   ____

   ____

6. How could we improve this book? (For example, books can also be evaluated in many other ways, including: ease of information retrieval, presentation, and use of reading aids, such as diagrams.)

   ____

Please return your comments by fax to (416) 597-7104, or mail your comments to: Customer Documentation, Nortel, 522 University Ave – 14th floor., Toronto, Ontario, Canada. M5G 1W7.

Reader's Response Form

Meridian

Meridian ACCESS

Application Programming Interface (API)
Reference Manual

Customer Documentation
Northern Telecom
522 University Avenue, 14th Floor
Toronto, Ontario, Canada,
M5G 1W7

Publication number:	555-7001-317
Product release:	10.0
Document release:	Standard 1.0
Date:	August 1995

NORTEL