MSAM Functions

This section describes the functions that you use to retrieve messages from and submit messages to the IPM Manager. Most functions handle messages of all types, but certain functions in the API are specific to letters or reports. Unless the function description refers to a specific message type, you should assume that the function handles all types of messages.

Functions whose names begin with *MSAMPut* apply to incoming messages; functions whose names begin with *MSAMGet* apply to outgoing messages. Functions whose names begin with *PMSAM* apply only to personal MSAMs; those whose names begin with *SMSAM* apply only to server MSAMs.

You must completely specify any structure that you provide to a function unless the description states otherwise.

All of the functions take a pointer to an `MSAMParam` parameter block as input. Each function description includes a list of the fields in the parameter block that are used by the function.

Most functions in the MSAM API have the following form:

```
pascal OSErr function (MSAMParam *paramBlock, Boolean asyncFlag);
```

You should call those functions asynchronously so that you can receive and process an AOCE high-level event at any time.

Some functions can be called only synchronously or asynchronously; therefore, they do not have the `asyncFlag` parameter. The form of those functions is:

```
pascal OSErr function (MSAMParam *paramBlock);
```

You can call a function from assembly language. Listing 2-16 illustrates one way to do this for a function that takes both the parameter block pointer and the Boolean value `asyncFlag` as parameters. (If a function can be called only synchronously or asynchronously, the assembly code would not manipulate the `asyncFlag` value.)

Listing 2-16 Calling an MSAM function from assembly language

```
_oceTBDispatch          OPWORD   $aa5e
   subq #2,a7               ; make room for function result
   movea paramBlock,-(sp)  ; push the param block pointer
                             onto stack
   move.q asyncFlag, d0    ; move async flag into D0
   move.b d0,-(sp)         ; push the flag (byte) onto stack
   moveq #opCode, d0       ; move op code into D0
   move.w d0,-(sp)         ; place the op code on the stack
   _oceTBDispatch          ; trap call
   move.w (sp)+, d0        ; get result code
```

The function returns its result code in the `ioResult` field of the parameter block.

When you call a function synchronously, the function returns its result both as the function result and in the `ioResult` field of the `MailParamBlockHeader` structure. Note that the function also clears the `ioCompletion` field.

When you call a function asynchronously and the function has successfully queued the request, it returns `noErr` and sets the `ioResult` field to 1. After the call completes, the function sets the `ioResult` field to the actual result and calls the completion routine, if one is specified. There is one exception to this behavior: if the IPM Manager is not currently ready to accept a request, it may return `corErr` as the function result. In this case, the `ioResult` field has an indeterminate value and the completion routine is not called.

IMPORTANT

If you choose to poll the `ioResult` field to determine if the request has completed, it is safest to check that it has changed from 1 to some other value. While the IPM Manager does not return positive error codes, system utilities may return positive error codes, and these may be passed through without being caught. Nominally, this would be due to an IPM Manager bug; however, you can and should attempt to protect against this. ▲

Initializing an MSAM

You use the routines in this section to initialize an MSAM. A personal MSAM begins by calling the `PMSAMGetMSAMRecord` function to obtain the creation ID of its record in the Setup catalog. Then it calls the `PMSAMOpenQueues` function for each of its slots to obtain the queue references for each slot. A server MSAM calls the `SMSAMSetup` function to obtain identifying information about itself and then calls the `SMSAMStartup` function to obtain its outgoing queue reference.

PMSAMGetMSAMRecord

The `PMSAMGetMSAMRecord` function provides you with the record creation ID of the record that represents your personal MSAM in the Setup catalog.

```
pascal OSErr PMSAMGetMSAMRecord (MSAMParam *paramBlock);
```

`paramBlock` Pointer to a parameter block.

Parameter block

←	`ioResult`	`OSErr`	Result code
←	`msamCID`	`CreationID`	Creation ID of personal MSAM record

See “The MSAM Parameter Block” on page 2-94 for a description of the `ioResult` field.

Field descriptions

`msamCID`	The creation ID of the record in the Setup catalog that represents your personal MSAM.

DESCRIPTION

You call the `PMSAMGetMSAMRecord` function to obtain the record creation ID of your personal MSAM's MSAM record in the Setup catalog.

The MSAM record contains a list of all the slots associated with the MSAM. In addition, your MSAM and its associated setup template may store private data that is global to the MSAM in the MSAM record.

The IPM Manager knows that a personal MSAM exists by its MSAM record in the Setup catalog.

IMPORTANT

The `PMSAMGetMSAMRecord` function is intended to be called only by a personal MSAM. Calling it from anywhere else yields indeterminate results. ▲

SPECIAL CONSIDERATIONS

This function is always executed synchronously.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0506

RESULT CODES

`noErr`	0	No error
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kOCEInvalidRef`	–1502	Invalid message reference number
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM
`kMailNoMSAMErr`	–15056	No such MSAM

SEE ALSO

The `CreationID` structure is described in the chapter "AOCE Utilities" in *Inside Macintosh: AOCE Application Interfaces*.

See the chapter "Service Access Module Setup" in this book for more information on the MSAM record in the Setup catalog.

PMSAMOpenQueues

The `PMSAMOpenQueues` function obtains the queue references for a slot that you specify.

```
pascal OSErr PMSAMOpenQueues (MSAMParam *paramBlock);
```

`paramBlock` Pointer to a parameter block.

Parameter block

←	`ioResult`	`OSErr`	Result code
←	`inQueueRef`	`MSAMQueueRef`	Incoming queue reference
←	`outQueueRef`	`MSAMQueueRef`	Outgoing queue reference
→	`msamSlotID`	`MSAMSlotID`	Address slot identification number

See “The MSAM Parameter Block” on page 2-94 for a description of the `ioResult` field.

Field descriptions

`inQueueRef`	If the slot you specify in the `msamSlotID` field is a mail slot, this value is the queue reference for the slot’s incoming queue. If the slot you specify in the `msamSlotID` field is a messaging slot, this value identifies the slot itself. (The `MSAMQueueRef` data type is `long`.)
`outQueueRef`	The queue reference for the outgoing queue of the slot you specify in the `msamSlotID` field. (The `MSAMQueueRef` data type is `long`.)
`msamSlotID`	The identification number of the slot for which you are requesting queue references. This number is the slot ID that you generated and stored in the slot’s record in the Setup catalog after receiving a `kMailEPPCCreateSlot` high-level event.

DESCRIPTION

A personal MSAM calls the `PMSAMOpenQueues` function to get the queue references associated with a slot. You need to provide the appropriate queue reference in subsequent operations.

Only mail slots have an incoming queue into which an MSAM places letters coming from an external messaging system that are addressed to the user. In the case of a messaging slot, the value in the `inQueueRef` field is a reference to the slot itself.

Typically, you call the function when starting up or after you respond to an `kMailEPPCCreateSlot` high-level event. On startup, you should call this function for every slot that you manage.

If you specify a suspended slot, the function returns a `kMailSlotSuspended` result code, but the queue references are still valid. (A slot is suspended when a personal MSAM calls the `PMSAMLogError` function to indicate a serious operational error associated with the slot.) In general, you should not attempt operations on a suspended slot.

If you specify an inactive slot (if the `active` field in the `MailStandardSlotInfoAttribute` structure is set to `false`), the queue references are valid. However, in general, you should not attempt operations on an inactive slot.

After you respond with a `noErr` result to the `kMailEPPCCreateSlot` high-level event, it is possible that the IPM Manager will encounter an error instantiating the new slot. If this happens, when you call the `PMSAMOpenQueues` function to obtain the new slot's queue references, the function returns a `kMailNoSuchSlot` result code.

Queue references remain valid as long as the slot is not deleted and the Macintosh remains running. The conservative approach is to call the function each time your personal MSAM starts up.

IMPORTANT

The `PMSAMOpenQueues` function is intended to be called only by a personal MSAM. Calling it from anywhere else yields indeterminate results. ▲

SPECIAL CONSIDERATIONS

There is a very small period immediately after you respond to a `kMailEPPCCreateSlot` high-level event during which the `PMSAMOpenQueues` function returns a `kMailNoSuchSlot` result code even if no error occurred. You should call the function periodically until it completes successfully.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0500

RESULT CODES

`noErr`	0	No error
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kOCEInvalidRef`	–1502	Invalid message reference number
`kOCEInternalErr`	–1506	Serious internal error
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM
`kMailNoMSAMErr`	–15056	No such personal MSAM
`kMailNoSuchSlot`	–15062	No such slot
`kMailBadMSAM`	–15066	MSAM unusable for unspecified reason

SEE ALSO

See the description of the `kMailEPPCCreateSlot` high-level event on page 2-221 for more information about slot IDs.

SMSAMSetup

The `SMSAMSetup` function creates the MSAM's Forwarder record.

```
pascal OSErr SMSAMSetup (MSAMParam *paramBlock);
```

`paramBlock` Pointer to a parameter block.

Parameter block

←	`ioResult`	`OSErr`	Result code
↔	`serverMSAM`	`RecordIDPtr`	Server MSAM's record ID pointer
→	`password`	`RStringPtr`	Pointer to server MSAM's password
→	`gatewayType`	`OSType`	Server MSAM's extension type
→	`gatewayTypeDescription`	`RStringPtr`	Description of extension type
←	`catalogServerHint`	`AddrBlock`	Catalog server address

See "The MSAM Parameter Block" on page 2-94 for a description of the `ioResult` field.

Field descriptions

`serverMSAM` A pointer to the record ID of the server MSAM's Forwarder record. Set the `recordName` field to the name of the server MSAM and the `recordType` field to the constant `kMnMForwarderRecTypeNum`. The function returns the Forwarder record's creation ID in the `cid` field and the record location information.

`password` A pointer to the server MSAM's password string.

`gatewayType` The MSAM's 4-character extension type.

`gatewayTypeDescription`
A pointer to an `RString` containing a user-readable description of the MSAM type. For example, an AppleLink MSAM whose type is `'ALNK'` might provide the string "AppleLink".

`catalogServerHint`
The AppleTalk address of the PowerShare catalog server that created the MSAM's Forwarder record. The MSAM can later pass this value to a Catalog Manager function (in the `serverHint` field of the function's parameter block) if it wants to direct the request to that particular catalog server.

DESCRIPTION

You call the `SMSAMSetup` function as part of a server MSAM's initialization process. The function creates the MSAM's Forwarder record. Before calling the function, you need to obtain from the system administrator the server MSAM's name and password, its extension type, and a string describing the extension type. (A server MSAM may also have built-in knowledge of its extension type.) When the function completes successfully, you should save knowledge of the fact that the function completed successfully in your preferences file in the Preferences folder so that you do not call the function again after a subsequent launch.

SPECIAL CONSIDERATIONS

After calling the `SMSAMSetup` function, call the `SMSAMStartup` function to get the server MSAM's queue reference.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0523

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	All allocation blocks on the volume are full
`kOCEParamErr`	–50	Invalid parameter
`kOCEWriteAccessDenied`	–1541	Identity lacks write access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can't find specified dNode
`kOCENoDupAllowed`	–1641	Duplicate name and type

SEE ALSO

For a description of the server MSAM initialization process, see "Initializing a Server MSAM" beginning on page 2-40.

The `SMSAMStartup` function is described next.

SMSAMStartup

The `SMSAMStartup` function informs a PowerShare mail server that the server MSAM that you specify has started up.

```
pascal OSErr SMSAMStartup (MSAMParam *paramBlock);
```

`paramBlock` Pointer to a parameter block.

Parameter block

←	`ioResult`	`OSErr`	Result code
→	`msamIdentity`	`AuthIdentity`	Server MSAM identifier
←	`queueRef`	`MSAMQueueRef`	Queue reference

See "The MSAM Parameter Block" on page 2-94 for a description of the `ioResult` field.

Field descriptions

`msamIdentity`	The server MSAM's authentication identity. You obtain this identity from the `AuthBindSpecificIdentity` function.
`queueRef`	A value that identifies the outgoing queue for the server MSAM that you specify.

DESCRIPTION

You call the `SMSAMStartup` function to inform the PowerShare mail server that a server MSAM is active and that the PowerShare mail server can send the MSAM high-level events and request status information. You must call this function every time your server MSAM starts up.

The function returns a queue reference for the server MSAM's outgoing queue. You provide the queue reference to the `MSAMOpen` function when you want to open an outgoing message. In addition, you provide the queue reference to the `MSAMCreate` function when you want to create an incoming message. In that situation, the queue reference identifies the MSAM itself.

You must have successfully called the `SMSAMSetup` function to create the MSAM's Forwarder record before you call the `SMSAMStartup` function. Otherwise, `SMSAMStartup` returns the `kMailNoSuchSlot` result code.

The queue reference is valid until the server MSAM's PowerShare mail server quits. You know that the PowerShare mail server is not running when any of the MSAM API functions return the `corErr` result code. When the PowerShare mail server starts up again, you need to call the `SMSAMStartup` function again to get a new queue reference.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0501

RESULT CODES

`noErr`	0	No error
`corErr`	–3	PowerShare mail server not running
`memFullErr`	–108	Not enough memory
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kMailNoSuchSlot`	–15062	Unknown server MSAM

SEE ALSO

The `AuthBindSpecificIdentity` function and authentication identities are discussed in the chapter "Authentication Manager" in *Inside Macintosh: AOCE Application Interfaces*.

The `SMSAMSetup` function is described on page 2-135.

The AppleTalk Transition Queue is described in the chapter "Link-Access Protocol (LAP) Manager" in *Inside Macintosh: Networking*.

A server MSAM's initialization process is described in the section "Initializing a Server MSAM" beginning on page 2-40.

Enumerating Messages in a Queue

Both personal and server MSAMs can use the `MSAMEnumerate` function to list messages in an outgoing queue. Personal MSAMs can also use the function to list letters in an incoming queue.

MSAMEnumerate

The `MSAMEnumerate` function returns information about the messages in a queue that you specify.

```
pascal OSErr MSAMEnumerate (MSAMParam *paramBlock,
                            Boolean asyncFlag);
```

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`queueRef`	`MSAMQueueRef`	Queue reference number
→	`startSeqNum`	`long`	Starting message
←	`nextSeqNum`	`long`	Message to continue next enumeration
↔	`buffer`	`MailBuffer`	Your buffer structure

See “The MSAM Parameter Block” on page 2-94 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`queueRef`	The value that identifies the queue about which you want information. A personal MSAM specifies either the outgoing queue reference or the incoming queue reference that it obtained from the `PMSAMOpenQueues` function, depending on which queue it wants to enumerate. A server MSAM specifies the outgoing queue reference that it obtained from the `SMSAMStartup` function.
`startSeqNum`	The sequence number of the message in the queue at which you want the `MSAMEnumerate` function to start the enumeration. Set this field to 1 to begin the enumeration with the first message in the queue. When you call the function and there is insufficient space in your buffer to hold information about all of the remaining messages in the queue, the function returns in the `nextSeqNum` field the sequence number of the next message. Use that number in the `startSeqNum` field the next time you call the function.

`nextSeqNum`	The sequence number of the first message in the queue whose information did not fit into your buffer. The function sets this field when your buffer is too small to hold all the information you requested. To continue the enumeration, call the `MSAMEnumerate` function again and set the `startSeqNum` field to the current value of the `nextSeqNum` field. The `MSAMEnumerate` function sets the `nextSeqNum` field to 0 when it has returned information about all of the messages in the queue.
`buffer`	A `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. Because the number of messages in the queue varies, use your best estimate to choose the size of the buffer. The `MSAMEnumerate` function retrieves information about the messages in the queue that you specify and writes it into your buffer, the `buffer` field. It sets the value of the `dataSize` field to the number of bytes of data it placed in the buffer.

DESCRIPTION

You call the `MSAMEnumerate` function to obtain information about messages in a queue that you specify. The function stores this information in a buffer that you provide. If your buffer is not large enough to hold all of the information, you can call this function repeatedly. When the function sets the `nextSeqNum` field to 0, you have retrieved information on all of the messages in the queue.

Both personal and server MSAMs can enumerate an outgoing queue. When an MSAM enumerates an outgoing queue, the function returns information about all of the messages in the queue, including letters and non-letter messages.

Only a personal MSAM can enumerate an incoming queue to get information about the letters in the queue because incoming queues are specific to personal MSAMs.

No matter which type of queue you enumerate, the function places the data in your buffer in the form of a `MailReply` structure. The first 2 bytes contain a count of the total number of structures that follow it in the buffer. The structures that follow are either `MSAMEnumerateOutQReply` (if you enumerate an outgoing queue) or `MSAMEnumerateInQReply` structures (if you enumerate an incoming queue). See the descriptions of the `MSAMEnumerateInQReply` and `MSAMEnumerateOutQReply` structures, respectively, for information on what specific data you retrieve when you enumerate an incoming or an outgoing queue.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0503

RESULT CODES

`noErr`	0	No error
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kOCEInvalidRef`	–1502	Invalid queue reference
`kOCEBufferTooSmall`	–1503	Buffer is too small
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM, or server MSAM's mail server is shutting down

SEE ALSO

The `MSAMEnumerateOutQReply` structure is described on page 2-97.

The `MSAMEnumerateInQReply` structure is described on page 2-98.

The `MailReply` structure is described on page 2-97.

The `MailBuffer` structure is described on page 2-96.

Opening an Outgoing Message

Call the `MSAMOpen` function to open a message in an outgoing queue. Once a message is open, you can read its contents.

MSAMOpen

The `MSAMOpen` function opens a message in an outgoing queue.

```
pascal OSErr MSAMOpen (MSAMParam *paramBlock
                        Boolean asyncFlag);
```

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`queueRef`	`MSAMQueueRef`	Queue reference number
→	`seqNum`	`long`	Sequence number of message in queue
←	`mailMsgRef`	`MailMsgRef`	Message reference number

See “The MSAM Parameter Block” on page 2-94 for descriptions of the `ioResult` and `ioCompletion` fields.

Field descriptions

`queueRef`	The queue reference of the queue that contains the message you want to open. For a personal MSAM, specify the outgoing queue reference you obtained from the `PMSAMOpenQueues` function. For a server MSAM, specify the queue reference you obtained from the `SMSAMStartup` function.
`seqNum`	The sequence number that identifies the message you want to open. You get this number from the `seqNum` field in the `MSAMEnumerateOutQReply` structure returned by the `MSAMEnumerate` function.
`mailMsgRef`	A message reference number that identifies the opened message. The `MSAMOpen` function returns a reference number for the message that you use in subsequent function calls to read the message.

DESCRIPTION

You call the `MSAMOpen` function to open a message in the outgoing queue you specify in the `queueRef` field.

The `MSAMOpen` function provides a unique message reference number to each MSAM that opens a given message. Once you close the message by calling the `MSAMClose` function, the message reference number becomes invalid and you cannot use it in subsequent function calls. (In contrast, the value of the `seqNum` field is a reference to the message that remains valid until you delete the message by calling the `MSAMDelete` function.)

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0508

RESULT CODES

`noErr`	0	No error
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kOCEInvalidRef`	–1502	Invalid queue reference
`kOCEDoesntExist`	–1511	No such letter
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM, or server MSAM's mail server is shutting down

SEE ALSO

The `MSAMEnumerateOutQReply` structure is described on page 2-97.

The `PMSAMOpenQueues` function is described on page 2-133.

The `SMSAMStartup` function is described on page 2-136.

The `MSAMClose` function is described on page 2-167.

The `MSAMDelete` function is described on page 2-202.

Reading Header Information

To read letter attributes from an open letter, use the `MSAMGetAttributes` function. You can read the recipients of a message with the `MSAMGetRecipients` function. To read the header of a non-letter message, use the `MSAMGetMsgHeader` function.

MSAMGetAttributes

The `MSAMGetAttributes` function reads attributes from the header of an open letter that you specify.

```
pascal OSErr MSAMGetAttributes (MSAMParam *paramBlock,
                                Boolean asyncFlag);
```

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`mailMsgRef`	`MailMsgRef`	Letter reference number
→	`requestMask`	`MailAttributeBitmap`	Attribute types requested
↔	`buffer`	`MailBuffer`	Your buffer structure
←	`responseMask`	`MailAttributeBitmap`	Attribute types returned
←	`more`	`Boolean`	Is there more data?

See “The MSAM Parameter Block” on page 2-94 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`mailMsgRef`	A reference number that identifies the letter whose attributes you want to read. You obtain the reference number when you call the `MSAMOpen` function.
`requestMask`	A bit field structure that specifies which attributes in the letter’s header you want to read. The attributes whose values you may retrieve with this function are listed below. Set the bit for each attribute that you want to read. Clear the remaining bits.
`buffer`	A `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. The `MSAMGetAttributes` function writes attribute values into your buffer (the `buffer` field) and sets the value of the `dataSize` field to the number of bytes of data it placed in the buffer.

`responseMask`	A bit field structure that specifies the attributes for which the `MSAMGetAttributes` function returned values in the buffer. If the function did not return an attribute because either a requested attribute does not exist in the letter or you did not request the attribute, the function sets the corresponding bit in the structure to 0.
`more`	A Boolean value that indicates whether there are more attribute values than can fit in your buffer. If your buffer is too small to hold all of the attribute values that you requested, the `MSAMGetAttributes` function sets this field to `true`; otherwise, it sets this field to `false`. If the value of the `more` field is `true`, you can call the `MSAMGetAttributes` function again, setting the bits in the request mask for the attributes you did not yet receive.

DESCRIPTION

You call the `MSAMGetAttributes` function to read letter attributes by setting the appropriate bits in the `requestMask` field. You can request any combination of the following attributes:

Letter attribute	**Bit constant**	**Mask constant**
Indications	`kMailIndicationsBit`	`kMailIndicationsMask`
Letter creator & type	`kMailMsgTypeBit`	`kMailMsgTypeMask`
Letter ID	`kMailLetterIDBit`	`kMailLetterIDMask`
Send timestamp	`kMailSendTimeStampBit`	`kMailSendTimeStampMask`
Nesting level	`kMailNestingLevelBit`	`kMailNestingLevelBMask`
Message family	`kMailMsgFamilyBit`	`kMailMsgFamilyMask`
Reply ID	`kMailReplyIDBit`	`kMailReplyIDMask`
Conversation ID	`kMailConversationIDBit`	`kMailConversationIDMask`
Subject	`kMailSubjectBit`	`kMailSubjectMask`

The `MSAMGetAttributes` function reads the attribute values you requested from the letter header and writes them into your buffer, starting with the attribute specified by the least significant bit in the `requestMask` field and continuing in ascending order. If the length of an attribute value is odd, it adds a pad byte so that each attribute value starts on an even boundary.

You can request attributes for any letter you have previously opened.

You cannot read a letter's `to`, `from`, `cc`, or `bcc` attributes by calling the `MSAMGetAttributes` function. Call the `MSAMGetRecipients` function for this purpose. The `MSAMGetAttributes` function ignores the bits in the request mask that correspond to recipient attributes and sets the equivalent bits in the response mask to 0 to indicate that it is not returning the values for these attributes. The `MSAMGetAttributes` function does not return an error in this case.

SPECIAL CONSIDERATIONS

Because the `MailAttributeBitmap` data type is defined as a bit field structure, you cannot use the predefined masks such as `kMailSubjectMask`, `kMailMsgTypeMask`, and so forth to set or test the value of a bit field in the `requestMask` or `responseMask` field. The masks operate on variables of type `long`.

You cannot read a letter's `letterFlags` attribute by calling the `MSAMGetAttributes` function. Only incoming letters have that attribute.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$050B

RESULT CODES

`noErr`	0	No error
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kOCEInvalidRef`	–1502	Invalid queue reference

SEE ALSO

The `MailAttributeBitmap` structure, including the complete list of letter attributes, is described on page 2-100.

The `MailBuffer` structure is described on page 2-96.

The `MSAMGetRecipients` function is described next.

See the section "Reading Letter Attributes" beginning on page 2-47 for an example of reading attributes from a letter header.

MSAMGetRecipients

The `MSAMGetRecipients` function returns recipient information from the header of an open message that you specify.

```
pascal OSErr MSAMGetRecipients (MSAMParam *paramBlock,
                                Boolean asyncFlag);
```

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`mailMsgRef`	`MailMsgRef`	Message reference number
→	`attrID`	`MailAttributeID`	Recipient type requested
→	`startIndex`	`unsigned short`	Recipient to start from
↔	`buffer`	`MailBuffer`	Your buffer structure
←	`nextIndex`	`unsigned short`	Recipient to continue from next time
←	`more`	`Boolean`	Is there more data?

See "The MSAM Parameter Block" on page 2-94 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`mailMsgRef`	A reference number that identifies the message about which you want recipient information. You obtain the reference number when you call the `MSAMOpen` function.
`attrID`	A constant that identifies the type of recipient about which you want information. Specify `kMailResolvedList` if you want information about resolved recipients. If you want information about an original recipient type, specify `kMailFromBit`, `kMailToBit`, `kMailCcBit`, or `kMailBccBit`.You can specify one type of recipient each time you call the `MSAMGetRecipients` function.
`startIndex`	The position in the recipient list at which you want the `MSAMGetRecipients` function to begin extracting information to store in your buffer. Set this field to 1 to start with the first recipient of the type specified by the `attrID` field.
`buffer`	A `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. The `MSAMGetRecipients` function writes recipient information into your buffer (the `buffer` field) and sets the value of the `dataSize` field to the number of bytes of data it placed in the buffer. The function places the data in your buffer in the form of a `MailReply` structure. The first 2 bytes contain a count of the number of recipient structures that follow in your buffer. If you request information about an original recipient type (`to`, `cc`, `bcc`, `from`), the `MSAMGetRecipients` function returns the recipient information as one or more `MailOriginalRecipient` structures. If you request information about resolved recipients, the function returns the information as one or more `MailResolvedRecipient` structures. If a recipient structure has an odd length, the function adds a pad byte so that the next structure can start on a word boundary.
`nextIndex`	If the value of the `more` field is `true`, the `nextIndex` field indicates the position in the recipient list of the first attribute that did not fit into your buffer. If the value of the `more` field is `false`, the `nextIndex` field is undefined.

`more`	A Boolean value that indicates whether there is more recipient information than can fit in your buffer. If your buffer is too small to hold all of the recipient information that you requested, the `MSAMGetRecipients` function sets this field to `true`; otherwise, it sets this field to `false`. If the function sets this field to `true`, you can call it again to retrieve additional information by setting the `startIndex` field for the next call to the value of the `nextIndex` field.

DESCRIPTION

You call the `MSAMGetRecipients` function to get a list of original or resolved recipients for the message that you specify in the `mailMsgRef` field. You need to get original recipients so that you can properly display them as From, To, cc, or bcc recipients in the message you send to an external messaging system. You need to get a list of resolved recipients so that you know to which recipients you must send the message.

By setting the `attrID` field appropriately, you can specify either a resolved recipient or one type of original recipient each time you call the `MSAMGetRecipients` function.

If you specify an original recipient type in the `attrID` field, the function returns data in the form of one or more `MailOriginalRecipient` structures. Each of these structures contains the absolute index of the recipient followed immediately by information about one recipient. The absolute index is useful if you need to match an original recipient with the corresponding resolved recipient.

If you specify a resolved recipient in the `attrID` field, the function returns data in the form of one or more `MailResolvedRecipient` structures. Each of these structures contains the absolute index of the recipient, the Boolean variable `responsible`, and recipient flags, followed immediately by information about one recipient. If the value of the `responsible` field is `true`, you are responsible for delivering the message to that recipient and submitting delivery and non-delivery reports to the sender if those are requested. Naturally, you should not attempt to deliver a message to a recipient for which the `responsible` field is set to `false`. If the `kIPMBCCRecBit` bit in the `recipientFlags` field is set, the recipient is a bcc recipient.

Note
A From recipient may appear in the resolved list, but in that case the `responsible` field is always set to `false`. ◆

As you read `MailResolvedRecipient` structures from your buffer, you must save the ordinal-position value for each resolved recipient. The first recipient's ordinal-position value is 1; the second recipient's ordinal-position value is 2; the *nth* recipient's ordinal-position value is *n*, and so forth. The `MSAMnMarkRecipients` function requires you to provide the ordinal-position value to identify a recipient you want to mark. If you need to call `MSAMGetRecipients` more than once to get all of the resolved recipients, do not set the ordinal-position value back to 0 on successive calls to the function. Rather, increment the ordinal-position value continuously across multiple calls to the `MSAMGetRecipients` function for a given letter so that each resolved recipient is associated with a unique ordinal-position value.

Personal MSAMs will find a one-to-one correspondence between their resolved recipients and their displayable (original) recipients because all group addresses are expanded into individual recipients before the `MSAMGetRecipients` function returns recipient information to the personal MSAM.

Server MSAMs may find they have more resolved recipients than original recipients. This is because the PowerShare mail server expands PowerShare group addresses into individual addresses when you ask for resolved recipients. However, it does not necessarily expand PowerShare group addresses when you ask for original recipients. The `MSAMGetRecipients` function does not expand any external group addresses.

Server MSAMs may also find that there are resolved recipients that are not exactly the same as the corresponding original recipients. These have been resolved by the AOCE software to a more specific form.

The PowerShare mail server does not suppress duplicate external addresses. Sometimes it suppresses duplicate addresses resulting from the expansion of a PowerShare group address. However, you are not guaranteed that the `MSAMGetRecipients` function will not return duplicate addresses.

SPECIAL CONSIDERATIONS

For non-letter messages, the From recipient is a reply queue address, a return address that is not necessarily the same as the sender’s address.

This function does not apply to delivery and non-delivery reports. You cannot read the recipient attribute of a report.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$050C

RESULT CODES

`noErr`	0	No error
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kOCEInvalidRef`	–1502	Invalid message reference number
`kOCEBufferTooSmall`	–1503	Buffer is too small

SEE ALSO

The `MailOriginalRecipient` structure is described on page 2-108.

The `MailResolvedRecipient` structure is described on page 2-108.

Original and resolved recipients are discussed in the section “Reading Addresses” beginning on page 2-51.

The `MailBuffer` structure is described on page 2-96.

The `MailReply` structure is described on page 2-97.

Reply queues are discussed with the `MSAMPutMsgHeader` function on page 2-183.

The `MSAMnMarkRecipients` function is described on page 2-163.

MSAMGetMsgHeader

The `MSAMGetMsgHeader` function reads data from the header of a non-letter message that you specify.

```
pascal OSErr MSAMGetMsgHeader (MSAMParam *paramBlock,
                               Boolean asyncFlag);
```

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`mailMsgRef`	`MailMsgRef`	Message reference number
→	`selector`	`IPMHeaderSelector`	Type of header data requested
→	`offset`	`unsigned long`	Begin reading from here
↔	`buffer`	`MailBuffer`	Your buffer
←	`remaining`	`unsigned long`	Number of bytes still to read

See “The MSAM Parameter Block” on page 2-94 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`mailMsgRef`	A reference number that identifies the message for which you want header information. You obtain the reference number when you call the `MSAMOpen` function.
`selector`	A constant that indicates the type of header information that you are requesting. The possible values are defined below. You cannot add or combine constant values in the `selector` field.
`offset`	The byte position, relative to the beginning of the header information specified in the `selector` field, from which you want the `MSAMGetMsgHeader` function to begin reading. To read from the beginning of the header information field, set this field to 0. If your buffer is too small to hold all of the data you requested, you can call the `MSAMGetMsgHeader` function again and compute a new value for the `offset` field using the `dataSize` value that the function returns in the `MailBuffer` structure.

`buffer`	A `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. The `MSAMGetMsgHeader` function writes header information into your buffer and sets the value of the `dataSize` field to the number of bytes of data it placed in the buffer.
`remaining`	The number of bytes of data remaining to be read. The `MSAMGetMsgHeader` function sets this field to 0 when it has returned all of the information that you requested.

DESCRIPTION

You call the `MSAMGetMsgHeader` function to obtain information from the header of a non-letter message. Do not call this function to read headers of letters or reports.

If the buffer you provide is not large enough to hold the information requested, you must make additional calls to the `MSAMGetMsgHeader` function to obtain it.

The format of the information that the `MSAMGetMsgHeader` function places in your buffer varies according to the value of the `selector` field. You may use any of the following constants in the `selector` field:

Selector value	**Description**
`kIPMTOC`	The function returns an array of `TOC` structures, one for each block in the message. Each entry in the array contains the block's size, creator, type, offset, and up to 4 bytes of private data that the application that created the block may have added for its own purposes when it created the block. The array of `TOC` structures is ordered; the sequential position of a block entry in the table of contents is a message block's index. The index of the first block is 1. You can identify a message block by its index number.
`kIPMSender`	The function returns the identity of the sender of the message in an `IPMSender` structure.
`kIPMProcessHint`	The function returns a Pascal string of up to 32 characters. The application that created the message may add a string for its own purposes when it creates the message.
`kIPMMessageTitle`	The function returns the title of the message in an `RString` structure.
`kIPMMessageType`	The function returns the creator and type of the message in an `IPMMsgType` structure.
`kIPMFixedInfo`	The function returns selected header information in an `IPMFixedHdrInfo` structure.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0511

RESULT CODES

`noErr`	0	No error
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kOCEInvalidRef`	–1502	Invalid message reference number
`kOCEBufferTooSmall`	–1503	Buffer is too small

SEE ALSO

The `TOC`, `IPMSender`, `IPMFixedHdrInfo`, and `IPMMsgType` structures are described in the chapter “Interprogram Messaging Manager” in *Inside Macintosh: AOCE Application Interfaces*.

The `MSAMGetMsgHeader` function is virtually identical to the `IPMReadMsgHeader` function. An application creating a message adds the process hint Pascal string when it calls the `IPMNewMsg` function and the private data in a message block when it calls the `IPMNewBlock` function. All of these functions are described in the chapter “Interprogram Messaging Manager” in *Inside Macintosh: AOCE Application Interfaces*.

The `RString` structure is described in the chapter “AOCE Utilities” in *Inside Macintosh: AOCE Application Interfaces*.

The `MailBuffer` structure is described on page 2-96.

Reading a Message

The MSAM API provides a number of functions to read outgoing messages that have been opened. The functions `MSAMGetContent` and `MSAMGetEnclosure` apply only to letters. The `MSAMEnumerateBlocks`, `MSAMGetBlock`, and `MSAMOpenNested` functions apply to any type of message.

MSAMGetContent

The `MSAMGetContent` function returns information about (and if requested, data from) a single segment in a letter’s content block.

```
pascal OSErr MSAMGetContent (MSAMParam *paramBlock,
                             Boolean asyncFlag);
```

`paramBlock` Pointer to a parameter block.

`asyncFlag` A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`mailMsgRef`	`MailMsgRef`	Letter reference number
→	`segmentMask`	`MailSegmentMask`	Segment type you want to read
↔	`buffer`	`MailBuffer`	Your buffer structure
↔	`textScrap`	`StScrpRec*`	Pointer to style scrap structure
←	`script`	`ScriptCode`	Character set
←	`segmentType`	`MailSegmentType`	Segment type returned
←	`endOfScript`	`Boolean`	End of data of one character set?
←	`endOfSegment`	`Boolean`	End of segment data?
←	`endOfContent`	`Boolean`	End of letter content?
←	`segmentLength`	`long`	Length of segment
↔	`segmentID`	`long`	Segment identifier

See “The MSAM Parameter Block” on page 2-94 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`mailMsgRef` A reference number that identifies the letter whose content you want to read. You obtain the reference number when you call the `MSAMOpen` function.

`segmentMask` The types of segments that you want to read. The content of a letter consists of text, pictures, sound, QuickTime movies, and styled text segments. The constants that you use to specify the segment types you want are described on page 2-110.

You can request any combination of segment types in the same request except text and styled text segments. If you request styled text segments, the function returns both plain text and styled text segments. If you request plain text segments, it returns any plain text segments that are in the letter and also converts styled text segments to plain text segments and returns them to you.

`buffer` A `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. If the current segment is one of the types that you specified in the segment mask, the `MSAMGetContent` function writes the segment into your buffer and sets the value of the `dataSize` field to the number of bytes of data it placed in the buffer.

`textScrap` A pointer to a style scrap structure (`StScrpRec`). If you request styled text segments, you can choose to allocate the structure, depending on which of two methods you want to use to read styled text. Both methods are described in the discussion below.

If you choose to allocate the style scrap structure, set its `scrpNStyles` field to the number of styles your buffer can hold. When the function writes styled text to your buffer, it returns style information in the style scrap structure and sets the `scrpNStyles` field to the actual number of styles returned.

If you are not requesting styled text segments, the function ignores this field.

`script`	A value that indicates the character set (Roman, Arabic, Kanji, etc.) of the text that the function placed in your buffer. The function sets this field only when it returns text data (it sets the `segmentType` field to `kMailTextSegmentType` or `kMailStyledTextSegmentType`).
`segmentType`	A constant that indicates the type of the current data segment. A segment can contain text, pictures, sound, QuickTime movies, or styled text. The constants that the function may return in this field are described on page 2-109. (If you are reading data from the segment and you need to call the `MSAMGetContent` function more than once to retrieve all of the data from the segment, the function returns a value in this field only the first time you call it for that segment.)
`endOfScript`	A Boolean value that indicates whether the text in your buffer is the end of a script run. The function sets this flag only when it returns data from a plain text or styled text segment. If there is more text in the current script run, it sets this field to `false`.
`endOfSegment`	A Boolean value that indicates whether the `MSAMGetContent` function has reached the end of a segment. If you did not request the current segment type in your segment mask, the function always sets this field to `true`. If you requested the current segment type in your segment mask, the function sets this field to `true` if it has returned all of the data in the current segment and to `false` if there is more data in the current segment.
`endOfContent`	A Boolean value that indicates whether the `MSAMGetContent` function has reached the end of the letter's content block. The `MSAMGetContent` function sets the `endOfContent` field to `true` when it reaches the end of the last segment in the content block; otherwise it sets this field to `false`.
`segmentLength`	The number of bytes in the current segment. The `MSAMGetContent` function returns a value in this field the first time you call it for a given segment.
`segmentID`	A segment identifier. This is both an input and an output. Set this field to 0 the first time you call it for a given letter. The function returns a value in this field the first time it reads each segment in a letter. On subsequent calls to the function, you set it to 0 or to a known segment ID. If you set it to 0, the function continues reading sequentially the current segment (or if `endOfSegment` is set to `true`, the next segment). If you set it to a segment ID, the function reads the segment specified by the segment ID.

DESCRIPTION

The `MSAMGetContent` function returns information about a single segment in a letter's content block each time you call it. If the current segment type is one that you specified in your segment mask, the function also returns actual segment data from the segment. You must previously have opened the letter by calling either the `MSAMOpen` or `MSAMOpenNested` function.

A content block contains a series of segments in standard interchange format; that is, each segment consists of either text, pictures, sound, styled text, or QuickTime movies. You tell the `MSAMGetContent` function what types of segments you want to read by setting the `segmentMask` field appropriately. The function examines the value of the `segmentMask` field the first time you call it for a given letter and at the beginning of each segment in the letter to determine whether it should write the segment data into the buffer that you provide.

At the beginning of each segment, the `MSAMGetContent` function sets the `segmentType`, `segmentLength`, `segmentID`, `endOfSegment`, and `endOfContent` fields. You can detect a new segment by examining the `endOfSegment` flag: if its value is `true` you know that you will get information on a new segment the next time you call the `MSAMGetContent` function.

You can read the segments in a letter's content block in sequential order or in any order you wish, depending on the value you specify for the segment ID. To read segments in the order they are stored in the content block, specify 0 in the `segmentID` field. The first time it reads a given segment, the function returns the segment ID. Because it is both an input and output value, be sure to clear the `segmentID` field after the start of a new segment to continue reading segments sequentially. If you do not set the `segmentID` field to 0, you will read the same segment over and over again.

To read segments in random order, you must know the segment's segment ID. Provide the ID in the `segmentID` field to access the segment randomly. When you specify a segment ID other than 0, the function repositions the offset at which it begins reading to the start of the segment you identify.

Note
To build a table of contents of segments, their segment types, their lengths, and their segment IDs, set the `segmentMask` field to 0 and call the `MSAMGetContent` function repeatedly until the `endOfContent` field returns `true`. ◆

There are two types of text data: plain text and styled text. If you request styled text segments, the function returns both plain text and styled text segments. If you request plain text segments, it returns any plain text segments that are in the letter and also converts styled text segments to plain text segments and returns them to you.

A text segment contains one or more script runs. A script run is a string of text in the same character set. When the function returns text data (that is, when the function sets the `segmentType` field to `kMailTextSegmentType` or `kMailStyledTextSegmentType`), the `script` field indicates the character set. The function identifies the end of a script run by setting the `endOfScript` field to `true`.

When you request plain text (that is, when you specify `kMailTextSegmentMask` in your segment mask), the `MSAMGetContent` function retrieves styled text as plain text. You lose all style information when you do this (except for the character set specified in the `script` field).

A styled text segment consists not of a stream of bytes but rather of a series of “style runs” akin to style runs in TextEdit.

To read a styled text segment, you allocate a style scrap structure and set the `textScrap` field to point to it. You should allocate a `StScrpRec` structure of a size appropriate to your MSAM. The function places the text into your buffer and the style information into the style scrap structure. It sets the `scrpStartChar` field in each `ScrpSTElement` structure in the style scrap structure to the offset of the text to which it applies, relative to the start of your buffer. The function completes when it has returned all the styled text or when it runs out of room for either the style information or text. If additional styled text exists, it sets `endOfSegment` to `false`.

If the function completes because it runs out of room for either the style information or the text, then the next time you call the function, it continues writing text from the same segment into your buffer and putting text styles in your style scrap structure. In this case, the offsets in the `scrpStartChar` field of the `ScrpSTElement` structure in your style scrap structure apply only to the data currently in your buffer, not to the offsets in the original segment in the letter.

For example, suppose that the next segment in the letter to be read is a styled text segment 120 bytes in length containing 12 different styles. The eleventh style starts at an offset of 90 (that is, at the 91st byte of the segment). Suppose further that your text buffer is 200 bytes but your style scrap structure can hold only 10 styles. In this case, the `MSAMGetContent` function stops writing data to your buffer after it has placed 10 styles in your style scrap structure. Because these 10 styles applied to the first 90 bytes of text, the `dataSize` field of your `MailBuffer` structure indicates that 90 bytes of data were written to your buffer, and the value of the `endOfSegment` field is `false`.

The next time you call the function, it writes the last 30 bytes of text into your buffer and puts the last two styles into your style scrap structure. It returns a value of 2 in the `scrpNStyles` field of your style scrap structure and sets the `endOfSegment` field to `true`. In this case, the first offset in the `scrpStartChar` field of the script table in the style scrap structure is 0, indicating that the first style in the text scrap starts with the first byte of text currently in your buffer. (The offset is *not* 90, as it would have been for this portion of text had your style scrap structure been able to hold all of the styles at once.)

You cannot specify `kMailTextSegmentMask` and `kMailStyledTextSegmentMask` at the same time.

SPECIAL CONSIDERATIONS

Different Macintosh computers may use the same font number for different fonts. That is, font numbers may vary from computer to computer, but font names are supposed to be unique. The `SMPAddContent` function in the Standard Mail Package creates a block containing a table that maps font numbers to font names. To ensure that you apply the right fonts to styled text, you need to read this font block. Its block creator is `'fish'` and its block type is `'font'`.

You can use the following format information to read the font block. The first word in the block contains the number of font information elements in the block, followed by a packed array of font information elements. Each element consists of a word containing a font number followed by a Pascal string containing the font name and, if necessary, a pad byte for word alignment.

Constants are not defined for the `'fish'` and `'font'` block creator and type.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$050D

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Requested both plain and styled text segments
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kOCEInvalidRef`	–1502	Invalid message reference number
`kMailInvalidRequest`	–15045	Message reference number does not refer to a letter
`kMailMalformedContent`	–15061	Content data malformed

SEE ALSO

The `MailBuffer` structure is described on page 2-96.

The values that you can use in the `segmentType` and `segmentMask` fields are described in the section "The Segment Types" beginning on page 2-109.

A script run is a sequence of text in a single character set. For more information about script runs, see *Inside Macintosh: Text*.

The `ScrpSTElement` and the `StScrpRec` structures are described in *Inside Macintosh: Text*.

MSAMGetEnclosure

The `MSAMGetEnclosure` function reads file enclosures from a letter that you specify.

```
pascal OSErr MSAMGetEnclosure (MSAMParam *paramBlock,
                               Boolean asyncFlag);
```

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`mailMsgRef`	`MailMsgRef`	Letter reference number
←	`contentEnclosure`	`Boolean`	Is enclosure main letter content?
↔	`buffer`	`MailBuffer`	Your buffer structure
←	`endOfFile`	`Boolean`	End of file?
←	`endOfEnclosures`	`Boolean`	All enclosures read?

See “The MSAM Parameter Block” on page 2-94 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`mailMsgRef`	A reference number that identifies the letter whose enclosures you want to read. You obtain the reference number when you call the `MSAMOpen` function.
`contentEnclosure`	A Boolean value that indicates whether the enclosure is the content enclosure for the letter. A content enclosure contains the content of a letter. It is typically a file in an application’s native format. When you call the `MSAMGetEnclosure` function the first time, it sets this field to `true` if the enclosure is a content enclosure or to `false` if it is not. The function also sets the value of this field the first time you call it after the function sets the `endOfFile` flag to `true`. At other times, consider the value of this field invalid.
`buffer`	A `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. The `MSAMGetEnclosure` function writes the information that you request into your buffer and sets the value of the `dataSize` field to the number of bytes of data it placed in the buffer.
`endOfFile`	A Boolean value that indicates whether an entire enclosure file has been read. If your buffer is not large enough to hold the entire enclosure file, the `MSAMGetEnclosure` function sets the `endOfFile` field to `false`. You can call the function repeatedly until it sets the `endOfFile` field to `true`, at which point an entire enclosure file has been read. The `MSAMGetEnclosure` function does not put data belonging to more than one enclosure file into your buffer at the same time, even when the end of file is reached on one enclosure file, there are additional enclosure files to read, and your buffer is not full. When a letter has no enclosures, the function sets this field to `false`. To detect the no-enclosure condition, test only the `endOfEnclosures` field.
`endOfEnclosures`	A Boolean value that indicates whether the `MSAMGetEnclosure` function has reached the end of all of the enclosures for the letter that you specify. When the `MSAMGetEnclosure` function has retrieved all enclosures for the current nesting level, it sets the `endOfEnclosures` field to `true`.

DESCRIPTION

You call the `MSAMGetEnclosure` function to retrieve all file enclosures for a letter that you specify. To get all of the enclosures in a letter, you should call the function repeatedly until the value of the `endOfEnclosures` field is `true`.

A letter’s enclosures can be folders or Macintosh files in AppleSingle stream format. The `MSAMGetEnclosure` function returns all of the files to you; it does not return any folder information.That is, you do not know how the files might have been organized and stored in the letter.

Because the PowerTalk system software works with the hierarchical file system, it is possible for an outgoing letter to contain more than one enclosed file with the same name, so long as the files are in different enclosed folders. You may need to adjust the filenames of identically named enclosed files so that each one is unique. Otherwise, it is possible that only one of such files will be retained by the external messaging system.

Note
An enclosure is not a nested letter. A nested letter is a letter that a recipient has forwarded or replied to. Enclosures are files or folders that the sender has enclosed with a letter. ◆

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$050E

RESULT CODES

`noErr`	0	No error
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kOCEInvalidRef`	–1502	Invalid message reference number
`kMailInvalidRequest`	–15045	Nested letter already created for this letter

SEE ALSO

The `MailBuffer` structure is described on page 2-96.

For more information on AppleSingle stream format, see the APDA document *AppleSingle/AppleDouble Formats for Foreign Files Developer Note.*

MSAMEnumerateBlocks

The `MSAMEnumerateBlocks` function returns an array of message block descriptors for the blocks in a message.

```
pascal OSErr MSAMEnumerateBlocks (MSAMParam *paramBlock,
                                  Boolean asyncFlag);
```

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`mailMsgRef`	`MailMsgRef`	Message reference number
→	`startIndex`	`unsigned short`	Message block to start from
↔	`buffer`	`MailBuffer`	Your buffer structure
←	`nextIndex`	`unsigned short`	Message block to continue from next time
←	`more`	`Boolean`	Is there more data?

See "The MSAM Parameter Block" on page 2-94 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`mailMsgRef`	A reference number that identifies the message whose message blocks you want to enumerate. You obtain the reference number when you call the `MSAMOpen` function.
`startIndex`	The sequence number of the block about which you want information. Set this field to 1 to start with the first message block. When you call the function and there is insufficient space in your buffer to hold information about all of the remaining blocks, the function returns in the `nextIndex` field the sequence number of the next block. Use that number in the `startIndex` field the next time you call the function.
`buffer`	A `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. The `MSAMEnumerateBlocks` function places data in your buffer in the form of a `MailReply` structure. The first 2 bytes in the `MailReply` structure are a count of the number of `MailBlockInfo` structures, followed immediately by the structures. The function sets the value of the `dataSize` field to the number of bytes of data it placed in the buffer.
`nextIndex`	The sequence number of the first block whose information did not fit into your buffer. The function sets this field when your buffer is too small to hold all the information you requested. If there is no more information to return, the value of the `nextIndex` field is undefined. You must check the value of the `more` field before interpreting the value in the `nextIndex` field. The `nextIndex` field contains meaningful data only when the value of the `more` field is `true`.
`more`	A Boolean value that indicates whether there is more message block information than can fit in your buffer. If your buffer is too small to hold all of the block information that you requested, the `MSAMEnumerateBlocks` function sets this field to `true`; otherwise, it sets this field to `false`. If the function sets this field to `true`, you can call it again to retrieve additional information by setting the `startIndex` field for the next call to the value of the `nextIndex` field.

DESCRIPTION

You call the `MSAMEnumerateBlocks` function to get information about all of the blocks in a message. For each block, the function returns a `MailBlockInfo` structure that specifies the block's creator and type, its offset in bytes from the beginning of the message (the offset is zero-based), and its length in bytes. You can use this information to read specific blocks in the message.

```
struct MailBlockInfo {
   OCECreatorType    blockType;  /* block creator and type */
   unsigned long     offset;     /* offset from start of msg */
   unsigned long     blockLength;/* number of bytes in block */
};
```

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
_oceTBDispatch	$050F

RESULT CODES

`noErr`	0	No error
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kOCEInvalidRef`	–1502	Invalid message reference number
`kOCEBufferTooSmall`	–1503	Buffer is too small

SEE ALSO

The `MailBuffer` structure is described on page 2-96.

The `MailReply` structure is described on page 2-97.

MSAMGetBlock

The `MSAMGetBlock` function reads a block from a message that you specify.

```
pascal OSErr MSAMGetBlock (MSAMParam *paramBlock,
                           Boolean asyncFlag);
```

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`mailMsgRef`	`MailMsgRef`	Message reference number
→	`blockType`	`OCECreatorType`	Block creator and type
→	`blockIndex`	`unsigned short`	Sequential position of block
↔	`buffer`	`MailBuffer`	Your buffer structure
→	`dataOffset`	`unsigned long`	Byte offset within block
←	`endOfBlock`	`Boolean`	End of block?
←	`remaining`	`unsigned long`	Number of bytes not read in block

See “The MSAM Parameter Block” on page 2-94 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`mailMsgRef`	A reference number that identifies the message whose blocks you want to read. You obtain the reference number when you call the `MSAMOpen` function.
`blockType`	A structure that specifies the creator and the type of the block that you want to read. You cannot specify a wildcard value for either the creator or block type.
`blockIndex`	A value that indicates the relative position of the block of type `blockType` that you want to read. To read all blocks of a specific block type, set this field to 1 the first time you call the `MSAMGetBlock` function and increment it by 1 in subsequent calls to the function until you have read all blocks of that type in the message. (Note that the value you supply here is distinct from the index used in the `MSAMEnumerateBlocks` function.)
`buffer`	A `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. The `MSAMGetBlock` function writes the information that you request into your buffer and sets the value of the `dataSize` field to the number of bytes of data it placed in the buffer.
`dataOffset`	The byte position relative to the beginning of the block at which you want the `MSAMGetBlock` function to begin reading. Set this field to 0 to read from the beginning of the block.
`endOfBlock`	A Boolean value that indicates whether the `MSAMGetBlock` function has returned the entire block. If the buffer that you provide is not large enough to contain an entire block, the `MSAMGetBlock` function sets this field to `false`. You can call the function again with an updated value in the `dataOffset` field to retrieve additional data. When the `MSAMGetBlock` function has returned the entire block, it sets the value of the `endOfBlock` field to `true`.
`remaining`	The number of bytes of data remaining in the block that the `MSAMGetBlock` function has not returned to you. If the `endOfBlock` field is set to `true`, the value of this field is 0.

DESCRIPTION

You call the `MSAMGetBlock` function to read data from a block in a message. You identify the block that you want to read by the values of the `blockType` and `blockIndex` fields. Use the `dataOffset` field to identify the point at which you want to begin reading within your chosen block.

Typically, you call the `MSAMGetBlock` function to read report blocks, image blocks, and private blocks because the MSAM API provides no other way to read these types of blocks. Although it is possible to call the `MSAMGetBlock` function to read blocks that contain letter content, attributes, enclosures, and so forth, the internal format of these blocks is private. You should use the specific functions provided in the MSAM API for reading these types of blocks.

There are no restrictions on the number of times that you may read a given block. You may read the blocks in a message in any order.

To read a report block, in the `blockType` field, set the block creator to `kMailAppleMailCreator` and set the block type to `kMailReportType`. Set the `blockIndex` field to 1. The `MSAMGetBlock` function places a report block in your buffer. The data in a report block consists of a header, `IPMReportBlockHeader`, followed by an array of elements, each of type `OCERecipientReport`. (You can detect a report in your outgoing queue when you call the `MSAMEnumerate` function. The message creator is always `kIPMSignature` and the message type is `kIPMReportNotify`.)

To read an image block, in the `blockType` field, set the block creator to `kMailAppleMailCreator` and set the block type to `kMailImageBodyType`. The data that the `MSAMGetBlock` function places in your buffer is a structure of type `TPfPgDir`, followed by the actual picture elements (PICTs).

Blocks of type `kMailMSAMType` contain data whose format and content are private to an MSAM. To read a private block, in the `blockType` field, set the block creator to a value that you define, and set the block type to `kMailMSAMType`.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0510

RESULT CODES

`noErr`	0	No error
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kOCEInvalidRef`	–1502	Invalid message reference number
`kIPMBlkNotFound`	–15107	No such block

SEE ALSO

The `IPMReportBlockHeader`, `OCECreatorType`, and `OCERecipientReport` structures are described in the chapter “Interprogram Messaging Manager” in *Inside Macintosh: AOCE Application Interfaces*.

The `MailBuffer` structure is described on page 2-96.

The `TPfPgDir` structure is described on page 2-113.

For more information about PICT format, see *Inside Macintosh: Imaging With QuickDraw.*

The `MailIndications` structure is described beginning on page 2-102.

MSAMOpenNested

The `MSAMOpenNested` function opens a message that is nested within a message that you specify.

```
pascal OSErr MSAMOpenNested (MSAMParam *paramBlock,
                             Boolean asyncFlag);
```

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`mailMsgRef`	`MailMsgRef`	Message reference number
←	`nestedRef`	`MailMsgRef`	Reference number of the nested message

See "The MSAM Parameter Block" on page 2-94 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`mailMsgRef`	A reference number that identifies the message that contains a nested message that you want to open. You obtain the reference number when you call the `MSAMOpen` function.
`nestedRef`	A reference number that identifies the nested message opened by the `MSAMOpenNested` function.

DESCRIPTION

Call `MSAMOpenNested` to open a message that is nested within a message. You can open only one message nested within a message at a given nesting level. A nested message itself may contain a nested message.

The `MSAMOpenNested` function returns a reference number to the opened nested message. The nested message reference number is analogous to the message reference number of the parent message. Use the nested message reference number when calling functions to read or close the nested message.

You can call the `MSAMClose` function to close the nested message explicitly. Alternately, you can close a nested message by closing its parent message. The `MSAMClose` function always closes the message you specify and all messages nested within it.

SPECIAL CONSIDERATIONS

Although a letter, by definition, can have only one nested letter per nesting level, a non-letter message may actually have more than one nested message per nesting level. The IPM Manager API allows applications to create such messages. However, you can open only the first message nested within a message at a given nesting level.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0509

RESULT CODES

`noErr`	0	No error
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kOCEInvalidRef`	–1502	Invalid message reference number
`kOCEVersionErr`	–1504	Wrong version of nested message

SEE ALSO

Nested messages are described in the section “Letters” beginning on page 2-17.

The `MSAMClose` function is described on page 2-167.

Marking a Recipient

When you have completed your attempts to deliver a message to a recipient, you should mark the recipient, indicating that you have completed your delivery attempts. The `MSAMnMarkRecipients` function allows you to do that. If you need to mark a recipient of a message you have closed, you can use the `MSAMMarkRecipients` function.

MSAMnMarkRecipients

The `MSAMnMarkRecipients` function allows you to indicate that you have completed your attempts to deliver a given open message to the recipients that you specify.

```
pascal OSErr MSAMnMarkRecipients (MSAMParam *paramBlock,
                                  Boolean asyncFlag);
```

`paramBlock` Pointer to a parameter block.

`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`mailMsgRef`	`MailMsgRef`	Message identifier
↔	`buffer`	`MailBuffer`	Your buffer structure

See "The MSAM Parameter Block" on page 2-94 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`mailMsgRef`	A message reference number that identifies an open message whose recipients you want to mark. You obtain this reference number from the `MSAMOpen` function. It is valid if you have not yet closed the message by calling the `MSAMClose` function.
`buffer`	A `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. You place data in your buffer in the form of a `MailReply` structure. The first 2 bytes in the buffer contain the number of identifying values that follow. Then you store a value that identifies each recipient that you want to mark. Each identifying value is 2 bytes long. The `dataSize` field in the `MailBuffer` structure is unused.

DESCRIPTION

Calling the `MSAMnMarkRecipients` function for one or more recipients indicates that you have delivered the specified message or have finished attempting to deliver the message to those recipients. You may have delivered the message directly to a recipient or to an agent within the non-AOCE system that has responsibility for delivery to the final destination.

The value that identifies a recipient that you want to mark is its ordinal position in the buffer returned by the `MSAMGetRecipients` function. When you call the `MSAMGetRecipients` function to get resolved recipients, `MSAMGetRecipients` places some number of `MailResolvedRecipient` structures in your buffer. You must save the ordinal-position value of each resolved recipient as you retrieve these structures. The first recipient's ordinal-position value is 1; the second recipient's ordinal-position value is 2 (the *nth* recipient's ordinal-position value is *n*). Do not use the absolute index of the recipient contained in a `MailResolvedRecipient` structure to identify a recipient. The `MSAMnMarkRecipients` function will not work correctly if you do so.

The `MSAMnMarkRecipients` function clears the `responsible` flag for the recipients you specify. If you call the `MSAMGetRecipients` function after calling `MSAMnMarkRecipients`, the marked recipients have the `responsible` field of their corresponding `MailResolvedRecipient` structures set to `false`. After you mark all of the recipients for a message, the `done` field in the `MSAMEnumerateOutQReply` structure is set to `true` for that message when you enumerate the outgoing queue.

You can call the `MSAMnMarkRecipients` function more than once for a given message, specifying one or more recipients each time you call it.

Note
Calling the `MSAMnMarkRecipients` function for a given recipient does not necessarily mean that you have successfully delivered the message. You should use a report to indicate whether or not you have successfully delivered a message. ◆

SPECIAL CONSIDERATIONS

If you must mark a recipient of a message you have closed, you can call the earlier version of this function, the `MSAMMarkRecipients` function. Instead of a message reference number, you provide the reference number of the outgoing queue that contains the message and the message sequence number. The `MSAMMarkRecipients` function produces the same result, but it executes much more slowly.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0512

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Incoming queue reference not allowed
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kOCEInvalidRef`	–1502	Invalid queue reference
`kOCEDoesntExist`	–1511	No such letter
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM, or server MSAM's mail server is shutting down

SEE ALSO

The `MailResolvedRecipient` structure is described on page 2-108.

The `MailBuffer` structure is described on page 2-96.

The `MailReply` structure is described on page 2-97.

The `MSAMGetRecipients` function is described beginning on page 2-144.

The `MSAMMarkRecipients` function is described next.

MSAMMarkRecipients

The `MSAMMarkRecipients` function, like the `MSAMnMarkRecipients` function, allows you to indicate that you have completed your attempts to deliver a particular message to the recipients that you specify, but it executes much more slowly.

```
pascal OSErr MSAMMarkRecipients (MSAMParam *paramBlock,
                                 Boolean asyncFlag);
```

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`queueRef`	`MSAMQueueRef`	Queue reference number
→	`seqNum`	`long`	Message sequence number
↔	`buffer`	`MailBuffer`	Your buffer structure

See “The MSAM Parameter Block” on page 2-94 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`queueRef`	The value that identifies the outgoing queue that contains the message whose recipients you want to mark.
`seqNum`	A value that identifies the message whose recipients you want to mark. You obtain this value from the `MSAMEnumerate` function.
`buffer`	A `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. You place data in your buffer in the form of a `MailReply` structure. The first 2 bytes in the buffer contain the number of identifying values that follow. Then you store a value that identifies each recipient that you want to mark. Each identifying value is 2 bytes long. The identifying value is described on page 2-164. The `dataSize` field in the `MailBuffer` structure is unused.

DESCRIPTION

The `MSAMMarkRecipients` function produces the same result as the `MSAMnMarkRecipients` function, described in the previous section.

SPECIAL CONSIDERATIONS

It is strongly recommended that you do not call this function unless you must mark a recipient for a message that you have already closed. Instead, you should call the `MSAMnMarkRecipients` function, which executes much more quickly.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0505

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Incoming queue reference not allowed
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kOCEInvalidRef`	–1502	Invalid queue reference
`kOCEDoesntExist`	–1511	No such letter
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM, or server MSAM's mail server is shutting down

SEE ALSO

The `MailResolvedRecipient` structure is described on page 2-108.

The `MailBuffer` structure is described on page 2-96.

The `MSAMGetRecipients` function is described on page 2-144.

The `MSAMnMarkRecipients` function is described on page 2-163.

Closing a Message

When you have finished reading a message, whether it is nested or not, use the function `MSAMClose` to close the message.

MSAMClose

The `MSAMClose` function closes an open message that you specify.

```
pascal OSErr MSAMClose (MSAMParam *paramBlock, Boolean asyncFlag);
```

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`mailMsgRef`	`MailMsgRef`	Message reference number

See "The MSAM Parameter Block" on page 2-94 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`mailMsgRef`	A reference number that identifies the message that you want to close. You obtain the reference number when you call the `MSAMOpen` function. When the `MSAMClose` function completes successfully, this reference number is no longer valid.

DESCRIPTION

The `MSAMClose` function closes any message or nested message that you have previously opened. Closing a letter automatically closes any open nested messages within it.

You should close a message once you have read it and have marked the recipients for the message. Closing a message releases system resources. You can reopen a message you previously closed by calling the `MSAMOpen` function.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$050A

RESULT CODES

`noErr`	0	No error
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kOCEInvalidRef`	–1502	Invalid message reference number

SEE ALSO

The `MSAMOpen` function is described on page 2-140.

Creating, Reading, and Writing Message Summaries

A personal MSAM must create a message summary for each letter it transfers from an external messaging system to an AOCE system. Message summaries are stored in the incoming queue for a slot and are used by the Finder to display information about the letters to the user. You use the `PMSAMCreateMsgSummary` function to create a new message summary. Once you have created a message summary, you can modify portions of it. To do so, first call the `PMSAMGetMsgSummary` function to read the message summary; then modify it; and, finally, call the `PMSAMPutMsgSummary` function to write it again.

Note that a personal MSAM creates message summaries only for letters, not for other types of messages.

PMSAMCreateMsgSummary

The `PMSAMCreateMsgSummary` function creates a message summary in an incoming queue that you specify.

```
pascal OSErr PMSAMCreateMsgSummary (MSAMParam *paramBlock,
                                   Boolean asyncFlag);
```

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`inQueueRef`	`MSAMQueueRef`	Incoming queue reference
←	`seqNum`	`long`	Message summary sequence number
→	`msgSummary`	`MSAMMsgSummary*`	Summary information for a letter
↔	`buffer`	`MailBuffer*`	Your private data

See “The MSAM Parameter Block” on page 2-94 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`inQueueRef`	The reference number that identifies the queue into which you want to place the message summary. You obtain the queue reference from the `PMSAMOpenQueues` function.
`seqNum`	The sequence number of the new message summary. You use the sequence number with the `PMSAMGetMsgSummary` and `PMSAMPutMsgSummary` functions to identify the message summary.
`msgSummary`	A pointer to an `MSAMMsgSummary` structure that you allocate. You must provide values for some of the fields of the structure.
`buffer`	A pointer to a `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. Your buffer size may not exceed the number of bytes specified by the `kMailMaxPMSAMMsgSummaryData` constant. You provide a pointer to your buffer in the `buffer` field of the structure and store in the buffer private data that you want to add to the message summary. The function reads your data from the buffer and sets the value of the `dataSize` field to the number of bytes of data it wrote to the message summary. Set this field to `nil` if you do not want to add any private data to the message summary.

DESCRIPTION

You call the `PMSAMCreateMsgSummary` function to create a message summary for an incoming letter. You must create a message summary for each incoming letter. The Finder uses the message summary to display information about the letter to the user. (Because only letters are displayed to the user, you do not create a message summary for a message that is not a letter.)

Prior to assigning a particular value to any field of a new `MSAMMsgSummary` structure, you should initialize all of its fields to 0. The section "The Personal MSAM Message Summary Structures" beginning on page 2-124 describes all of the fields of the message summary and indicates whether you or the IPM Manager is responsible for providing a value for a given field. Note that when the IPM Manager adds a value in the message summary, it updates the `MSAMMsgSummary`.`MailMasterData`.`attrMask` field if appropriate.

With one exception, the values of the letter attributes that you provide when you create a message summary must be exactly the same values as those you provide to the `MSAMPutAttribute` function when you write the associated letter to the incoming queue. If the attribute values do not match, the consequences are unpredictable. The exception is the `subject` attribute. It may be truncated in the message summary due to size limitations in the `MSAMMsgSummary` structure.

You can provide private data that the IPM Manager stores with the message summary. If your private data exceeds `kMailMaxPMSAMMsgSummaryData` bytes, the function returns the `kOCEParamErr` result code.

You can modify your private data. To do so, call the `PMSAMGetMsgSummary` function to read your private data associated with the message summary; then modify your data; and, finally, call the `PMSAMPutMsgSummary` function to write your modified private data.

The `PMSAMCreateMsgSummary` function returns a sequence number. You must provide the sequence number to the `MSAMCreate` function when you create the letter for this message summary. The sequence number correctly associates the letter and the message summary.

SPECIAL CONSIDERATIONS

The private data area associated with a message summary is a sort of scratch pad, intended for brief notations for MSAM-specific uses. Storing large amounts of data degrades system performance and is strongly discouraged. For best results, you should use no more than 8–16 bytes of private data.

The `sender` and `subject` fields of the `MailCoreData` structure in the message summary require special handling. Be sure to read the information in the section "Creating a Letter's Message Summary" beginning on page 2-64 for an understanding of how to manipulate these fields.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0522

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	All allocation blocks on the volume are full
`kOCEParamErr`	–50	Private data too large
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kOCEInvalidRef`	–1502	Invalid queue reference
`kMailInvalidPostItVersion`	–15046	Message summary is wrong version
`kMailNotASlotInQ`	–15047	Queue reference does not refer to an incoming queue

SEE ALSO

The `MSAMMsgSummary` structure is described on page 2-127.

The `MailCoreData` structure is described on page 2-125.

The `PMSAMGetMsgSummary` function is described next.

The `PMSAMPutMsgSummary` function is described on page 2-173.

The `MSAMPutAttribute` function is described on page 2-179.

For more information on the use of message summaries and for sample code that shows how to create a message summary, see the section "Creating a Letter's Message Summary" beginning on page 2-64.

PMSAMGetMsgSummary

The `PMSAMGetMsgSummary` function reads a message summary, an MSAM's private data associated with a message summary, or both.

```
pascal OSErr PMSAMGetMsgSummary (MSAMParam *paramBlock,
                                 Boolean asyncFlag);
```

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`inQueueRef`	`MSAMQueueRef`	Incoming queue reference
→	`seqNum`	`long`	Message summary sequence number
↔	`msgSummary`	`MSAMMsgSummary*`	Message summary
↔	`buffer`	`MailBuffer*`	Buffer for private data
→	`msgSummaryOffset`	`unsigned short`	Point at which to begin reading

See "The MSAM Parameter Block" on page 2-94 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`inQueueRef`	The value identifying the incoming queue that holds the message summary you want to read. You obtain the queue reference from the `PMSAMOpenQueues` function.
`seqNum`	The sequence number that identifies the message summary in the incoming queue. You obtain this value from the `PMSAMCreateMsgSummary` function.
`msgSummary`	A pointer to a buffer in which the function stores the `MSAMMsgSummary` structure. You provide this buffer. Set this field to `nil` if you do not want to read the message summary.
`buffer`	A pointer to a `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. The `PMSAMGetMsgSummary` function stores your private data associated with the message summary into the buffer and sets the value of the `dataSize` field to the number of bytes of data it actually placed in your buffer. Set this field to `nil` if you do not want to read your private data.
`msgSummaryOffset`	The offset from the beginning of your private data area identifying the point at which you want to begin reading. If the `buffer` field is set to `nil`, the function ignores this field.

DESCRIPTION

You call the `PMSAMGetMsgSummary` function to read an existing message summary, the private data associated with the message summary, or both.

You can modify the `letterFlags` field of the `MSAMMsgSummary` structure or your private data, or both.

If the `msgUpdated` flag in the message summary was set to `true`, the IPM Manager resets it to `false` after the `PMSAMGetMsgSummary` function returns with no error.

SPECIAL CONSIDERATIONS

Reading your private data area is slower than reading the `MSAMMsgSummary` structure. Each read request may result in two additional disk accesses. You should avoid reading your private data whenever it is reasonable to do so.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0526

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kOCEInvalidRef`	–1502	Invalid queue reference
`kOCEDoesntExist`	–1511	No such message summary
`kMailNotASlotInQ`	–15047	Queue reference does not refer to an incoming queue

SEE ALSO

You use the `PMSAMPutMsgSummary` function to write the modified message summary, private data, or both. The `PMSAMPutMsgSummary` function is described next.

The `MSAMMsgSummary` structure is described on page 2-127.

The `MailBuffer` structure is described on page 2-96.

PMSAMPutMsgSummary

The `PMSAMPutMsgSummary` function writes a modified message summary, private data associated with the message summary, or both.

```
pascal OSErr PMSAMPutMsgSummary (MSAMParam *paramBlock,
                                 Boolean asyncFlag);
```

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`inQueueRef`	`MSAMQueueRef`	Slot's incoming queue reference
→	`seqNum`	`long`	Message summary's sequence number
→	`letterFlags`	`MailMaskedLetterFlags*`	System and user flags
↔	`buffer`	`MailBuffer*`	Private data buffer

See "The MSAM Parameter Block" on page 2-94 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`inQueueRef`	The value that identifies the queue in which the message summary resides. You obtain this value from the `PMSAMOpenQueues` function.
`seqNum`	The sequence number that identifies the message summary that you want to modify or whose associated private data you want to modify.
`letterFlags`	A pointer to a `MailMaskedLetterFlags` structure, which consists of a set of user and system flags and their values. The flags indicate certain aspects of the status of your letter. You can modify the `kMailReadBit` bit in the user flags portion of the letter flags. Set this field to `nil` if you do not want to modify the `kMailReadBit` bit.
`buffer`	A pointer to a `MailBuffer` structure that contains your private data associated with the message summary. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. Your buffer size may not exceed the number of bytes specified by `kMailMaxPMSAMMsgSummaryData`. The `PMSAMPutMsgSummary` function stores your private data with the message summary and sets the value of the `dataSize` field to the number of bytes of data it actually wrote. Set this field to `nil` if you do not want to modify your private data.

DESCRIPTION

You use the `PMSAMPutMsgSummary` function to overwrite your private data associated with a message summary, to modify the user flags portion of the letter flags, or both.

You can modify the `kMailReadBit` bit in the user portion of letter flags in a letter's message summary. Typically, you do this to reflect, in the incoming queue, changes in a letter's status on the external messaging system. For example, when you write a letter to an incoming queue, you initially set the `kMailReadBit` bit to 0 to indicate that the user has not read the letter. Assume that the user logs onto the external account directly, perhaps while travelling, and reads the letter. The next time you connect to the external system, you note that the letter has been read. At this point, you can call the `PMSAMPutMsgSummary` function to set the `kMailReadBit` bit to 1, indicating that the user read the letter. Note that the `kMailReadBit` bit applies to the letter in general, not simply a local copy of the letter.

You manage your private data for your own purposes. If you provide more than the maximum number of bytes (`kMailMaxPMSAMMsgSummaryData`) of private data in your buffer, the function returns the `kOCEParamErr` result code.

SPECIAL CONSIDERATIONS

Writing your private data area is slower than writing the letter flags in a message summary. Each write request may result in two additional disk accesses. You should avoid writing your private data whenever it is reasonable to do so.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0527

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	All allocation blocks on the volume are full
`kOCEParamErr`	–50	Invalid parameter
`kOCEToolboxNotOpen`	–1500	Collaboration toolbox is shutting down
`kOCEInvalidRef`	–1502	Invalid queue reference
`kOCEDoesntExist`	–1511	No such message summary
`kMailInvalidPostItVersion`	–15046	Message summary is wrong version
`kMailNotASlotInQ`	–15047	Queue reference does not refer to an incoming queue

SEE ALSO

The `MailMaskedLetterFlags` structure is described on page 2-124. The user portion of the letter flags is defined by the `MailLetterUserFlags` data type, described on page 2-122.

The `MSAMMsgSummary` structure is described on page 2-127.

The `PMSAMGetMsgSummary` function is described on page 2-171.

The `MailBuffer` structure is described on page 2-96.

Creating a Message

To create a new message going to an AOCE address, use the function `MSAMCreate`.

MSAMCreate

The `MSAMCreate` function begins the process of creating a message and returns a reference number for the message.

```
pascal OSErr MSAMCreate (MSAMParam *paramBlock,
                         Boolean asyncFlag);
```

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`queueRef`	`MSAMQueueRef`	Queue reference number
→	`asLetter`	`Boolean`	Create a letter?
→	`msgType`	`IPMMsgType`	Message creator and type
→	`refCon`	`long`	Reserved for your use
→	`seqNum`	`long`	Sequence number of new message
→	`tunnelForm`	`Boolean`	Always `false`
→	`bccRecipients`	`Boolean`	Are there blind copy recipients?
←	`newRef`	`MailMsgRef`	Message reference number

See “The MSAM Parameter Block” on page 2-94 for descriptions of the `ioResult` and `ioCompletion` fields.

Field descriptions

`queueRef`	For a personal MSAM, specify the incoming queue reference that you obtained from the `PMSAMOpenQueues` function. The queue reference must belong to the slot to which the message is addressed. For a mail slot, the queue reference identifies the slot’s actual incoming queue in which you want to deposit a letter. For a messaging slot, the queue reference identifies the slot itself. For a server MSAM, specify the server MSAM’s queue reference that you obtained from the `SMSAMStartup` function.
`asLetter`	A Boolean value that indicates whether the message you are creating is a letter.
`msgType`	An `IPMMsgType` structure. If you are creating a letter, you must set the `format` field of the `IPMMsgType` structure to `kIPMOSFormatType` to indicate that the remainder of the `IPMMsgType` structure consists of an `OCECreatorType` structure.

	Then set the message creator and type appropriately. If you are creating a non-letter message, you can set the `IPMMsgType` field to either format type, `kIPMOSFormatType` or `kIPMStringFormatType`.
`refCon`	A value reserved for your private use when you create a non-letter message. You may provide a value to be interpreted by the recipient. This field is ignored when you create a letter. If you provide a value in the `refCon` field, it is stored in the message header. The recipient can retrieve the value by calling the `MSAMGetMsgHeader` function and specifying `kIPMFixedInfo` in the `selector` field of its parameter block.
`seqNum`	This field applies only to personal MSAMs. If you are creating a message that is not a letter, you do not provide a value for this field. Otherwise, you provide the sequence number that identifies the message summary associated with the letter that you want to create. You obtain the sequence number from the `PMSAMCreateMsgSummary` function.
`tunnelForm`	You must always set this field to `false`.
`bccRecipients`	This field applies only when you want to create a letter. You set this field to `true` if you intend to specify blind copy recipients for the letter when you call the `MSAMPutRecipient` function.
`newRef`	A value that uniquely identifies the message that has just been created. The `MSAMCreate` function returns a reference number for the message that you use in subsequent function calls to write the message.

DESCRIPTION

You call the `MSAMCreate` function to begin the process of writing a message from an external messaging system to an AOCE system. The function returns a reference number that you need to provide to the *MSAMPut* functions that write the various parts of the message.

If you are creating a letter that contains data in standard interchange format, image format, or a regular enclosure, you should set the message creator to `'lap2'` and the message type to `kMailLtrMsgType`. In this case, the AppleMail application opens the letter. If the letter contains only a content enclosure, you can set the message creator to the signature of the application that created the content enclosure. If the letter contains a content enclosure or private block and if you set the message creator to the signature of the application that created the enclosure or private block, then you can use a message type that you define consistent with the message creator.

You set the message creator and message type in the `msgCreator` and `msgType` fields of the `OCECreatorType` structure, part of the`IPMMsgType` structure.

If you are creating a non-letter message, use an application-defined creator and type. You can set the `format` field of the `IPMMsgType` structure to either `kIPMOSFormatType` (which specifies that the message creator and message type information is formatted as type `OCECreatorType`) or `kIPMStringFormatType` (which specifies that the message creator and message type information is formatted as type `Str32`). Typically, you use type `OCECreatorType`; type `Str32` is included for compatibility with the Program-to-Program Communications (PPC) Toolbox.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0514

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	All allocation blocks on the volume are full
`memFullErr`	–108	Not enough memory
`kOCEInvalidRef`	–1502	Invalid queue reference number
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM, or server MSAM's mail server is shutting down
`kMailNotASlotInQ`	–15047	Queue reference refers to a personal MSAM's outgoing queue
`kIPMInvalidMsgType`	–15091	Only `kIPMOSFormatType` allowed when creating a letter

SEE ALSO

The types of data that constitute standard letter content are described on page 2-109.

The `IPMMsgType` and the format types, `kIPMOSFormatType` and `kIPMStringFormatType`, are described in the chapter "Interprogram Messaging Manager" in *Inside Macintosh: AOCE Application Interfaces*.

The `OCECreatorType` structure is described in the chapter "Interprogram Messaging Manager" in *Inside Macintosh: AOCE Application Interfaces*.

The `PMSAMOpenQueues` function is described on page 2-133.

The `PMSAMGetMsgSummary` function is described on page 2-171.

The `MSAMPutRecipient` function is described on page 2-180.

See the section "Choosing Creator and Type for Messages and Blocks" beginning on page 2-64 for a discussion of message creators and types.

Writing Header Information

To write letter attributes into a newly created letter, use the `MSAMPutAttribute` function. You can add recipients to a message with the `MSAMPutRecipient` function. To write the header of a non-letter message, use the `MSAMPutMsgHeader` function.

MSAMPutAttribute

The `MSAMPutAttribute` function adds a letter attribute to a letter you are writing.

```
pascal OSErr MSAMPutAttribute (MSAMParam *paramBlock,
                               Boolean asyncFlag);
```

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`mailMsgRef`	`MailMsgRef`	Letter reference number
→	`attrID`	`MailAttributeID`	Type of letter attribute
↔	`buffer`	`MailBuffer`	Your buffer structure

See “The MSAM Parameter Block” on page 2-94 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`mailMsgRef`	A reference number that identifies the letter to which you want to add an attribute. You obtain the reference number when you call the `MSAMCreate` function.
`attrID`	A value that identifies the type of attribute that you want to add to the letter.
`buffer`	A `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. You store the value of the attribute that you want to add to the letter in your buffer. The `MSAMPutAttribute` function writes the information from the buffer to the letter and sets the value of the `dataSize` field to the number of bytes of data it actually wrote.

DESCRIPTION

You call the `MSAMPutAttribute` function to add a letter attribute to a letter header. The `attrID` field can have any of the following values:

Constant	Value	Description
`kMailIndicationsBit`	3	Indications and priority
`kMailSendTimeStampBit`	6	Send timestamp
`kMailMsgFamilyMask`	8	Message family
`kMailReplyIDBit`	9	Reply ID
`kMailConversationIDBit`	10	Conversation ID
`kMailSubjectBit`	11	Subject

You cannot use the `MSAMPutAttribute` function to add recipients to a letter. Use the `MSAMPutRecipient` function to add the From, To, cc, and bcc attributes to a letter.

There are three attributes—the letter's creator and type, its letter ID, and its nesting level—that you can read from a letter header with the `MSAMGetAttributes` function but cannot write to the letter header with `MSAMPutAttribute`. You set the letter's creator and type when you call the `MSAMCreate` function to create the letter, and the IPM Manager sets the letter ID and nesting level of any letters that you create.

The `letterFlags` attribute is stored in a letter's message summary rather than in the letter header. Therefore, you add the `letterFlags` attribute when you call the `PMSAMCreateMsgSummary` function.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0518

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	All allocation blocks on the volume are full
`kOCEParamErr`	–50	Invalid parameter
`kOCEInvalidRef`	–1502	Invalid message reference number
`kOCEAlreadyExists`	–1510	Attribute already exists in the letter header
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM, or server MSAM's mail server is shutting down
`kMailInvalidRequest`	–15045	Cannot call function with this message reference number

SEE ALSO

The `MailBuffer` structure is described on page 2-96.

Letter attributes and their formats are defined in the section "The Letter Attribute Structures" beginning on page 2-99.

The `MSAMGetAttributes` function is described on page 2-142.

The `PMSAMCreateMsgSummary` function is described on page 2-169.

The `MSAMPutRecipient` function is described next.

MSAMPutRecipient

The `MSAMPutRecipient` function adds a recipient to a message you are writing.

```
pascal OSErr MSAMPutRecipient (MSAMParam *paramBlock,
                               Boolean asyncFlag);
```

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`mailMsgRef`	`MailMsgRef`	Message reference number
→	`attrID`	`MailAttributeID`	Type of recipient
→	`recipient`	`MailRecipient*`	Recipient information
→	`responsible`	`Boolean`	Must server MSAM deliver message?

See “The MSAM Parameter Block” on page 2-94 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`mailMsgRef` A reference number that identifies the message to which you want to add recipient information. You obtain the reference number from the `MSAMCreate` function.

`attrID` A constant that indicates the type of recipient you want to add to the message. If you are adding a recipient to a letter, you can use any of the following constants; if you are adding a recipient to a non-letter message, `kMailToBit` is the only valid value you can specify in this field.

Constant	Value	Recipient type
`kMailFromBit`	12	From
`kMailToBit`	13	To
`kMailCcBit`	14	cc
`kMailBccBit`	15	bcc

`recipient` A pointer to a `MailRecipient` structure in which you provide complete addressing information about the recipient.

`responsible` A Boolean value that indicates whether the IPM Manager is responsible for delivering this message to the recipient identified by the `rcpt` field.

DESCRIPTION

You call the `MSAMPutRecipient` function to add a recipient to a message that you specify. You can add one recipient each time you call the function. To add a list of recipients, you must call the function multiple times.

If you are adding a recipient to a letter, you can specify any type of recipient: From, To, cc, or bcc. If you are adding a recipient to a non-letter message, you can specify only a To recipient. To add a From recipient to a non-letter message, call the `MSAMPutMsgHeader` function and specify the From recipient in the `replyQueue` field.

When you add the From address to a letter, you should set the `recordName` field in the `MailRecipient` structure to the value you provided in the `sender` field when you created the letter's message summary.

You must add all recipients of a given recipient type in consecutive calls to the `MSAMPutRecipient` function. If you attempt to intermingle calls to add different recipient types, the function returns a `kOCEAlreadyExists` result code. For example, if you call the function to add a To recipient, call it again to add a cc recipient, and call it a third time to add a second To recipient, the function returns the error the third time you call it.

A personal MSAM should check each recipient address to see if it maps to the owner of the computer. If so, you need to set the `recordName` field in the `MailRecipient` structure to the owner's name, sometimes referred to as the *Key Chain name* or *local identity name*. You can obtain the owner's name by looking up the record attribute indexed by the constant `kLocalNameAttrTypeNum` in the Setup record in the Setup catalog.

Every time you add a recipient, you must indicate if the IPM Manager is responsible for delivering the message to that recipient. If you are adding a From recipient, you should always set the `responsible` field to `false`.

A personal MSAM should set the `responsible` field as follows. If you are adding a recipient to a letter, always set the `responsible` field to `false`. If you are adding a recipient to a non-letter message, set the `responsible` field to `true` for the recipients that are local to the computer on which the MSAM is running. These are the ones for which you want the AOCE system to be responsible for delivering the message. Take, for example, an application that sends the same non-letter message to three other applications, each of which is running on a separate computer. A personal MSAM receiving this message would call the `MSAMPutRecipient` function three times, setting the `responsible` field to `true` for the recipient that is local and to `false` for the other two recipients.

To modify the example a bit, suppose an application sends the same non-letter message to three other applications, all of which are running on the same computer. In this case, the personal MSAM receiving the message would call the `MSAMPutRecipient` function three times, setting the `responsible` field to `true` for all three of the recipients.

For incoming non-letter messages, it is the task of the personal MSAM and its external messaging system to identify addresses that are local to the computer on which the personal MSAM is running so that the personal MSAM can set the `responsible` field appropriately. When a personal MSAM sets the `responsible` field to `true`, the AOCE software attempts to deliver the message to the named queue on the local computer.

Server MSAMs should set the `responsible` field to `true` for any To, cc, or bcc recipient to which they want the AOCE system to deliver a message, regardless of the type of message.

Note that when you call the `MSAMCreate` function, you create a letter or a non-letter message by setting the `asLetter` field to `true` or `false`, respectively.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0519

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	All allocation blocks on the volume are full
`kOCEParamErr`	–50	Invalid parameter
`kOCEInvalidRef`	–1502	Invalid message reference number
`kOCEAlreadyExists`	–1510	Duplicate recipient type
`kOCEInvalidRecipient`	–1514	Bad recipient
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM, or server MSAM's mail server is shutting down
`kMailMalformedContent`	–15061	Content data malformed

SEE ALSO

The `MailRecipient` structure is defined to be an `OCERecipient` structure, which is described on page 2-106.

Recipient types are included in letter attributes. Letter attributes and their formats are defined in the section "The Letter Attribute Structures" beginning on page 2-99.

The `MSAMPutMsgHeader` function is described next.

MSAMPutMsgHeader

The `MSAMPutMsgHeader` function writes information to the header of a non-letter message that you specify.

```
pascal OSErr MSAMPutMsgHeader (MSAMParam *paramBlock,
                              Boolean asyncFlag);
```

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies whether the function is to be executed asynchronously. Set this to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`mailMsgRef`	`MailMsgRef`	Message reference number
→	`replyQueue`	`OCERecipient*`	Return address
→	`sender`	`IPMSender*`	Sender's record ID
→	`deliveryNotification`	`IPMNotificationType`	Delivery notification option
→	`priority`	`IPMPriority`	Delivery priority setting

See "The MSAM Parameter Block" on page 2-94 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`mailMsgRef`	A reference number that identifies the non-letter message whose header you want to write. You obtain the reference number when you call the `MSAMCreate` function.
`replyQueue`	A pointer to an `OCERecipient` structure that specifies a reply queue—that is, a return address. You allocate the structure and completely specify it. The receiving application uses this address when it replies to the message. The IPM Manager sends reports to the reply queue address. You are free to specify that replies and reports go to an alternate address, instead of to the sender.
`sender`	A pointer to an `IPMSender` structure that contains the packed record ID or string that identifies the sender of the message.
`deliveryNotification`	A bit array that indicates the type of information you want to receive about the delivery of the message. Set the bit values appropriately to request reports with delivery indications (`kIPMDeliveryNotificationMask`), reports with non-delivery indications (`kIPMNonDeliveryNotificationMask`), or no reports (`kIPMNoNotificationMask`).
`priority`	A value that specifies the priority for delivering the message. Set this field to `kIPMHighPriority` to specify high priority. Set this field to `kIPMLowPriority` to specify low priority. Set this field to `kIPMNormalPriority` to specify normal priority.

DESCRIPTION

You call the `MSAMPutMsgHeader` function to write information to the header of the non-letter message that you are creating. Do not use this function with messages that are letters or reports. The information that you provide to the `MSAMPutMsgHeader` function includes an address for replies, the sender, the type of report information you want, and the priority for delivering the message.

You should understand the distinction between the use of the `sender` and the `replyQueue` fields. The address that you provide in the `replyQueue` field shows up as the From recipient when the message is delivered. It allows a sender to designate an address to which replies should be sent. For example, cooperating applications can agree to define reply queue addresses that are associated with specific message creators, message types, and message families. In addition, the IPM Manager sends reports to the reply queue address.

In contrast, the `sender` field simply identifies the originator of the message. A recipient can retrieve the value of the `sender` field by calling the `MSAMGetMsgHeader` function. The record ID portion of the return address need not be the same as that which you provide in the `sender` field.

The IPM Manager defines several masks for delivery notification options. However, the only valid values that you can use to set bits in the `deliveryNotification` field are `kIPMDeliveryNotificationMask`, `kIPMNonDeliveryNotificationMask`, and `kIPMNoNotificationMask`. The IPM Manager ignores the settings of all other bits because the IPM Manager never includes a copy of the original message in an MSAM report and the IPM Manager may include more than one indication (delivery, non-delivery, or both) in a single report, depending on the number of recipients and other factors.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$051D

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	All allocation blocks on the volume are full
`kOCEParamErr`	–50	Invalid parameter
`kOCEInvalidRef`	–1502	Invalid message reference number
`kOCEInvalidRecipient`	–1514	Invalid recipient
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM, or server MSAM's mail server is shutting down
`kMailInvalidRequest`	–15045	Message reference number refers to a letter

SEE ALSO

The `OCERecipient` structure is described on page 2-106.

The `IPMSender`, `IPMNotificationType`, and `IPMPriority` structures are defined in the chapter "Interprogram Messaging Manager" in *Inside Macintosh: AOCE Application Interfaces*. The chapter also has a discussion of IPM queues.

All of the delivery notification constants are described in the chapter "Interprogram Messaging Manager" in *Inside Macintosh: AOCE Application Interfaces*.

To add a To recipient attribute to your message header, call the `MSAMPutRecipient` function, described on page 2-180.

Writing a Message

To write the various parts of a message, use the functions `MSAMPutBlock`, `MSAMBeginNested`, and `MSAMEndNested`. The functions `MSAMPutContent` and `MSAMPutEnclosure` are used for writing the main content and enclosure portions of letters.