MSAMPutContent

The `MSAMPutContent` function writes the content block of a letter.

```
pascal OSErr MSAMPutContent (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
→	`segmentType`	`MailSegmentType`	Text, picture, sound, movie, or styled text
→	`append`	`Boolean`	Append data to current segment?
↔	`buffer`	`MailBuffer`	Your buffer structure
→	`textScrap`	`StScrpRec*`	Style scrap structure
→	`startNewScript`	`Boolean`	Start a new character set?
→	`script`	`ScriptCode`	Character set

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 content segments. You obtain the reference number when you call the `MSAMCreate` function.
`segmentType`	A value that indicates the segment type of the data that you want to write to the letter. Letter segments may be text, picture, sound, QuickTime movies, or styled text. You can specify only one segment type in this field each time you call the `MSAMPutContent` function. The values that you can specify in this field are described on page 2-109.
`append`	A Boolean value that indicates whether you want the `MSAMPutContent` function to write the data in your buffer to a new segment or append it to an existing segment. Set this field to `false` when you first call the `MSAMPutContent` function to begin writing a new segment. On subsequent calls to the function, set this field to `false` if you want to start a new segment. Set this field to `true` if you want to append the data in your buffer to the segment currently being written by the `MSAMPutContent` 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 the data that you want to write in your buffer. The `MSAMPutContent` 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.
`textScrap`	A pointer to a style scrap structure (`StScrpRec`) that you may provide when you are writing a styled text segment. It contains the style information for the text data in your buffer. Set this field to `nil` if you are not writing a styled text segment.
`startNewScript`	A Boolean value that indicates whether the data in your buffer uses a new character set. You set this field when you are writing either a plain text segment or a styled text segment. Set this field to `true` the first time you call the `MSAMPutContent` function to write the text segment. After that, set this field to `true` only if the text data in your buffer is in a different character set than that which you previously provided to the function. The function ignores this field when you set the `segmentType` field to any value other than `kMailTextSegmentType` or `kMailStyledTextSegmentType`.
`script`	A value that indicates the character set (Roman, Arabic, Kanji, and so on) of the data in your buffer. If you set `startNewScript` to `true`, set this field to the code for the text segment's character set. The `MSAMPutContent` function ignores this field when you set `startNewScript` to `false` or the `segmentType` field to any value other than `kMailTextSegmentType` or `kMailStyledTextSegmentType`.

DESCRIPTION

You call the `MSAMPutContent` function to write data segments in standard interchange format to a content block of a letter that you specify. You must have previously created the letter by calling the `MSAMCreate` function. The first time you call the `MSAMPutContent` function for a given letter, it creates a new block and puts the data into the block. Each time you call the function to add content to the same letter, it adds the data to that same block.

A content block consists of data segments, each of a specific type. You add one segment or a portion of a segment of data each time you call the function. The function writes the segments to the block in the order that you provide them. A single letter may contain more than one segment of a given type.

The IPM Manager does not interpret the data that you write to a segment except when you specify `kMailTextSegmentType` or `kMailStyledTextSegmentType` in the `segmentType` field.

When you write a text segment, you are responsible for establishing the script code of the text. You do this by setting the `startNewScript` field to `true` and the `script` field to the proper script code. A text segment may contain one or more script runs. Therefore, you need to call the `MSAMPutContent` function once for each script run in the segment, setting the `startNewScript` field to `true` and the `script` field to the proper script code for each script run.

The value that you provide in the `script` field must be a valid script in the range 0 to 64. You cannot specify the implicit script codes `smSystemScript` (the system script) and `smCurrentScript` (the font script). If necessary, you can obtain the system script by calling the `GetScriptManagerVariable` function with a selector constant of `smSysScript`. The font script is considered to be the one returned by the `FontScript` function.

When you write a plain text segment (segment type is `kMailTextSegmentType`), the function writes a styled text segment, using the following default values in the `ScrpSTElement` structure that it generates.

Field name	Default value
`scrpStartChar`	0
`scrpHeight`	12
`scrpAscent`	10
`scrpFont`	`monaco` if the script code is `smRoman`. The default value for non-Roman scripts is set to the font family ID of the “first” font within the range for the script.
`scrpFace`	0
`scrpSize`	9
`scrpColor`	{0, 0, 0}

The first font family ID for a non-Roman script is calculated as follows:

- Scripts with script codes in the range 1–32:

  ```
  firstID = 16384 + 512 * (scriptCode – 1)
  ```

- Scripts with script codes in the range 33–64:

  ```
  firstID = –32768 + 512 * (scriptCode – 33)
  ```

To write styled text, you provide a pointer to a style scrap structure in the `textScrap` field. The `scrpNStyles` field in a `StScrpRec` structure indicates the number of `ScrpSTElement` elements that follow. You should allocate a `StScrpRec` structure of a size appropriate to your MSAM. The style information in the style scrap structure applies to the text in your buffer. The IPM Manager uses the text in your buffer and the style information in the style scrap structure to create the segment. You can append additional text to the segment in subsequent calls to the function by providing the text in your buffer, placing the style information that applies to that text in the style scrap structure, and setting the `append` field to `true`.

Specifying `systemFont` or `applFont` in the `scrpFont` field of the `ScrpSTElement` structure is not recommended. If you want to specify the font family ID of the current system font or the current application font, use the functions `GetSysFont` and `GetAppFont` to obtain the appropriate font family ID.

Once you begin writing a letter’s content block, you must not call other MSAM functions until you finish writing the block. Calling a function other than the

`MSAMPutContent` function closes the content portion of the letter. If you then call the `MSAMPutContent` function again, it returns the `kMailInvalidOrder` result code.

It is not necessary to call the `MSAMPutAttribute` and `MSAMPutRecipient` functions prior to calling the `MSAMPutContent` function.

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. To ensure that the right fonts can be applied to the styled text when it is read by a letter application, you can map font numbers to font names when you add styled text to a letter.

Put the mapping of font numbers to font names in a block that has a block creator of `'fish'` and a block type of `'font'`. Then add the block to the letter. The first word in the block must contain 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`	$051A

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
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM, or server MSAM's mail server is shutting down
`kMailInvalidOrder`	–15040	Content already closed
`kMailInvalidRequest`	–15045	Message reference number does not refer to a letter

SEE ALSO

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

See *Inside Macintosh: Text* for more information about script runs, script code constants, style runs, the style scrap structure, and the functions `GetScriptManagerVariable`, `GetSysFont`, and `GetAppFont`.

The segment types that you can specify in the `segmentType` field and the data format for each segment type are described on page 2-109.

MSAMPutEnclosure

The `MSAMPutEnclosure` function adds an enclosure to a letter that you specify.

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

`paramBlock` Pointer to a parameter block.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`mailMsgRef`	`MailMsgRef`	Letter reference number
→	`contentEnclosure`	`Boolean`	Is enclosure main letter content?
→	`hfs`	`Boolean`	Is enclosure in HFS or memory?
→	`append`	`Boolean`	Append data to enclosure?
↔	`buffer`	`MailBuffer`	Your buffer structure
→	`enclosure`	`FSSpec`	File specification
→	`addlInfo`	`MailEnclosureInfo`	Additional enclosure info

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

Field descriptions

`mailMsgRef` A reference number that identifies the letter to which you want to add an enclosure. You obtain this reference number from the `MSAMCreate` function.

`contentEnclosure`
A Boolean value that indicates whether this enclosure contains the main content of the letter. A letter with a content enclosure may or may not contain a content block. A content block contains data in standard interchange format. A content enclosure typically is a file in an application's native format. Given a letter that contains both a content block and a content enclosure, the block and the enclosure are alternate representations of the same basic data.

Set this field to `true` if the enclosure you are adding is a content enclosure. You can identify only one enclosure as a content enclosure for each letter.

`hfs` A Boolean value that indicates the location of the enclosure that you want to add to the letter. Set this field to `true` to indicate that your enclosure is located on disk in the Macintosh file system. Set this field to `false` to indicate that your enclosure resides in memory.

`append` A Boolean value that indicates whether you want the function to append the data in your buffer to the current enclosure. The `MSAMPutEnclosure` function ignores this field when you set the `hfs` field to `true`. When you set the `hfs` field to `false`, set this field to `false` for your first call to the function. Set it to `true` on subsequent calls to continue writing the enclosure.

`buffer`	A `MailBuffer` structure. The `MSAMPutEnclosure` function ignores this field when you set the `hfs` field to `true`. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. You store the enclosure file's resource and data forks in your buffer. The `MSAMPutEnclosure` 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.
`enclosure`	A file system specification record that identifies the file or folder that you want to enclose. You specify this field when the file or folder that you want to enclose is located on disk on either the local computer or a mounted file server volume. The `MSAMPutEnclosure` function ignores this field when the `hfs` field is set to `false`.
`addlInfo`	A structure that you provide to specify file system information for the enclosure, such as the filename, icon, HFS catalog information, and so forth. You provide this information when you add an enclosure that resides in memory. The `MSAMPutEnclosure` function creates a file according to your specifications and puts your data in it. The function ignores this field when you add an enclosure that already exists as a file on disk (when the `hfs` field is set to `true`).

DESCRIPTION

You call the `MSAMPutEnclosure` function to enclose a file, a folder, or both in a letter that you specify. The enclosure that you specify may exist in memory or in the Macintosh hierarchical file system. In the memory form, you provide your enclosure data in buffers, and you specify additional information that defines the filename or file catalog information, and other characteristics of the enclosure. In the HFS form, you supply a path specification to an existing file or folder in the Macintosh file system, and the function encloses that file or folder in the letter.

To enclose a file or folder that resides in the Macintosh Hierarchical File System, set the `enclosure` field to point to the file or folder that you want to enclose. If you set the `enclosure` field to point to a folder, the function encloses the folder and all of the files and folders within it in the letter. Set the `hfs` field to `true` and specify the letter to which you want to add the enclosure in the `mailMsgRef` field. Then call the `MSAMPutEnclosure` function to enclose the file or folder.

To enclose a file that resides in memory, fully specify the `addlInfo` field. Set the `hfs` field to `false`, the `append` field to `false`, and specify the letter to which you want to add the enclosure in the `mailMsgRef` field. Store the enclosure file's resource fork and data fork into your buffer. Always store the resource fork before the data fork. Padding is not required. If a particular fork is empty, do not write any bytes for that fork. Call the `MSAMPutEnclosure` function to write the enclosure data to the letter. The function writes the file data in AppleSingle format. (AppleSingle format accommodates the Macintosh file structure.)

If you have more data to add to the enclosure, set the `append` field to `true` and store additional enclosure data in your buffer. Call the `MSAMPutEnclosure` function to write the enclosure data to the letter. You can repeatedly call the function with new data in your buffer until you have written the entire enclosure file. When the `append` field is set to `true`, the function ignores the `addlInfo` field.

With the memory form, you can enclose a folder instead of a file by specifying file catalog information in the `CInfoPBRec` structure (a component of the `MailEnclosureInfo` structure). Set the `catalog` bit in the `ioFlAttrib` field to identify the enclosure as a folder. In this case, the function ignores the `icon` field in the `MailEnclosureInfo` structure and the `buffer` and `append` fields (because folders don't have data or resource forks).

To enclose a file or a folder within a parent folder using the memory form of the function, first enclose the parent folder. Set the volume reference number (the `ioVRefNum` field in the `CInfoPBRec` structure) of the nested file or folder to the value of the parent folder's volume reference number (`ioVRefNum`) and set the parent folder ID (`ioFlParID`) of the nested file or folder to the parent folder's catalog ID (`ioDirID`).

You can add up to 50 enclosures to a letter, including a content enclosure. Each file and folder that you add counts as one enclosure. For example, if you add as an enclosure a folder containing three files, the total number of enclosures in the letter is four: one folder and three files.

For each letter, you can designate one enclosure as a content enclosure. A content enclosure typically is a file in an application's native format. A letter with a content enclosure may or may not contain a content block. A content block contains data in standard interchange format. Given a letter that contains both a content block and a content enclosure, the block and the enclosure are alternate representations of the same basic data. The standard interchange format content block maximizes the probability that the recipient will be able to read the letter. The application native format content enclosure may provide a richer representation of the basic data, but it can be read only if the recipient has the application. (Image blocks are a third form of letter content. See the discussion on page 2-18 for more information about different representations of letter content.)

IMPORTANT

Although it is technically possible to enclose a folder as a content enclosure, doing so may cause problems with later releases of the AOCE system software that use the services of the Translation Manager. ▲

SPECIAL CONSIDERATIONS

The `MSAMPutEnclosure` function is always executed synchronously.

ASSEMBLY-LANGUAGE INFORMATION

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

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
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM, or server MSAM's mail server is shutting down
`kMailBadEnclLengthErr`	–15044	Invalid data length
`kMailInvalidRequest`	–15045	Nested letter already created for this letter

SEE ALSO

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

The `MailEnclosureInfo` structure is described on page 2-111.

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

The `CInfoPBRec` structure is described in *Inside Macintosh: Files.*

MSAMPutBlock

The `MSAMPutBlock` function adds data to a block in a message.

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

`paramBlock`	Pointer to a parameter block.
`asyncFlag`	A Boolean value that specifies if 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
→	`refCon`	`long`	Reserved for your use
→	`blockType`	`OCECreatorType`	Block type
→	`append`	`Boolean`	Append data to current block?
↔	`buffer`	`MailBuffer`	Your buffer
→	`mode`	`MailBlockMode`	Location of mark in block
→	`offset`	`unsigned long`	Byte offset from mark location

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 write a block. You obtain the reference number when you call the `MSAMCreate` function.
`refCon`	A value reserved for your private use when you add a block to a non-letter message. You may provide a value to be interpreted by the recipient. This field is ignored when you add a block to 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 `kIPMTOC` in the `selector` field of its parameter block.
`blockType`	A structure that specifies the creator and type of the block that you want to write. The `creator` field indicates the creator of the block, for example, `kMailAppleMailCreator` if the block was created by AOCE software. The `type` field identifies the type of block.
`append`	A Boolean value that indicates whether you want the `MSAMPutBlock` function to append the data in your buffer to the current block. Set this field to `false` when you call the function to start a new block. If you set this field to `true`, the function uses the values in the `mode` and `offset` fields to determine where to begin writing to the current block.
`buffer`	A pointer to a `MailBuffer` structure in which you store the data that you want to write to the message that you specify. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. The `MSAMPutBlock` function reads the information that you placed in your buffer and sets the value of the `dataSize` field to the number of bytes of data it wrote into the block.
`mode`	A value that specifies the mode in which the function interprets the `offset` field. The `MSAMPutBlock` function uses the mode and offset to determine where in the current block to begin writing the data from your buffer. The function ignores this field when the value of the `append` field is `false`.
`offset`	A value that specifies an offset that the function uses to determine the starting point of the write operation. Set this field to 0 when you start a new block. The function ignores this field when the value of the `append` field is `false`.

DESCRIPTION

You call the `MSAMPutBlock` function to write data into a block whose type you specify in the `blockType` field. The function writes the data into a new block unless you set the `append` field to `true`.

You use the `mode` and `offset` fields to specify the point in the block at which the `MSAMPutBlock` function starts writing. You can set a variable of type `MailBlockMode` (the `mode` field) to any one of the following values:

```
enum {
   kMailFromStart = 1,
   kMailFromLEOB  = 2,
   kMailFromMark  = 3
};
```

Constant descriptions

`kMailFromStart`	The function interprets the value in the `offset` field as an offset from the beginning of the block. When you use this mode, you cannot set the `offset` field to a negative value.
`kMailFromLEOB`	The function interprets the value in the `offset` field as an offset from the current end of the block. The offset must always be negative and cannot extend beyond the beginning of the block.
`kMailFromMark`	The function interprets the value in the `offset` field as an offset from the current position of the mark. The mark points to the end of the last byte written. Use a 0 offset value to indicate a starting point right at the mark. Use a negative offset value to indicate a starting point prior to the current position of the mark and a positive offset value to indicate a starting point following the current position of the mark. You cannot specify a negative offset that extends beyond the beginning of the block.

If your buffer is too small to hold all of the data that you want to write to a block, you can call the function repeatedly until you have written the entire block. The first time you call the function, set the `append` field to `false`, the `mode` field to `kMailFromStart`, and the `offset` field to 0. On subsequent calls to write additional data to the same block, set the `append` field to `true`, the `mode` field to `kMailFromMark`, and the `offset` field to 0.

You can overwrite data you have already written to a block, but cannot modify a completed block once you start a new block.

Once you begin writing a block, you must not call other MSAM functions until you finish writing the block. Calling a function other than `MSAMPutBlock` closes the current block.

Typically, you call the `MSAMPutBlock` function to write image blocks (block type is `kMailImageBodyType`) or private blocks (block type is `kMailMSAMType`) because the MSAM API provides no other way to write these types of blocks. Although it is possible to call the `MSAMPutBlock` function to write blocks that contain letter content, attributes, enclosures, and so forth, you should use the specific functions provided for writing that type of information.

The `kMailMSAMType` block type indicates a block whose format and content are private to the MSAM. If you add a private block to a message, AOCE software includes the private block when it generates a report on the message.

If you are adding an image block to a message, you provide the block's data in the format of a `TPfPgDir` structure, followed by the picture elements (PICTs).

ASSEMBLY-LANGUAGE INFORMATION

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

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
`kIPMMsgTypeReserved`	–1511	Message creator and/or type specified not allowed
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM, or server MSAM's mail server is shutting down

SEE ALSO

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

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

MSAMBeginNested

The `MSAMBeginNested` function begins the process of creating a nested message.

```
pascal OSErr MSAMBeginNested (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
→	`refCon`	`long`	Reserved for your use
→	`msgType`	`IPMMsgType`	Message type of nested message

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

Field descriptions

`mailMsgRef`	A reference number that identifies the message to which you want to add a nested message. You obtain the reference number when you call the `MSAMCreate` function.
`refCon`	A value reserved for your private use when you create a non-letter nested message. You may provide a value to be interpreted by the recipient. This field is ignored when you create a nested letter.
`msgType`	The creator and type of the nested message that you are creating.

DESCRIPTION

You call the `MSAMBeginNested` function to begin the process of creating a message nested within a message that you have already created but not yet submitted for delivery. The function increments the nesting level of the existing message. All subsequent calls that you make to `MSAMPut` functions refer to this nesting level until you call either the `MSAMEndNested` function or the `MSAMBeginNested` function. You can call the `MSAMBeginNested` function repeatedly to create a hierarchy of nested messages, but you cannot create more than one nested message per nesting level.

If you provide a value in the `refCon` field when you create a non-letter nested message, it is stored in its message header. The recipient can retrieve the value by calling the `MSAMOpenNested` function to obtain the nested message's reference number and then calling the `MSAMGetMsgHeader` function, specifying that reference number and setting the `selector` field of its parameter block to `kIPMFixedInfo`.

▲ **WARNING**
You cannot delete the nested portion of a message once you put data (recipients, blocks, enclosures, and so on) in it. Furthermore, an empty nested message is not allowed. If you call the `MSAMEndNested` function immediately after you call the `MSAMBeginNested` function, the function returns the `kMailHdrAttrMissing` result code, indicating that the nested message is incomplete. In this case, the function deletes the *entire* message, not just the nested message. ▲

SPECIAL CONSIDERATIONS

You do not get a separate reference number for a nested message. You always use the reference number of the outermost message when adding any kind of data to a nested message, regardless of how deeply it is nested.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0515

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	All allocation blocks on the volume are full
`memFullErr`	–108	Not enough memory
`kOCEInvalidRef`	–1502	Invalid message reference number
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM, or server MSAM's mail server is shutting down
`kMailHdrAttrMissing`	–15043	Required attribute not written into header
`kMailInvalidRequest`	–15045	Nested letter already created for this letter

SEE ALSO

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

MSAMEndNested

The `MSAMEndNested` function ends the nested message currently being written.

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

`paramBlock` Pointer to a parameter block.

Parameter block

←	`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 contains the message letter that you want to end. You obtain the reference number when you call the `MSAMCreate` function.

DESCRIPTION

You call the `MSAMEndNested` function to indicate that you have finished constructing your nested message. After the function successfully completes, you cannot make any additions to the nested message. Subsequent calls that you make to *MSAMPut* functions apply to the next higher nesting level.

▲ **WARNING**
An empty nested message is not allowed. If you call the `MSAMEndNested` function immediately after you call the `MSAMBeginNested` function, the `MSAMEndNested` function returns the `kMailHdrAttrMissing` result code, indicating that the nested message is incomplete. In this case, `MSAMEndNested` deletes the *entire* message, not just the nested message. ▲

SPECIAL CONSIDERATIONS

This function is always executed synchronously.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0516

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	All allocation blocks on the volume are full
`kOCEParamErr`	–50	Invalid parameter
`memFullErr`	–108	Not enough memory
`kOCEInvalidRef`	–1502	Invalid message reference number
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM, or server MSAM's mail server is shutting down
`kMailHdrAttrMissing`	–15043	Required attribute not added to message
`kMailBadEnclLengthErr`	–15044	Number of bytes written not equal to number of bytes needed for `memForm` enclosure in progress

SEE ALSO

The `MSAMBeginNested` function is described on page 2-196.

Submitting a Message

When you have finished composing a letter, report, or non-letter message, use the function `MSAMSubmit` to submit it for delivery into the AOCE system.

MSAMSubmit

The `MSAMSubmit` function submits a completed letter, report, or non-letter message for delivery to the addressee or requests that it be deleted.

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

`paramBlock` Pointer to a parameter block.

Parameter block

←	`ioResult`	`OSErr`	Result code
→	`mailMsgRef`	`MailMsgRef`	Message reference number
→	`submitFlag`	`Boolean`	Submit or delete message?

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

Field descriptions

`mailMsgRef`	A reference number that identifies the message to which the request applies. You obtain the reference number when you call the `MSAMCreate` function.
`submitFlag`	A Boolean value that indicates whether you want the `MSAMSubmit` function to accept the message that you specify for delivery or to delete it. Set this field to `true` to indicate that the message is complete and ready for delivery. Set this field to `false` if you want the function to delete the message.

DESCRIPTION

You call the `MSAMSubmit` function to request delivery of a incoming message to an AOCE addressee or to request that the message be deleted.

A message must be complete at the time you call the `MSAMSubmit` function to submit the message for delivery. To be complete, you must have added to the message header at least a `to`, a `from`, and a `sendTimeStamp` attribute. You should also add all nested messages, enclosures (letters only), blocks, content (letters only), attributes, and recipients before you submit the message for delivery. After you call the `MSAMSubmit` function, the message reference number is invalid and you can make no further changes to the message.

You can call the `MSAMSubmit` function to delete a message at any time after you create the message.

If you submit a message to which you did not add a `msgFamily` attribute, AOCE software adds a `msgFamily` attribute and sets it to `kIPMFamilyUnspecified` for a non-letter message and to `kMailFamily` for a letter. If you submit a letter to which you did not add an `indications` attribute, AOCE software adds it and sets the `priority` bit field to `kIPMNormalPriority` and all of the other bit fields to 0.

If a personal MSAM sets the `submitFlag` field to `false` for a letter, the function deletes the letter, but not the letter's message summary. To delete a letter's message summary, call the `MSAMDelete` function.

SPECIAL CONSIDERATIONS

The `MSAMSubmit` function is always executed synchronously.

Because it normally has continuous access to the PowerShare mail server, a server MSAM should translate incoming messages immediately and submit them to the PowerShare mail server. If the PowerShare mail server quits, the server MSAM should either stop accepting incoming messages or store the incoming messages until the PowerShare mail server is available again.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0517

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	All allocation blocks on the volume are full
`kOCEParamErr`	–50	Invalid parameter
`memFullErr`	–108	Not enough memory
`kOCEInvalidRef`	–1502	Invalid message reference number
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM, or server MSAM's mail server is shutting down
`kMailHdrAttrMissing`	–15043	Required attribute not added to message
`kMailBadEnclLengthErr`	–15044	Number of bytes written not equal to number of bytes needed for `memForm` enclosure in progress

SEE ALSO

Methods of detecting when a PowerShare mail server quits and starts are discussed on page 2-42.

The `MSAMDelete` function is described next.

Letter attributes and the `MailIndications` data type are described on page 2-100 and page 2-102, respectively.

Deleting a Message

A server MSAM uses the `MSAMDelete` function to delete a message from its outgoing queue. A personal MSAM uses the function to delete letters and message summaries from its incoming queues.

MSAMDelete

The `MSAMDelete` function deletes a message from a queue that you specify.

```
pascal OSErr MSAMDelete (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 the queue
→	`msgOnly`	`Boolean`	Delete letter, not message summary?
→	`result`	`OSErr`	Reserved

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

Field descriptions

`queueRef`	The queue that contains the message that you want to delete. A personal MSAM may specify either an outgoing queue reference or an incoming queue reference. It obtains queue references from the `PMSAMOpenQueues` function. A server MSAM specifies the queue reference that it obtained from the `SMSAMStartup` function, which refers to its outgoing queue.
`seqNum`	The sequence number that identifies the message that you want to delete. You obtain this value from the `MSAMEnumerate` function.
`msgOnly`	A Boolean value that indicates whether a personal MSAM wants to delete only a letter or both a letter and its message summary from an incoming queue. You set this field to `true` if you want to delete only the letter itself. If you set this field to `false`, you delete both the letter and its associated message summary. A personal MSAM that is deleting a letter from an outgoing queue, and all server MSAMs, should set this field to `false`.
`result`	Reserved. Set this field to the `noErr` result code.

DESCRIPTION

You call the `MSAMDelete` function to delete a message that you specify. You identify the message by its sequence number. Once you have deleted a message, it is no longer available to you on the local computer.

Generally, a personal MSAM should not call this function to delete a letter from an outgoing queue. Instead, it should leave letters in an outgoing queue so that the user can peruse them. An exception to this rule occurs when a user wants to delete a letter rather than send it. In that case, the IPM Manager sends the personal MSAM a `kMailEPPCDeleteOutQMsg` event, and the personal MSAM should delete the letter.

A server MSAM calls this function to delete messages from its outgoing queue.

The `MSAMDelete` function allows a personal MSAM to delete a letter, with or without the message summary, from an incoming queue. For example, it may want to delete a letter, but not the message summary, when it decides the letter no longer needs to be cached locally. If the personal MSAM is trying to mirror the letter's status on its external messaging system, it can delete the letter and the message summary when the letter is removed from the external messaging system. If a personal MSAM sets the `msgOnly` field to `false` and only the message summary is present in the queue, the function deletes it and returns the `noErr` result code.

The `MSAMDelete` function closes a message if it is open.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0504

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	All allocation blocks on the volume are full
`kOCEParamErr`	–50	Invalid parameter
`memFullErr`	–108	Not enough memory
`kOCEInvalidRef`	–1502	Invalid message reference number
`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

Message summaries are discussed in the section "MSAM Modes of Operation" beginning on page 2-12.

The IPM Manager may also delete a letter from a personal MSAM's incoming queue in response to a user action. In that case, it sets the `msgDeleted` flag in the letter's message summary and sends the `kMailEPPCInQUpdate` event. The `kMailEPPCInQUpdate` event is described on page 2-228.

The `kMailEPPCDeleteOutQMsg` event is described on page 2-231.

Generating Log Entries and Reports

A personal MSAM may run into operational problems. Use the function `PMSAMLogError` to log such problems.

Use `MSAMCreateReport` and `MSAMPutRecipientReport` to create delivery and non-delivery reports when the originator of a message has requested them.

PMSAMLogError

The `PMSAMLogError` function reports operational errors in a personal MSAM.

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

`paramBlock` Pointer to a parameter block.

Parameter block

←	`ioResult`	`OSErr`	Result code
→	`msamSlotID`	`MSAMSlotID`	Personal MSAM or slot ID
→	`logEntry`	`MailErrorLogEntryInfo*`	Error log record

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

Field descriptions

`msamSlotID`	A value that indicates whether the error you are logging applies to the personal MSAM as a whole or to one of its slots. Set this field to 0 to indicate that the error applies to the personal MSAM. Otherwise, set it to the slot ID of the slot to which the error applies.
`logEntry`	A pointer to a `MailErrorLogEntryInfo` structure that contains information about the error that you are logging.

DESCRIPTION

You call the `PMSAMLogError` function to log information about an operational error in a personal MSAM or in one of its slots. In some cases, you also log suggested actions a user can take to correct the problem.

To log an error, you must provide values in the `version`, `errorType`, and `errorCode` fields of the `MailErrorLogEntryInfo` structure. In addition, you must fill in the `errorResource` field if the `errorCode` field has the value `kMailMSAMErrorCode`, and you must fill in the `actionResource` field if the `errorType` field has the value `kMailELECorrectable`.

Errors of type `kMailELEError`, `kMailELEWarning`, and `kMailELEInformational` either require no user intervention or cannot be corrected by user intervention. Errors of type `kMailELECorrectable` do require user intervention to correct the problem.

When you log a correctable error (`kMailELECorrectable`), the IPM Manager considers either the personal MSAM or one of its slots to be suspended. While the personal MSAM is suspended, the IPM Manager does not send it any high-level events or restart it at scheduled times if it quits. While a slot is suspended, the user cannot modify or delete it. Moreover, if you specify the suspended slot in a call to the `PMSAMOpenQueues` function, it returns the `kMailSlotSuspended` result code. Other than these exceptions, a personal MSAM can continue whatever activity it deems appropriate while it or one of its slots is suspended. The IPM Manager reinstates a suspended personal MSAM or a slot when the user informs the IPM Manager that the error is corrected or when the computer on which the personal MSAM is running is restarted. If the personal MSAM is not running when the error is marked as corrected, the IPM Manager launches it. If the personal MSAM is running, it receives an `kMailEPPCContinue` high-level event.

Because logging a correctable error implies that the problem is not transient in nature, the `PMSAMLogError` function does not provide you with a mechanism for canceling correctable errors or accessing logged entries. Also, because correctable errors by definition require a user's attention, you should not log them unless absolutely necessary.

You can supply your own error messages. To do so, you must set the `errorCode` field to `kMailMSAMErrorCode`. You must also set the `errorResource` field in the `MailErrorLogEntryInfo` structure. This field is an index into a list of error messages. The list is a '`STR#`' (string list) resource in the personal MSAM's resource file. The first index into the string list is 1. The resource ID for the string list is `kMailMSAMErrorStringListID`. This method ensures that all error messages are localizable.

When the value of `errorType` is `kMailELECorrectable`, you must specify an action that a user should take to correct the error. The procedure is the same as the one just described for MSAM-defined error messages, except that the resource ID of the string list is `kMailMSAMActionStringListID` and the field that you set is `actionResource`.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0521

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	All allocation blocks on the volume are full
`kOCEParamErr`	–50	Invalid parameter
`memFullErr`	–108	Not enough memory
`kOCEInvalidRef`	–1502	Invalid queue reference
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM
`kMailNoMSAMErr`	–15056	No such MSAM
`kMailNoSuchSlot`	–15062	No such slot

SEE ALSO

The `MailErrorLogEntryInfo` structure is described on page 2-128.

See the section "Logging Personal MSAM Operational Errors" on page 2-91 for more information about logging operational errors.

MSAMCreateReport

The `MSAMCreateReport` function creates a report about a message that you specify and returns a reference number for the report.

```
pascal OSErr MSAMCreateReport (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
←	`mailMsgRef`	`MailMsgRef`	Report reference number
→	`msgID`	`MailLetterID`	Message the report applies to
→	`sender`	`MailRecipient*`	Sender of the message

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

Field descriptions

`queueRef`	A reference number that identifies the queue from which the MSAM read the message about which it is reporting. A personal MSAM specifies an outgoing queue reference that it obtained from the `PMSAMOpenQueues` function. A server MSAM specifies the queue reference that it obtained from the `SMSAMStartup` function.
`mailMsgRef`	A reference number that identifies the report that you create. The `MSAMCreateReport` function returns this to you upon successfully completing execution.
`msgID`	A value that identifies the message about which you want to create a report. If the message is a letter, you provide the letter's letter ID attribute. If it is a non-letter message, you provide the message ID from the message header's fixed information.
`sender`	A pointer to a `MailRecipient` structure that contains the address of the sender of the message about which you want to report. If the message is a letter, you provide the value of the letter's From recipient. If it is a non-letter message, you provide the value of the reply queue address in the message header.

DESCRIPTION

You call the `MSAMCreateReport` function to create a report about a message that you are responsible for delivering. Use the `MSAMPutRecipientReport` function to fill in the report information.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$051F

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	All allocation blocks on the volume are full
`kOCEParamErr`	–50	Invalid parameter
`memFullErr`	–108	Not enough memory
`kOCEInvalidRef`	–1502	Invalid queue reference
`kOCEInvalidRecipient`	–1514	Bad recipient
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM, or server MSAM's mail server is shutting down

SEE ALSO

The `MailRecipient` structure is defined to be of type `OCERecipient`. The `OCERecipient` structure is described on page 2-106.

You get the value of the reply queue address in the message header by calling the `MSAMGetMsgHeader` function with the `selector` field set to `kIPMSender`. The `MSAMGetMsgHeader` function is described on page 2-148.

The section "Generating a Report" beginning on page 2-61 explains how to determine when you are required to create a report.

MSAMPutRecipientReport

The `MSAMPutRecipientReport` function adds information about one recipient to a report.

```
pascal OSErr MSAMPutRecipientReport (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`	Report reference number
→	`recipientIndex`	`short`	Message recipient
→	`result`	`OSErr`	Result of delivery attempt

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 report to which you want to add recipient information. You obtain this reference number from the `MSAMCreateReport` function.
`recipientIndex`	A value that identifies the recipient about which you are reporting. You obtain this value from the `index` field of the `MailResolvedRecipient` structure returned by the `MSAMGetRecipients` function.
`result`	A value that indicates the result of your delivery attempts. The constants that you may use here are described below.

DESCRIPTION

You call the `MSAMPutRecipientReport` function to report on the result of your attempt to deliver a message to a recipient that you specify. You can specify only one recipient to the `MSAMPutRecipientReport` function. To report on more than one recipient, make multiple calls to the function. Use the report reference number that you obtained from the `MSAMCreateReport` function to associate your recipient report information with a particular report. When you have finished adding recipient information to the report, you must call the `MSAMSubmit` function to request delivery of the report.

The `result` field contains either a delivery or a non-delivery indication for a given recipient. Set the `result` field to `noErr` to add a delivery indication. The values you can use for a non-delivery indication are described in the following list:

Constant descriptions

`kIPMNoSuchRecipient`	The recipient does not exist.
`kIPMRecipientMalFormed`	The address is malformed. An MSAM detects an invalid extension value.
`kIPMRecipientAmbiguous`	The MSAM is unable to resolve, look up, or find the specified recipient.
`kIPMRecipientAccessDenied`	The recipient probably exists and may be valid, but the MSAM doesn't have access to deliver the message.
`kIPMGroupExpansionProblem`	The MSAM was unable to expand a group address completely. It may have delivered the message to some of the recipients in the group address.
`kIPMMsgUnreadable`	The MSAM cannot read the message; it's corrupted or missing.

`kIPMMsgExpired`	The MSAM's time limit ran out before it was able to confirm delivery of the message to the specified recipient. Note that this does not mean that the message was not successfully delivered to the recipient.
`kIPMMsgNoTranslatibleContent`	The message is missing information that is considered critical to its delivery—for example, there is no subject, no content, or no image content (for a fax MSAM).
`kIPMRecipientReqStdCont`	The MSAM could not deliver the message to a particular recipient because the message did not contain a required standard interchange format block.
`kIPMRecipientReqSnapShot`	The MSAM could not deliver the message to a particular recipient because the message did not contain a required snapshot (image) format block.
`kIPMNoTransferDiskFull`	The destination system refused delivery because of a disk/system full condition.
`kIPMNoTransferMsgRejectedbyDest`	The destination system refused delivery for an unspecified reason.
`kIPMNoTransferMsgTooLarge`	The destination system refused delivery because the message exceeded the maximum size limit for messages in that system.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0520

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	All allocation blocks on the volume are full
`kOCEParamErr`	–50	Invalid parameter
`memFullErr`	–108	Not enough memory
`kOCEInvalidRef`	–1502	Invalid message reference number
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM, or server MSAM's mail server is shutting down
`kMailInvalidRequest`	–15045	Nested letter already created for this letter

SEE ALSO

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

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

The `MSAMSubmit` function is described on page 2-200.

For more information about adding delivery or non-delivery indications to a report, see the section "Generating a Report" on page 2-61.

The non-delivery indication constants for use in the `result` field are also documented in the chapter "Interprogram Messaging Manager" in *Inside Macintosh: AOCE Application Interfaces*.

Shutting Down a Server MSAM

A server MSAM calls the `SMSAMShutdown` function to notify its PowerShare mail server that it is shutting down.

SMSAMShutdown

The `SMSAMShutdown` function informs a PowerShare mail server that a server MSAM is shutting down.

```
pascal OSErr SMSAMShutdown (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`	Outgoing queue reference

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

Field descriptions

`queueRef`	A value that identifies the queue belonging to the server MSAM that is shutting down. Set this field to the queue reference value you obtained from the `SMSAMStartup` function.

DESCRIPTION

You call the `SMSAMShutdown` function as part of the process of shutting down a server MSAM. The queue reference is not valid after the function successfully completes.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0502

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	All allocation blocks on the volume are full
`kOCEInvalidRef`	–1502	Invalid queue reference
`kOCERefIsClosing`	–1516	Server MSAM's mail server is shutting down

Setting Message Status

A personal MSAM calls the `PMSAMSetStatus` function to set the status of a message in a queue.

PMSAMSetStatus

The `PMSAMSetStatus` function sets the status of a message in a queue.

```
pascal OSErr PMSAMSetStatus (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`	ID number of queue
→	`seqNum`	`long`	Message sequence number
→	`msgHint`	`long`	Letter reference value
→	`status`	`PMSAMStatus`	Status to set

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 that holds the message whose status you want to set.
`seqNum`	The sequence number of the message whose status you want to set. For an outgoing message, you obtain the sequence number of a message from the `MSAMEnumerateOutQReply` structure returned by the `MSAMEnumerate` function. For an incoming letter, you obtain the sequence number either from the `MSAMEnumerateInQReply` structure returned by the `MSAMEnumerate` function or from the `SMCA` structure associated with a `kMailEPPCMsgOpened` event.

`msgHint`	A reference value associated with a letter. You set this field to the reference value when you are reporting a problem with retrieving a letter that the user has opened. You obtain this value from the `SMCA` structure associated with a `kMailEPPCMsgOpened` event. Set this field to 0 when you are reporting status for a letter in an outgoing queue.
`status`	The status that you want to set.

DESCRIPTION

A personal MSAM calls the `PMSAMSetStatus` function to set the status of a message.

You call the function to set the status of a letter in an incoming queue after you have received a `kMailEPPCMsgOpened` high-level event for that letter. The Finder uses the status information that you provide to display the status of the letter to the user. To provide an acceptable response time for the user, it is very important that you call the `PMSAMSetStatus` function in a timely manner. Note that you set the status only for incoming letters, not non-letter messages.

You set the status of all messages in an outgoing queue. You call the `PMSAMSetStatus` function as a result of your personal MSAM's handling of the message. The Finder uses the status information that you provide to display the status of outgoing letters to the user. It is important to call the `PMSAMSetStatus` function in a timely manner for outgoing messages, although it is not as critical as it is with incoming letters. With incoming letters, you must respond to a user action; with outgoing messages, you do not.

The following table describes the status settings:

Constant	**Value**	**Description**
`kPMSAMStatusPending`	1	Applies to all types of messages in the outgoing queue. Set this status when you have not yet tried to deliver a message, or when you have tried and failed but will try again.
`kPMSAMStatusError`	2	Applies to letters in an incoming queue. Set this status when you have failed to retrieve a letter from the external messaging system and to write it to the incoming queue.
`kPMSAMStatusSending`	3	Applies to all types of messages in the outgoing queue. Set this status to indicate that you are in the process of sending the message.
`kPMSAMStatusCaching`	4	Applies to letters in the incoming queue. Set this status to indicate that you are in the process of writing the letter into the incoming queue.
`kPMSAMStatusSent`	5	You do not set this status. When all of the recipients of a message in the outgoing queue have been marked as delivered, the IPM Manager sets this status for the message.

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
`memFullErr`	–108	Not enough memory
`kOCEInvalidRef`	–1502	Invalid queue reference number
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM
`kMailInvalidSeqNum`	–15041	Invalid message sequence number
`kMailNotASlotInQ`	–15047	If you set `msgHint`, it does not refer to a slot's incoming queue
`kMailBadState`	–15068	Invalid status setting

Personal MSAM Template Functions

The functions described in this section are called not by a personal MSAM itself, but by its AOCE setup template.

MailCreateMailSlot

The `MailCreateMailSlot` function creates a new mail slot.

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

`paramBlock` Pointer to a parameter block.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`mailboxRef`	`MailboxRef`	Reserved
→	`timeout`	`long`	Timeout interval
→	`pmsamCid`	`CreationID`	Creation ID of personal MSAM record
↔	`smca`	`SMCA`	Shared communications area

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

Field descriptions

`mailboxRef`	Reserved. Set this field to 0.
`timeout`	The amount of time, expressed in ticks, that you are willing to wait for a response from the personal MSAM. It is recommended that you set the timeout period to be a number of seconds. If the timeout period elapses without a response from the personal MSAM, the function completes with a `noRelErr` result code.

`pmsamCid`	The creation ID of the MSAM record, which represents the personal MSAM to which you want to add a mail slot.
`smca`	An `SMCA` structure. You set the `slotCID` field to the creation ID of the Mail Service or Combined record, which contains information about the newly created mail slot. The IPM Manager sets the `result` field to 1 before sending the `kMailEPPCCreateSlot` high-level event to the personal MSAM. When the `MailCreateMailSlot` function completes, the `result` field contains the MSAM's result, if the personal MSAM has processed the `kMailEPPCCreateSlot` event. Otherwise, it still contains 1.

DESCRIPTION

Your setup template calls the `MailCreateMailSlot` function to add a new mail slot to a personal MSAM. This causes the IPM Manager to send a `kMailEPPCCreateSlot` high-level event to the personal MSAM.

Do not poll the `smca.result` field to determine when the function has completed. If you poll, poll the `ioResult` field. Then check the value of the `smca.result` field.

If the MSAM responds to the event, the `MailCreateMailSlot` function completes with the `noErr` result code, regardless of the value of the `smca.result` field. Therefore, you should always check the value of the `smca.result` field to get the result of the MSAM's processing of the event. You cannot assume that if the `MailCreateMailSlot` function returns `noErr`, the MSAM also reported no error.

If the personal MSAM is not running at the time the associated template calls this function, the IPM Manager launches the MSAM before sending it the `kMailEPPCCreateSlot` event.

SPECIAL CONSIDERATIONS

The `MailCreateMailSlot` function is always executed asynchronously. After calling `MailCreateMailSlot`, you should call the `kDETcmdBusy` callback routine to provide time for the personal MSAM to receive and respond to the `kMailEPPCCreateSlot` high-level event.

Your template does not need to delete a mail slot. The AOCE software deletes a mail slot in response to a user action.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$052B

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	All allocation blocks on the volume are full
`kOCEParamErr`	–50	Invalid parameter
`memFullErr`	–108	Not enough memory
`noRelErr`	–1101	Timer expired before MSAM responded
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM
`kMailIgnoredErr`	–15053	MSAM ignored high-level event
`kMailLengthErr`	–15054	Error occurred in sending the event
`kMailTooManyErr`	–15055	IPM Manager too busy to send event
`kMailNoMSAMErr`	–15056	No such MSAM
`kMailMSAMSuspended`	–15059	MSAM is suspended
`kMailBadSlotInfo`	–15060	Invalid slot information

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 information about the personal MSAM's record.

The `kMailEPPCCreateSlot` high-level event is described on page 2-221.

The `kDETcmdBusy` callback routine is described in the chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces*.

MailModifyMailSlot

The `MailModifyMailSlot` function modifies the information in a mail slot.

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

`paramBlock` Pointer to a parameter block.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`mailboxRef`	`MailboxRef`	Reserved
→	`timeout`	`long`	Timeout interval
→	`pmsamCid`	`CreationID`	Creation ID of personal MSAM record
↔	`smca`	`SMCA`	Shared communications area

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

Field descriptions

`mailboxRef`	Reserved. Set this field to 0.
`timeout`	The amount of time, expressed in ticks, that you are willing to wait for a response from the personal MSAM. It is recommended that you set the timeout period to be a number of seconds. If the timeout period elapses without a response from the personal MSAM, the function completes with a `noRelErr` result code.
`pmsamCid`	The creation ID of the MSAM record, which represents the personal MSAM whose mail slot you want to modify.
`smca`	An `SMCA` structure. You set the `slotCID` field to the creation ID of the new Mail Service or Combined record, which contains information about the modified mail slot. The IPM Manager sets the `result` field to 1 before sending the `kMailEPPCModifySlot` high-level event to the personal MSAM. When the function completes, if the personal MSAM has processed the `kMailEPPCModifySlot` event, the `result` field contains the MSAM's result. Otherwise, it still contains 1.

DESCRIPTION

Your setup template calls the `MailModifyMailSlot` function to change the information in a mail slot. This causes the IPM Manager to send a `kMailEPPCModifySlot` high-level event to the personal MSAM. You invoke the function after you have created a new Mail Service record in the Setup catalog that contains the changed information.

Do not poll the `smca.result` field to determine when the function has completed. If you poll, poll the `ioResult` field. Then check the value of the `smca.result` field.

If the MSAM responds to the event, the `MailModifyMailSlot` function completes with the `noErr` result code, regardless of the value of the `smca.result` field. Therefore, you should always check the value of the `smca.result` field to get the result of the MSAM's processing of the event. You cannot assume that if the `MailModifyMailSlot` function returns `noErr`, the MSAM also reported no error.

If the MSAM specifies `noErr` in the `result` field of the `SMCA` structure, you should delete the old Mail Service record and update the slot attribute (attribute type index is `kMailServiceAttrTypeNum`) in the MSAM record in the Setup catalog to point to the new Mail Service record. If the MSAM reports an error, you should leave the original Mail Service record intact, delete the new Mail Service record, and report the error to the user.

SPECIAL CONSIDERATIONS

The `MailModifyMailSlot` function is always executed asynchronously. After calling `MailModifyMailSlot`, you should call the `kDETcmdBusy` callback routine to provide time for the personal MSAM to receive and respond to the `kMailEPPCModifySlot` high-level event.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$052C

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	All allocation blocks on the volume are full
`kOCEParamErr`	–50	Invalid parameter
`noRelErr`	–1101	Timer expired before MSAM responded
`kOCERefIsClosing`	–1516	IPM Manager is shutting down the personal MSAM
`kMailIgnoredErr`	–15053	MSAM ignored high-level event
`kMailLengthErr`	–15054	Error in sending the event
`kMailTooManyErr`	–15055	IPM Manager too busy to send event
`kMailNoMSAMErr`	–15056	No such MSAM
`kMailNoSuchSlot`	–15062	No such slot

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 information about the personal MSAM's record, Mail Service records, and the Setup catalog.

The `kDETcmdBusy` callback routine is described in the chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces*.

MailWakeupPMSAM

The `MailWakeupPMSAM` function causes the IPM Manager to send a `kMailEPPCWakeup` event to the personal MSAM that you specify.

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

`paramBlock` Pointer to a parameter block.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`pmsamCid`	`CreationID`	Record ID of MSAM record
→	`mailSlotID`	`MailSlotID`	Reserved

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

Field descriptions

`pmsamCid`	The creation ID of the MSAM record in the Setup catalog that represents the personal MSAM you want to launch.
`mailSlotID`	Reserved. Set this field to 0.

DESCRIPTION

You call the `MailWakeupPMSAM` function to request that the IPM Manager send a `kMailEPPCWakeup` event to the personal MSAM that you specify.

Typically, you call this function in response to unpredictable events that require action by the MSAM. For example, a fax modem driver might call the `MailWakeupPMSAM` function when it receives an incoming call so that the MSAM can put the letter in the incoming queue.

If the MSAM is not running at the time you call the `MailWakeupPMSAM` function, the IPM Manager launches it.

The `kMailEPPCWakeup` event is not infallible. Therefore, you cannot count on it as a mechanism to force something to happen. However, the IPM Manager makes every attempt to inform you of possible failures so that you can retry the operation if you wish.

SPECIAL CONSIDERATIONS

The `MailWakeupPMSAM` function is always executed asynchronously. After calling `MailWakeupPMSAM`, you must call the `WaitNextEvent` function, which provides time for the personal MSAM to be launched.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0507

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	All allocation blocks on the volume are full
`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 about the personal MSAM’s record.

Application-Defined Function

This section describes the completion routine that you may provide when you call a function in the MSAM API asynchronously.

MyCompletionRoutine

When you call an MSAM API function asynchronously, you can provide a pointer to a completion routine.

```
void MyCompletionRoutine (MSAMParam *paramBlock);
```

`paramBlock`	A pointer to the parameter block that you provided when you called the MSAM function that is calling your completion routine.

DESCRIPTION

You can provide a completion routine to any MSAM function that you can call asynchronously. To do so, you pass a pointer to the completion routine in the `ioCompletion` field of the `MSAMParam` parameter block. If you provide a completion routine, it executes when the asynchronous request completes execution.

The MSAM function saves the value of your A5 register at the time you call it and then restores the A5 value before it calls your completion routine. Your completion routine is always called at deferred-task time. Running at deferred-task time is a safe practice when you use virtual memory.

You can write your completion routine in C, Pascal, or assembly language.

To declare a completion routine in Pascal, use the following statement:

```
PROCEDURE MyCompletionRoutine(VAR paramBlock: MSAMParam);
```

Note that if you do not want to specify a completion routine for an asynchronous function call, you can specify `nil` in the `ioCompletion` field and poll the `ioResult` field of the parameter block header. When you call an MSAM function asynchronously, it sets the `ioResult` field in the parameter block to 1 to indicate that the routine has not yet completed execution. When the routine completes execution, the MSAM function sets the `ioResult` field to the actual function result. If you poll, you should do so within a loop that calls either the `WaitNextEvent` or `EventAvail` routine so that other processes have access to processor time.

ASSEMBLY-LANGUAGE INFORMATION

When a completion routine written in assembly language is called, register A0 contains a pointer to the `MSAMParam` parameter block, and register D0 contains the MSAM function result code (also available in the `ioResult` field of the parameter block). The condition codes are set as a result of TST.W D0.

You cannot make any other assumptions about any part of your environment, including, but not limited to

- the stack pointer and register A6
- registers A2, A3, and A4
- low-memory global variables

You must preserve all registers except D0, D1, D2, A0, and A1.

High-Level Events

This section contains descriptions of the AOCE high-level events that an MSAM may receive. Server MSAMs may receive the `kMailEPPCAdmin` and `kMailEPPCMsgPending` high-level events. Personal MSAMs receive the `kMailEPPCMsgPending` event as well as a number of others. You can find a complete list of the events sent to personal and server MSAMs on page 2-32.

Each event description in this section provides a description of the `where` and `modifiers` fields of the event record. The `what`, `message`, and `when` field descriptions are the same for every event. They are provided here; this information is not repeated in the individual event descriptions.

Field name	Data type	Description
`what`	`short`	Always contains the constant `kHighLevelEvent`.
`message`	`long`	Always contains the event class `kMailAppleMailCreator`.
`when`	`long`	Unused.

Certain events require more information than can be passed in the event record. For these events, the MSAM obtains the additional information it needs by calling the `AcceptHighLevelEvent` function. If an event requires no additional information, an MSAM does not need to call the `AcceptHighLevelEvent` function.

The `AcceptHighLevelEvent` function returns a `MailEPPCMsg` structure that contains one of the following:

- a pointer to an `SMCA` structure
- a letter sequence number
- a `MailLocationInfo` structure

Where it applies, the event descriptions in this section include a description of the sequence number or the relevant fields of the `SMCA` or `MailLocationInfo` structure. The `SMCA` structure is described on page 2-114. The `MailLocationInfo` structure is described on page 2-116.

kMailEPPCCreateSlot

The `kMailEPPCCreateSlot` event informs a personal MSAM that the MSAM's template has added a new Mail Service or Combined record to the Setup catalog.

EVENT RECORD

Field name	Data type	Description
`where`	`long`	The constant `kMailEPPCCreateSlot`.
`modifiers`	`short`	Unused; contains 0.

MailEPPCMsg STRUCTURE

Field name	Data type	Description
`u.theSMCA->result`	`OSErr`	The result of performing the activity requested by the `kMailEPPCCreateSlot` event. When the personal MSAM receives the `kMailEPPCCreateSlot` event, this field is already set to 1. Set this field to the `noErr` result code if you successfully complete the activity. Otherwise, set this field to a result code that you define.
`u.theSMCA->u.slotCID`	`CreationID`	Creation ID of the new Mail Service or Combined record that represents the newly created slot.

DESCRIPTION

The IPM Manager sends the `kMailEPPCCreateSlot` event when a setup template calls the `MailCreateMailSlot` function. Receipt of a `kMailEPPCCreateSlot` event informs a personal MSAM that two actions have already taken place:

1. A new Mail Service or Combined record representing the new slot has been added to the Setup catalog.
2. The configuration information for the new slot has been added to the new record.

Upon receipt of a `kMailEPPCCreateSlot` event, the personal MSAM should call the `AcceptHighLevelEvent` function to get additional information associated with this event and get the creation ID of the new slot's record from the `u.theSMCA->u.slotCID` field of the `MailEPPCMsg` structure. Then the MSAM should read the new slot's record and validate the information it contains. If the information passes the validation checks, the personal MSAM should generate a unique 2-byte slot ID that distinguishes the new slot and add it to the slot's record in the Setup catalog. The MSAM should store the slot ID in an attribute whose type is referenced by the attribute type index `kSlotIDAttrTypeNum`. Valid values for a slot ID range from 1 to $FFFE.

After adding the new slot ID to the slot's record, the MSAM should return the `noErr` result code in the `MailEPPCMsg.u.theSMCA->result` field.

If the information in the new Mail Service or Combined record is invalid, if the MSAM fails to add the new slot ID to the record, or if some other error occurs, the MSAM should return an error code in the `result` field. This error code is available to the MSAM's setup template when the template's call to the `MailCreateMailSlot` function completes. The MSAM and its setup template define the values that the MSAM may return in the `result` field.

While it is running, the MSAM must be prepared to receive and process a `kMailEPPCCreateSlot` event at any time.

RESULT CODES

`noErr`	0	No error

SEE ALSO

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

The `SMCA` structure is described on page 2-114.

The `MailCreateMailSlot` function is described on page 2-213.

For information on setup templates, see the chapter "Service Access Module Setup" in this book.

kMailEPPCModifySlot

The `kMailEPPCModifySlot` event informs a personal MSAM that the user has modified the information associated with a particular slot.

EVENT RECORD

Field name	Data type	Description
`where`	`long`	The constant `kMailEPPCModifySlot`.
`modifiers`	`short`	The slot ID of the slot that has been modified.

MailEPPCMsg STRUCTURE

Field name	Data type	Description
`u.theSMCA->result`	`OSErr`	The result of performing the activity requested by the `kMailEPPCModifySlot` event. When the personal MSAM receives the `kMailEPPCModifySlot` event, this field is already set to 1. Set this field to the `noErr` result code if you successfully complete the activity. Otherwise, set this field to a result code that you define.
`u.theSMCA->u.slotCID`	`CreationID`	Creation ID of the new record that represents the slot that has been modified.

DESCRIPTION

When the information for one of the personal MSAM's slots changes, the MSAM gets a `kMailEPPCModifySlot` event. The IPM Manager sends the `kMailEPPCModifySlot` event when a setup template calls the `MailModifyMailSlot` function. When the IPM Manager sends the event, the MSAM's setup template has already created a new record containing the updated information for the slot and added the record to the Setup catalog. Upon receipt of this event, the personal MSAM should call the `AcceptHighLevelEvent` function to get additional information associated with this event. The MSAM should update any internal data it maintains for the slot and store the creation ID of the slot's new record so that it can read the record if it needs to. For instance, if the MSAM got a second `kMailEPPCModifySlot` event for the same slot, it would want to compare the new and old records to determine which information changed.

The `kMailEPPCModifySlot` event does not invalidate the slot's existing queue references.

After updating its internal data about the modified slot, the MSAM should return the `noErr` result code in the `u.theSMCA->result` field of the `MailEPPCMsg` structure. If it fails to do this for some reason, the MSAM should return an error code in this field. This error code is available to the MSAM's setup template when the template's call to the `MailModifyMailSlot` function completes. The MSAM and its setup template define the values that the MSAM may return in the `MailEPPCMsg`.`u.theSMCA->result` field.

While it is running, the MSAM must be prepared to receive and process a `kMailEPPCModifySlot` event at any time.

RESULT CODES

`noErr`	0	No error

SEE ALSO

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

The `SMCA` structure is described on page 2-114.

The `MailModifyMailSlot` function is described on page 2-215.

For information on setup templates, see the chapter "Service Access Module Setup" in this book.

kMailEPPCDeleteSlot

The `kMailEPPCDeleteSlot` event advises the personal MSAM that a slot will be deleted.

EVENT RECORD

Field name	**Data type**	**Description**
`where`	`long`	The constant `kMailEPPCDeleteSlot`.
`modifiers`	`short`	The slot ID of the slot to be deleted.

MailEPPCMsg STRUCTURE

Field name	**Data type**	**Description**
`u.theSMCA->result`	`OSErr`	The result of performing the activity requested by the `kMailEPPCDeleteSlot` event. When the personal MSAM receives the `kMailEPPCDeleteSlot` event, this field is already set to 1. Set this field to the `noErr` result code if you successfully complete the activity. Otherwise, set this field to a result code that you define.

DESCRIPTION

The IPM Manager sends the `kMailEPPCDeleteSlot` event when a user deletes a slot. Before a slot is actually deleted, the personal MSAM gets a `kMailEPPCDeleteSlot` event. The personal MSAM should call the `AcceptHighLevelEvent` function to get access to the `MailEPPCMsg` structure. It should do what is necessary to handle this event internally, such as discarding data that relates to that slot.

After taking whatever action is appropriate regarding the slot to be deleted, the MSAM should return the `noErr` result code in the `u.theSMCA->result` field of the `MailEPPCMsg`. If it fails to do this for some reason, the MSAM should return an MSAM-defined error result in this field.

If the MSAM returns a `noErr` result code, AOCE software deletes the slot's record in the Setup catalog. If the MSAM returns an error, the slot's record in the Setup catalog is not deleted.

While it is running, the MSAM must be prepared to receive and process a `kMailEPPCDeleteSlot` event at any time.

RESULT CODES

`noErr`	0	No error

SEE ALSO

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

The `SMCA` structure is described on page 2-114.

kMailEPPCMailboxOpened

The `kMailEPPCMailboxOpened` event tells a personal MSAM that a user has opened his or her AOCE desktop mailbox.

EVENT RECORD

Field name	Data type	Description
`where`	`long`	The constant `kMailEPPCMailboxOpened`.
`modifiers`	`short`	Unused; contains 0.

DESCRIPTION

This event notifies the personal MSAM that the user has opened his or her AOCE mailbox. A personal MSAM receiving this event should connect to its external messaging system, check for letters, and update the incoming queue for each of its mail slots.

This event is advisory only and requires no response from the personal MSAM.

kMailEPPCMailboxClosed

The `kMailEPPCMailboxClosed` event tells a personal MSAM that a user has closed his or her mailbox.

EVENT RECORD

Field name	Data type	Description
`where`	`long`	The constant `kMailEPPCMailboxClosed`.
`modifiers`	`short`	Unused; contains 0.

DESCRIPTION

This event notifies the MSAM that the user has closed his or her AOCE mailbox. A personal MSAM receiving this event should disconnect from its external messaging system.

This event is advisory only and requires no response from the personal MSAM.

kMailEPPCShutDown

The `kMailEPPCShutDown` event instructs a personal MSAM to quit immediately.

EVENT RECORD

Field name	Data type	Description
`where`	`long`	The constant `kMailEPPCShutDown`.
`modifiers`	`short`	Unused; contains 0.

DESCRIPTION

This event corresponds directly to the standard Apple event `kAEQuitApplication`. An MSAM should treat it in the same way as it does the `kAEQuitApplication` event. You get this event after the user chooses the Shut Down or Restart command from the Finder’s Special menu.

While it is running, an MSAM must be prepared to receive and process a `kMailEPPCShutDown` event at any time.

kMailEPPCContinue

The `kMailEPPCContinue` event instructs a personal MSAM to resume operation after previously suspending either itself or one of its slots.

EVENT RECORD

Field name	Data type	Description
`where`	`long`	The constant `kMailEPPCContinue`.
`modifiers`	`short`	Contains either the slot ID of a slot to be reactivated or 0. If this field is set to 0, the event applies to the personal MSAM itself.

DESCRIPTION

A personal MSAM may suspend itself or one of its slots if it runs into a problem that requires user intervention to correct. The MSAM should call the `PMSAMLogError` function to report such errors and then suspend itself or the particular slot, whichever is appropriate. While it is in a suspended state, the personal MSAM should continue to call the `WaitNextEvent` function. When the user has taken the appropriate corrective action, the personal MSAM gets the `kMailEPPCContinue` event advising that it should resume operations.

If the problem is with the personal MSAM itself, the MSAM can quit instead of suspending itself. In that case, the IPM Manager launches the MSAM when the user has taken the corrective action and then sends the MSAM the `kMailEPPCContinue` event.

kMailEPPCSchedule

The `kMailEPPCSchedule` event informs a personal MSAM that it is time to log on to its external messaging system and transfer mail on behalf of a specific slot.

EVENT RECORD

Field name	Data type	Description
`where`	`long`	The constant `kMailEPPCSchedule`.
`modifiers`	`short`	The slot ID of the slot whose scheduled time or interval has occurred.

DESCRIPTION

For each account or address that a user has on an external messaging system, the user can provide information on how often or at what time the personal MSAM should log on and transfer mail. The IPM Manager sends a personal MSAM a `kMailEPPCSchedule` event when the schedule information for one of the MSAM's slots indicates that it is time for the MSAM to connect to its external messaging system and transfer mail for that slot. If a personal MSAM is not running at a time when it should log on, the IPM Manager first launches it and then sends it a `kMailEPPCSchedule` event.

SEE ALSO

The frequency information is stored in a `MailStandardSlotInfoAttribute` structure, described on page 2-121.

A setup template obtains scheduling information from the user. See the chapter "Service Access Module Setup" in this book for more information.

kMailEPPCInQUpdate

The `kMailEPPCInQUpdate` event notifies a personal MSAM that a letter in an incoming queue has been updated.

EVENT RECORD

Field name	Data type	Description
`where`	`long`	The constant `kMailEPPCInQUpdate`.
`modifiers`	`short`	The slot ID of the slot whose incoming queue contains the letter to which the event applies.

MailEPPCMsg STRUCTURE

Field name	Data type	Description
`u.sequenceNumber`	`long`	The sequence number of the letter that has either had a change to its attribute values or that has been deleted.

DESCRIPTION

The `kMailEPPCInQUpdate` event informs a personal MSAM that the letter flags attribute for a particular letter has changed, or that the user has deleted the letter. The `modifiers` field of the event record contains the slot ID of the slot to which the letter belongs.

Upon receipt of this event, the personal MSAM should first call the `AcceptHighLevelEvent` function to get additional information associated with this event. The sequence number of the affected letter is specified in the `u.sequenceNumber` field of the `MailEPPCMsg` structure.

If the MSAM chooses to act on the event immediately, it should call the `PMSAMGetMsgSummary` function to read the message summary associated with the letter. If the letter has been deleted by the user, the `msgDeleted` field in the `MSAMMsgSummary` structure is set to `true`. An MSAM operating in online mode should delete the letter on its external messaging system. All MSAMs should delete the message summary for that letter.

If the letter flags attribute has changed, the `msgUpdated` field in the `MSAMMsgSummary` structure is set to `true`. An MSAM operating in online mode should update information about the letter on the external messaging system to maintain consistency with the changed local information about the letter. All MSAMs should set the `msgUpdated` field to `false`.

Alternatively, the personal MSAM can wait until the next time it enumerates the incoming queue that contains the affected letter. At that time, the MSAM can check for letters that have been deleted or whose letter flags attribute has been updated. Then it should take the appropriate action already described here.

SEE ALSO

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

The `SMCA` structure is described on page 2-114.

A personal MSAM deletes letters and message summaries from an incoming queue by calling the `MSAMDelete` function, described on page 2-202.

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

The `MSAMEnumerate` function is described on page 2-138.

Message summaries are described in the section "MSAM Modes of Operation" beginning on page 2-12.

The `MSAMMsgSummary` structure is described on page 2-124.

kMailEPPCMsgOpened

The `kMailEPPCMsgOpened` event tells a personal MSAM that the user wants to open a letter that does not currently exist in the incoming queue. The personal MSAM should place the letter into the incoming queue immediately.

EVENT RECORD

Field name	Data type	Description
`where`	`long`	The constant `kMailEPPCMsgOpened`.
`modifiers`	`short`	The slot ID of the slot whose incoming queue should contain the letter.

MailEPPCMsg STRUCTURE

Field name	Data type	Description
`u.theSMCA->result`	`OSErr`	When the personal MSAM receives the `kMailEPPCMsgOpened` event, this field is already set to 1. Set this field to the `noErr` result code to acknowledge receiving the event. If you already know that it is not possible to retrieve the letter that the user wants to open, set this field to a result code that you define.
`u.theSMCA->userBytes`	`long`	The sequence number of the letter that the user wants to open.
`u.theSMCA->u.msgHint`	`long`	A reference value associated with the letter. You supply this value to the `PMSAMSetStatus` function if you need to report an error.

DESCRIPTION

When a user double-clicks a letter to open it, the IPM Manager checks the associated message summary in the incoming queue to see if the letter itself is also in the queue. If only the message summary is in the incoming queue, the IPM Manager sends a `kMailEPPCMsgOpened` event to the personal MSAM. This event notifies the MSAM that a user wants to open a letter not currently in the incoming queue. Upon receipt of this event, the personal MSAM should call the `AcceptHighLevelEvent` function to get additional information associated with this event. You should acknowledge the event by setting the `u.theSMCA->result` field of the `MailEPPCMsg` structure to the `noErr` result code or, if you are aware of a condition that makes it impossible for you to successfully retrieve the letter, set the field to a result code that you define. If you set the field to `noErr`, you should retrieve the letter from your external messaging system, translate it, and write it to the incoming queue.

If you have a problem retrieving the letter, you should report the problem by calling the `PMSAMSetStatus` function. Set the `seqNum` and `msgHint` fields of the `PMSAMSetStatus` function parameter block to the values of the `u.theSMCA->userBytes` and `u.theSMCA->u.msgHint` fields of the `MailEPPCMsg` structure, respectively. Then set the `status` field of the parameter block to `kPMSAMStatusError` and call the function.

RESULT CODES

`noErr`	0	No error

SEE ALSO

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

The `SMCA` structure is described on page 2-114.

kMailEPPCDeleteOutQMsg

The `kMailEPPCDeleteOutQMsg` event instructs a personal MSAM to delete a message in its outgoing queue.

EVENT RECORD

Field name	**Data type**	**Description**
`where`	`long`	The constant `kMailEPPCDeleteOutQMsg`.
`modifiers`	`short`	The slot ID of the slot whose outgoing queue holds the letter to be deleted.

MailEPPCMsg STRUCTURE

Field name	**Data type**	**Description**
`u.sequenceNumber`	`long`	The sequence number of the letter that the user has deleted.

DESCRIPTION

This event tells a personal MSAM to delete, rather than send, a letter in its outgoing queue. The IPM Manager sends this event in response to a user action. Upon receipt of this event, the personal MSAM should call the `AcceptHighLevelEvent` function to get the sequence number of the letter.

SEE ALSO

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

The `SMCA` structure is described on page 2-114.

kMailEPPCWakeup

The `kMailEPPCWakeup` event notifies a personal MSAM that a process called the `MailWakeupPMSAM` function.

EVENT RECORD

Field name	Data type	Description
`where`	`long`	The constant `kMailEPPCWakeup`.
`modifiers`	`short`	Unused; contains 0.

DESCRIPTION

When a process calls the `MailWakeupPMSAM` function, the IPM Manager sends a `kMailEPPCWakeup` event to the personal MSAM specified by the application. Typically, a process calls the `MailWakeupPMSAM` function in response to an external event that cannot be predicted. For example, a fax modem driver might call the `MailWakeupPMSAM` function when it has received an incoming call so that the MSAM can put the fax into the incoming queue.

If the MSAM is not running at the time the `MailWakeupPMSAM` function is called, the IPM Manager launches it.

kMailEPPCLocationChanged

The `kMailEPPCLocationChanged` event notifies a personal MSAM that the current system location has changed or that a user has changed the location flags for the specified slot.

EVENT RECORD

Field name	Data type	Description
`where`	`long`	The constant `kMailEPPCLocationChanged`.
`modifiers`	`short`	The slot ID of the slot to which the event applies.

MailEPPCMsg STRUCTURE

Field name	Data type	Description
`u.locationInfo->location`	`OCESetupLocation`	A value that identifies the current system location. It may contain any integer value between 0–8.
`u.locationInfo->active`	`MailLocationFlags`	A bit array that defines whether the slot is active at a given location.

DESCRIPTION

The IPM Manager sends a `kMailEPPCLocationChanged` high-level event when either of two events occurs:

1. The current system location changes. In this case, the IPM Manager sends one `kMailEPPCLocationChanged` high-level event for each slot belonging to an MSAM.
2. A user activates or deactivates a mail slot in a given location. In this case, the IPM Manager updates the location flags in the `MailStandardSlotInfoAttribute` structure for that slot and sends a `kMailEPPCLocationChanged` high-level event to the MSAM.

The event tells the MSAM the slot to which the event applies, the current system location, and the location flags for the slot. Upon receipt of a `kMailEPPCLocationChanged` high-level event, an MSAM should examine the location flags. If the location flags show that the slot is inactive at the current location and the slot was previously active, the MSAM should immediately stop performing any activity on behalf of the slot, such as downloading letters or attempting to send letters. If the location flags show that the slot is active at the current location and the slot was previously inactive, the MSAM should begin acting on behalf of the slot.

SEE ALSO

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

The `MailLocationFlags` data type is described on page 2-115.

The `OCESetupLocation` data type is described on page 2-115.

kMailEPPCSendImmediate

The `kMailEPPCSendImmediate` event notifies a personal MSAM to send a letter in an outgoing queue as soon as possible.

EVENT RECORD

Field name	Data type	Description
`where`	`long`	The constant `kMailEPPCSendImmediate`.
`modifiers`	`short`	The slot ID of the slot in whose outgoing queue the letter resides.

MailEPPCMsg STRUCTURE

Field name	Data type	Description
`u.theSMCA->result`	`OSErr`	The result of performing the activity requested by the `kMailEPPCSendImmediate` event. When the personal MSAM receives the `kMailEPPCSendImmediate` event, this field is already set to 1. Set this field to the `noErr` result code if you successfully complete the activity. Otherwise, set this field to an appropriate result code.
`u.theSMCA->userBytes`	`long`	The sequence number of the letter that the MSAM should attempt to send immediately.

DESCRIPTION

The IPM Manager sends a `kMailEPPCSendImmediate` event in response to a user's request to send a letter immediately. When a personal MSAM receives the event, it should attempt immediate delivery of the letter to the external messaging system. The letter is specified in the `MailEPPCMsg.u.theSMCA->userBytes` field of the external messaging system.

After sending the letter, the MSAM should return the `noErr` result code in the `u.theSMCA->result` field of the `MailEPPCMsg` structure. If it is unable to send the letter, the MSAM should return an error result code in this field. Typically, the result codes it returns are `kMailSlotSuspended` and `kMailTooManyErr.`

RESULT CODES

`noErr`	0	No error
`kMailTooManyErr`	–15055	MSAM too busy to process event
`kMailSlotSuspended`	–15058	Slot is suspended

SEE ALSO

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

The `SMCA` structure is described on page 2-114.

kMailEPPCMsgPending

The `kMailEPPCMsgPending` event informs a personal or server MSAM that there is a message in an outgoing queue.

EVENT RECORD

Field name	**Data type**	**Description**
`where`	`long`	The constant `kMailEPPCMsgPending`.
`modifiers`	`short`	For personal MSAMs, this field contains the slot ID of the slot in whose outgoing queue the letter is located. For server MSAMs, this field contains 0.

DESCRIPTION

Upon receiving a `kMailEPPCMsgPending` event, a personal MSAM should retrieve the letter from the outgoing queue of the slot identified in the `modifiers` field. A server MSAM should retrieve the message from its single outgoing queue. Both personal and server MSAMs should then translate the letter or non-letter message and transmit it to the external messaging system.

When an MSAM is launched, it should check its outgoing queue or queues for messages awaiting transmittal. The `kMailEPPCMsgPending` event makes constant monitoring of the outgoing queue or queues for pending messages unnecessary. However, like all high-level events, a `kMailEPPCMsgPending` event may be lost. Therefore, an MSAM should periodically check its outgoing queue or queues rather than relying exclusively on the `kMailEPPCMsgPending` event to inform it of pending messages.

kMailEPPCAdmin

The `kMailEPPCAdmin` event notifies a server MSAM that its configuration has changed.

EVENT RECORD

Field name	**Data type**	**Description**
`where`	`long`	The constant `kMailEPPCAdmin`.
`modifiers`	`short`	Unused; contains 0.

***MailEPPCMsg* STRUCTURE**

Field name	Data type	Description
`u.theSMCA->result`	`OSErr`	When a server MSAM receives the `kMailEPPCAdmin` event, this field is already set to 1. Set this field to the `noErr` result code to acknowledge receiving the `kMailEPPCAdmin` event.
`u.theSMCA->userBytes`	`long`	Pointer to a `SMSAMAdminEPPCRequest` structure.

DESCRIPTION

The `kMailEPPCAdmin` high-level event notifies a server MSAM that its configuration has changed. Upon receiving the `kMailEPPCAdmin` event, a server MSAM should call the `AcceptHighLevelEvent` function to get additional information associated with this event. The `MailEPPCMsg.u.theSMCA->result` field is initially set to 1. The MSAM should set the `MailEPPCMsg.u.theSMCA->result` field to `noErr` to acknowledge receipt of the `kMailEPPCAdmin` event.

The `SMSAMAdminEPPCRequest` structure pointed to by the `MailEPPCMsg.u.theSMCA->userBytes` field contains an `adminCode` field. The value in the `adminCode` field indicates the format of the remaining data in the `SMSAMAdminEPPCRequest` structure. In release 1 of the PowerShare software, the `adminCode` field should always be set to `kSMSAMNotifyFwdrSetupChange`, indicating that the remaining data is an `SMSAMSetupChange` structure. If you receive a `kMailEPPCAdmin` event whose code value is not `kSMSAMNotifyFwdrSetupChange`, you should acknowledge it (set the `MailEPPCMsg.u.theSMCA->result` field to `noErr`) and then ignore the event.

In release 1 of the PowerShare software, the `kSMSAMNotifyFwdrSetupChange` subtype of the `kMailEPPCAdmin` event always indicates that the record location information of the server MSAM's foreign dNodes has changed. The MSAM can verify this by examining the `whatChanged` field in the `SMSAMSetupChange` structure. The `kSMSAMFwdrForeignRLIsChangedBit` bit should be set. The server MSAM should read its Forwarder record to obtain the new record location information of its foreign dNodes.

SPECIAL CONSIDERATIONS

Server MSAMs should act only on `kMailEPPCAdmin` events that are generated on the local computer. When you call the `AcceptHighLevelEvent` function, it returns a `TargetID` structure. Within that structure is a `LocationNameRec` structure. If the `locationKindSelector` field of the `LocationNameRec` structure is set to `ppcNoLocation`, you know that the event's sender resides on the local computer.

RESULT CODES

`noErr`	0	No error

SEE ALSO

See the section "AOCE Addresses" beginning on page 2-23 for a description of foreign dNodes.

The section "Initializing a Server MSAM" beginning on page 2-40 describes what types of information are found in a server MSAM's Forwarder record and how it gets there.

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

The `SMCA` structure is described on page 2-114.

The `SMSAMAdminEPPCRequest` structure is described on page 2-117.

The `SMSAMSetupChange` structure is described on page 2-117.

Record location information is specified by an `RLI` structure. It is described in the chapter "AOCE Utilities" in *Inside Macintosh: AOCE Application Interfaces*.

The `LocationNameRec` structure is described in *Inside Macintosh: Interapplication Communication*.

The `AcceptHighLevelEvent` function and the `TargetID` structure are described in *Inside Macintosh: Macintosh Toolbox Essentials*.