INSIDE MACINTOSH

AOCE Service Access Modules

Contents

Chapter 3   Catalog Service Access Modules   3-1

Chapter 4   Service Access Module Setup   4-1

vi

Glossary

Index

vii

Figures, Tables, and Listings

x

About This Book

This book, *Inside Macintosh: AOCE Service Access Modules,* describes the mechanisms by which you can add catalog and messaging services to those that are available through PowerTalk system software and PowerShare collaboration servers. The technology underlying the PowerTalk and PowerShare software is called the **Apple Open Collaboration Environment** (AOCE). In this book, the term *AOCE software* refers to the Macintosh Operating System managers, Finder extensions, and other system software that the PowerTalk system software and PowerShare servers use to implement their many features. You use this AOCE software to implement your service access module. The term *PowerTalk system software* refers specifically to the implementation of the AOCE technology for the Macintosh Computer, and the term *PowerShare collaboration servers* refers to AOCE-based servers provided by Apple Computer, Inc., that provide mail, messaging, catalog, security, and time services.

You need to read this book if you want to extend the capabilities of the PowerTalk system software to take advantage of services offered by external catalogs (also known as *directories* or *databases*) and external messaging systems. This book describes the architecture of catalog service access modules (CSAMs) and messaging service access modules (MSAMs) and explains how each type of service access module (SAM) interacts with AOCE software. This book also describes the special AOCE templates that SAMs require to obtain configuration and address information from the user. It provides a technical reference to the system software routines that you use to provide catalog and messaging services.

This book assumes that you are an experienced C and Macintosh programmer and are familiar with the capabilities of AOCE software. Before reading this book, you should read at least these chapters in *Inside Macintosh: AOCE Application Interfaces*:

n "Introduction to the Apple Open Collaboration Environment" describes some of the uses of PowerTalk and PowerShare system software and introduces all of the AOCE managers. It discusses some concepts fundamental to an understanding of the AOCE software and defines many new terms.

n "AOCE Utilities" describes AOCE data structures and utility routines.

n "AOCE Templates" describes AOCE template resources. You need to under-stand standard AOCE templates before you can write the setup template that most SAMs require and the address template that all MSAMs require.

n "Catalog Manager" describes functions you implement in your CSAM to service user requests for information about the catalogs that you support and to manipulate the data in those catalogs.

In addition, portions of the chapters "Interprogram Messaging Manager" and "Authentication Manager" in *Inside Macintosh: AOCE Application Interfaces* provide information useful in developing a SAM. This book contains cross-references to those chapters where appropriate.

In this book, the chapter "Introduction to Service Access Modules" provides a brief overview of the different types of SAMs and their setup and address templates.

The chapter "Catalog Service Access Modules" describes the architecture and the components of a CSAM. This chapter does not stand alone. To implement a CSAM, you need a sound understanding of Catalog Manager functions and AOCE data types, described in the chapters "Catalog Manager" and "AOCE Utilities" in *Inside Macintosh: AOCE Application Interfaces*.

The chapter "Messaging Service Access Modules" describes how you can interface an external mail or messaging system with the PowerTalk system software by writing an MSAM. It explains the structure of personal and server MSAMs and the differences between them, and describes how you can accomplish the most common tasks of an MSAM.

The chapter "Service Access Module Setup" describes the setup template, required for CSAMs and personal MSAMs, and the address template, required for all MSAMs. It also describes the records in the PowerTalk Setup catalog that the templates manipulate.

For your convenience, this book and *Inside Macintosh: AOCE Application Interfaces* include the same glossary of AOCE terminology. Thus, some glosssary entries refer to topics that are not introduced in this book.

Format of a Chapter

The chapters in this book typically contain an overview of the features provided by the subject of the chapter, sections that describe how to use the most common routines along with code samples, a reference section, and a summary section.

The content of the reference section differs somewhat from chapter to chapter. For example, whereas the reference section of the chapter "Messaging Service Access Modules" describes the data structures and functions used by the MSAM API, the reference section of the chapter "Service Access Module Setup" describes the records in the PowerTalk Setup catalog and the resources that constitute the setup template. In each case, the reference section provides a complete reference to the portion of AOCE system software described by that chapter.

Function descriptions follow a standard format, which gives the function declaration and a description of every parameter of the function. Some function descriptions also give additional descriptive information, such

as special considerations and cross-references to other sections, chapters, and books.

The summary section typically provides the API's C interface, as well as the Pascal interface, for the constants, data structures, functions, and result codes associated with the API. It also includes some assembly-language interface information.

Some chapters include additional main sections that provide more detailed discussions of certain topics. For example, the chapter "Messaging Service Access Modules" contains the section "AOCE Addresses," which describes the format of addresses used by PowerTalk software.

Conventions Used in This Book

Inside Macintosh uses various conventions to present information. Words that require special treatment appear in specific fonts or font styles. Certain information, such as parameter blocks, use special formats so that you can scan them quickly.

Special Fonts

All code listings, reserved words, and the names of actual data structures, constants, fields, parameters, and functions are shown in Courier (`this is Courier`).

Words that appear in **boldface** are key terms or concepts defined in the glossary.

Types of Notes

Three types of notes are used in this book:

Note
A note like this contains general information that is supplemental to the main text. (An example appears on page 3-5.) u

Special topic note
A note like this contains information about a specific topic that is supplemental to the main text. (An example appears on page 2-6.) u

IMPORTANT
A note like this contains information that is essential for an understanding of the main text and that might cause you problems if ignored. (An example appears on page 2-67.) s

Warnings like this indicate potentially severe problems that you should
be aware of as you design your application. Failure to heed these
warnings could result in system crashes or loss of data. (An example
appears on page 2-197.)   s

Parameter Block Information

Inside Macintosh presents information about the fields of a parameter block in
this format:

Parameter block

inAndOut	Boolean	Input/output parameter.
output1	OSErr	Output parameter.
input1	long	Input parameter.

The arrow in the far left column indicates whether the field is an input
parameter, output parameter, or both. You must supply values for all input
parameters and input/output parameters. The function returns values in
output parameters and input/output parameters.

The second column shows the field name as defined in the MPW C interface
files; the third column indicates the C data type of that field. The fourth
column provides a brief description of the use of the field. For a complete
description of each field, see the discussion that follows the parameter block
or the description of the parameter block in the reference section of the
chapter.

Development Environment

The system software routines described in this book are available using C or
Pascal interfaces. You can call most of these routines in assembly language,
but no assembly-language interface files are provided. How you access these
routines depends on the development environment you are using. This book
shows system software functions in their C interface using the Macintosh
Programmer's Workshop (MPW).

All code listings in this book are shown in C, or, for resources, in Rez input
format. They show methods of using various routines and illustrate
techniques for accomplishing particular tasks. Not all code listings have been
compiled or tested. These code listings are for illustrative purposes only;
Apple Computer, Inc., does not intend for you to use these code samples in
your application.

For More Information

APDA is Apple's worldwide source of information about more than 300 development tools, technical resources, and training products. APDA is a valuable resource for anyone interested in developing applications on Apple platforms. Customers receive the quarterly *APDA Tools Catalog* featuring all current versions of Apple development tools and the most popular third-party development tools. Ordering is easy. There are no membership fees, and application forms are not required for most products. APDA offers convenient payment and shipping options, including site licensing.

To order products or to request a complimentary copy of the *APDA Tools Catalog,* contact

APDA
Apple Computer, Inc.
P.O. Box 319
Buffalo, NY 14207-0319

Telephone	800-282-2732 (United States)
	800-637-0029 (Canada)
	716-871-6555 (International)
Fax	716-871-6511
AppleLink	APDA
America Online	APDAorder
CompuServe	76666,2405
Internet	APDA@applelink.apple.com

If you provide commercial products and services, call 408-974-4897 for information on the developer support programs available from Apple.

For information on registering application signatures, file types, Apple events, and other technical information, contact

Macintosh Developer Technical Support
Apple Computer, Inc.
20525 Mariani Avenue, M/S 303-2T
Cupertino, CA 95014-6299

Introduction to Service Access Modules

Contents

This book describes service access modules and their setup and address templates. A service access module (SAM) extends a user's PowerTalk system to provide access to non-AOCE mail and messaging services and catalog services.

The AOCE software comes with a set of application programming interfaces (APIs) that allow you to extend the features provided by PowerTalk, to build collaborative applications, and to create service access modules that integrate external services into a user's PowerTalk environment. This book describes the APIs you can use to create SAMs and tells you how to write and set up a SAM. The AOCE APIs used by application programs are described in the book *Inside Macintosh: AOCE Application Interfaces*.

You should read this chapter if you are interested in developing a service access module for PowerTalk system software. PowerTalk system software is the implementation of the Apple Open Collaboration Environment (AOCE) technology by Apple Computer, Inc. You will find detailed information about service access modules in the remaining chapters of this book.

This chapter gives a brief overview of service access modules and describes how they fit into a PowerTalk system. Then, it briefly describes mail and messaging service access modules, catalog service access modules, and the AOCE setup and address templates needed for the configuration of service access modules.

Overview

Service access modules and their setup and address templates provide

n   a user interface for non-AOCE mail and messaging services that is consistent with that provided by the PowerTalk system software and PowerShare servers

n   a user interface for browsing, searching, and editing information contained in non-AOCE databases and address directories that is consistent with that provided by PowerTalk and PowerShare catalogs

n   a consistent programming interface for collaborative application developers, facilitating the development of cross-platform collaborative applications

Through the mechanism of the service access module, the PowerTalk system software architecture simplifies a Macintosh computer user's interaction with existing mail and messaging services and with catalog services. A **service access module** (SAM) is a software component that provides the PowerTalk user with access to external mail and messaging services or catalog services. **External services** are those that are not provided automatically with PowerTalk system software and PowerShare servers. A SAM provides its services to the user through the Catalogs and Mailbox Extensions to the Finder and through AOCE templates. Therefore, the user interface is consistent across different mail and messaging services and catalog services.

Consider the situation prior to the advent of AOCE technology. A Macintosh user with accounts on a variety of electronic mail services, such as AppleLink, the Internet, CompuServe, cc:Mail, and QuickMail, had to log on to each of these services to send and receive mail. With PowerTalk software installed, by contrast, a user can employ a single

method to access all electronic mail services, sending and receiving all types of electronic mail through a single mailbox on the desktop. You can provide a PowerTalk user with access to an external mail or messaging system by writing a messaging service access module (MSAM).

Typically, a user needs a repository of addresses and the ability to look up those addresses to make use of a mail service. In a PowerTalk system, addresses are stored in catalogs. Although one of the primary uses of a catalog is to store addresses, the content of catalogs is not limited to address information. In fact, you can provide a catalog service access module (CSAM) and associated AOCE templates to allow a PowerTalk user to browse and read any sort of information stored in any sort of external database, address directory, or catalog, regardless of the structure of the underlying information.

AOCE technology allows two types of MSAMs: server-based and personal. Whereas a server-based MSAM requires a system administrator to set it up and maintain it, any user can install a personal MSAM on his or her computer, becoming both the system administrator and a user of the system (in much the same way that System 7 provided personal file sharing). All CSAMs are personal, installed on an individual user's computer. Personal SAMs require minimal setup by the user and need no intervention by a system administrator.

Messaging Service Access Modules

A messaging service access module (MSAM) provides a link or gateway to an external mail or messaging service. A mail service transfers information between people. A messaging service transfers information between processes. An MSAM may provide either mail or messaging services, or both.

An MSAM's basic tasks are to translate addresses and data from AOCE formats to external formats and vice versa, and to transfer messages. Historically, most gateways have been mail gateways, and most mail has consisted of plain text data. However, users can now exchange mail that contains styled text, pictures, sounds, and movies as well. Processes can also exchange data in a variety of formats.

MSAMs come in two basic types—server-based and personal. A server-based MSAM acts much like a traditional store-and-forward mail gateway. It provides a transport-level connection between a PowerShare mail server and one or more external mail or messaging services. The external services may be of different types. For instance, it is possible for a single server-based MSAM to provide a connection to AppleLink and to the Internet. A server-based MSAM must be set up and maintained by a system administrator and typically connects large systems.

A server-based MSAM does not work on behalf of individual users; it does not need individual account or password information. It delivers messages to a system. It is not responsible for delivering the message to the recipient. You implement a server-based MSAM as a foreground application.

Personal MSAMs represent a major innovation in the use of gateways, unique to the Macintosh computer. A personal MSAM is user-centered; it acts as the user's agent. It provides a user with a personal connection to an external mail or messaging service through the Mailbox Extension to the Finder. A user simply drops the MSAM into the System Folder and provides configuration information through the PowerTalk Key Chain. A personal MSAM does not require the services of a network or system administrator.

Usually, one thinks of a personal MSAM as connecting a user to a mail service, for example, the AppleLink service. A personal MSAM can also provide access to private devices connected to the user's Macintosh computer. For instance, you can write a personal MSAM to connect to a fax modem.

There are many implementation decisions you must make when writing a personal MSAM. For instance, once the user has read a message, you must decide whether to delete the message in the user's account on the external mail service, or to keep a copy of the message. The choice has certain implications for the user. Consider an MSAM that automatically deletes mail once it has been read. Suppose a user opts to have mail automatically downloaded at certain times (a feature all personal MSAMs should offer). In this case, when the user is not at his or her Macintosh computer, he or she may not have access to the mail (because, once the MSAM has downloaded the mail, no copy exists on the external mail service). If, however, the MSAM keeps a copy of previously read mail on the external mail system, then the user must periodically empty the mailbox on the external mail system or the mail server's disk will eventually become full. Your MSAM can deal with such choices in any way you see fit, including offering the user both options in a preferences dialog box.

Another example concerns whether or not to store incoming mail on the user's Macintosh computer. Personal MSAMs create message summaries for incoming mail. A message summary contains important information about the message, such as the sender, the subject, the time it was sent, and so forth. As a result, the user can browse incoming mail without the message itself being physically present on the user's computer. An MSAM can then download the message itself only when the user actually wants to open and read the message. Downloading a message on demand is an advantage if disk space is in short supply on the user's Macintosh. On the other hand, it is a disadvantage if the physical connection over which the message is transferred is slow.

You make these and other implementation decisions by considering the characteristics of the mail system to which you provide access and the needs of your users. PowerTalk system software does not dictate these decisions. You implement a personal MSAM as a background application.

The current implementation of AOCE system software does not fully support the transfer of process-to-process messages by a personal MSAM.

For detailed information about writing an MSAM, see the chapter "Messaging Service Access Modules" in this book.

Catalog Service Access Modules

A catalog service access module (CSAM) provides a user with access to one or more catalogs of information and with a consistent way of browsing and searching the information. A CSAM implements the Catalog Manager API for an external catalog or database and translates data between AOCE data formats and those of the external catalogs that the CSAM supports.

AOCE catalog services grew out of the need to provide a way for users to browse and search for the addresses of those they wanted to communicate with. Once an MSAM is available to a user, it is useful only if the user knows one or more addresses reachable through that MSAM. Typically, a user wants to look up addresses in an address directory. For this reason, an MSAM is usually accompanied by a CSAM that gives the user access to a catalog containing addresses available on a given messaging service.

AOCE catalogs can contain any type of information. You can write a CSAM that has no association with an MSAM. The CSAM may provide access to a database containing, for example, a native plant encyclopedia or a reference on human nutrients. A catalog can contain any information that can be stored in AOCE records and attributes and that can be displayed by AOCE templates.

Because not all external catalogs and databases have the same capabilities, a CSAM must provide a set of capability flags to the Catalogs Extension to the Finder. The flags indicate the capabilities of each catalog the CSAM supports. The user interacts directly with the Catalogs Extension (CE) to search a catalog. Therefore, the user is limited to the search capabilities of the CE and cannot use additional search or query capabilities that may exist in the external catalog or database.

A CSAM is not limited to accessing traditional shared databases. You can write a CSAM to access private devices that the user connects to the Macintosh computer. For example, a catalog can reside on a compact disc.

Most users search or browse catalogs in real time. Because the task of retrieving information is performance-sensitive, you implement a CSAM as a driver.

The AOCE software architecture does not prevent the development of server-based CSAMs. However, support for server-based CSAMs is not currently implemented in the AOCE software. If you want to make information available to networked users as a server-based catalog, you can write an application that transfers your external catalog information into a PowerShare catalog.

As is the case with MSAMs, there are numerous implementation decisions that you must make when writing a CSAM. The AOCE system software architecture allows for much flexibility. For example, you can cache information from your external catalog locally or you can retrieve it when the user wants it. You make these decisions based on the characteristics of the catalog services you support and the needs of your users.

For detailed information about writing a CSAM, see the chapter "Catalog Service Access Modules" in this book.

AOCE Setup and Address Templates

A special catalog called the *PowerTalk Setup catalog* stores information about all of the mail and messaging and catalog services available to the user of a given Macintosh. The writer of a personal MSAM or CSAM must provide a setup template, which is a set of AOCE templates that work with the PowerTalk Key Chain to allow the user to set up and configure the mail or catalog service. The user enters such information as the account name and password, automatic connection preferences, and so forth.

If you are writing an MSAM, you also need to provide an address template that allows the user to create addresses for the external messaging system.

For detailed information about writing AOCE setup and address templates, see the chapter "Service Access Module Setup" in this book.

Messaging Service Access Modules

Contents

This chapter describes Apple Open Collaboration Environment (AOCE) messaging service access modules. A messaging service access module is a software component that provides the PowerTalk user with access to external mail and messaging services. You do not need to read this chapter if you are writing a mail or messaging application or adding mail or messaging capabilities to your application.

To write a messaging service access module, you need to be familiar with many components of AOCE software. You should read the chapters "Introduction to the Apple Open Collaboration Environment" and "AOCE Utilities" in *Inside Macintosh: AOCE Application Interfaces* before reading this chapter to get a general overview of AOCE software components and the shared AOCE data types and the utility routines that act on them. This chapter assumes that you are familiar with AOCE catalogs and records and their structures, and that you know how to read and write data to them. The chapters "Standard Catalog Package" and "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* describe the high-level application programming interface (API) and the low-level API to AOCE catalogs, respectively.

To read and write AOCE records, you must obtain an authentication identity. Identities are described in the chapter "Authentication Manager" in *Inside Macintosh: AOCE Application Interfaces.*

Along with your messaging service access module, you need to provide a type of AOCE template called an *address template* to allow the user to enter address information. If you are writing a personal messaging service access module, you also need to provide a setup template that allows the user to configure your access module. The chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces* describes how to write an AOCE template. The chapter "Service Access Module Setup" in this book provides additional specific information about setup and address templates and their interaction with messaging service access modules and the PowerTalk Key Chain.

All messaging service access module developers need to be familiar with high-level events. See the chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for information about high-level events.

This chapter starts with an introduction to messaging service access modules. Subsequent sections describe

- n personal messaging service access modules

- n server messaging service access modules

- n the types of messages that are read and written by messaging service access modules

- n AOCE addresses

- n the AOCE high-level events

- n how to get messages out of an AOCE system

- n how to put messages into an AOCE system

- n the structures and routines in the messaging service access module API

Introduction to Messaging Service Access Modules

A **messaging system** is a combination of hardware and software that provides people and processes with the ability to exchange electronic messages—it provides messaging services. Apple's **AOCE messaging system** consists of PowerTalk system software and PowerShare mail servers that allow Macintosh users and processes accessible over a network or via a modem to exchange electronic messages. Today there are many types of messaging systems, such as Internet, AppleLink, QuickMail, and so forth, with which AOCE users might want to communicate. To facilitate the exchange of messages between an AOCE messaging system and other existing and future messaging systems, the AOCE architecture defines a **messaging service access module (MSAM).** An MSAM links Apple's AOCE messaging system to another messaging system, extending the reach of messaging service clients.

The AOCE architecture defines two kinds of MSAMs. A **personal MSAM** translates messages and transfers them between a user's Macintosh and the user's account on another messaging system. It runs on a user's Macintosh. A **server MSAM** translates and transfers messages between a PowerShare mail server and a non-AOCE messaging system. A server MSAM transfers messages for any number of users located on the AppleTalk network to which it is connected. It runs on a Macintosh with a PowerShare mail server. Thus, the MSAM component of AOCE software architecture is scalable. It can provide service to a single user who uses a non-networked Macintosh computer or to large numbers of users in large internetworks.

Figure 2-1 shows how adding an MSAM to an AOCE system extends the reach of AOCE users. Prior to adding an MSAM, AOCE users cannot exchange electronic messages with others who are accessible only on a non-AOCE messaging system. Once an MSAM that connects to the non-AOCE messaging system is added, the AOCE users can exchange messages with people accessible on the non-AOCE messaging system.

The basic services provided by both personal and server MSAMs include

n   transferring messages between an AOCE messaging system and another messaging system

n   translating the content of messages between AOCE-defined formats and other formats

n   translating message addresses between AOCE-defined formats and other formats

n   reporting the results of attempts to deliver messages

Personal and server MSAMs are described in more detail in the following sections.

A note on terminology

Throughout this chapter, the term *message* is used as an inclusive term to refer to all types of messages. When information applies only to letters (a specific type of message), the term *letter* is used. When information applies only to messages that are not letters, the term **non-letter message** is used. Letters and messages are defined in the section "Types of Messages" beginning on page 2-16.

Figure 2-1   Adding an MSAM
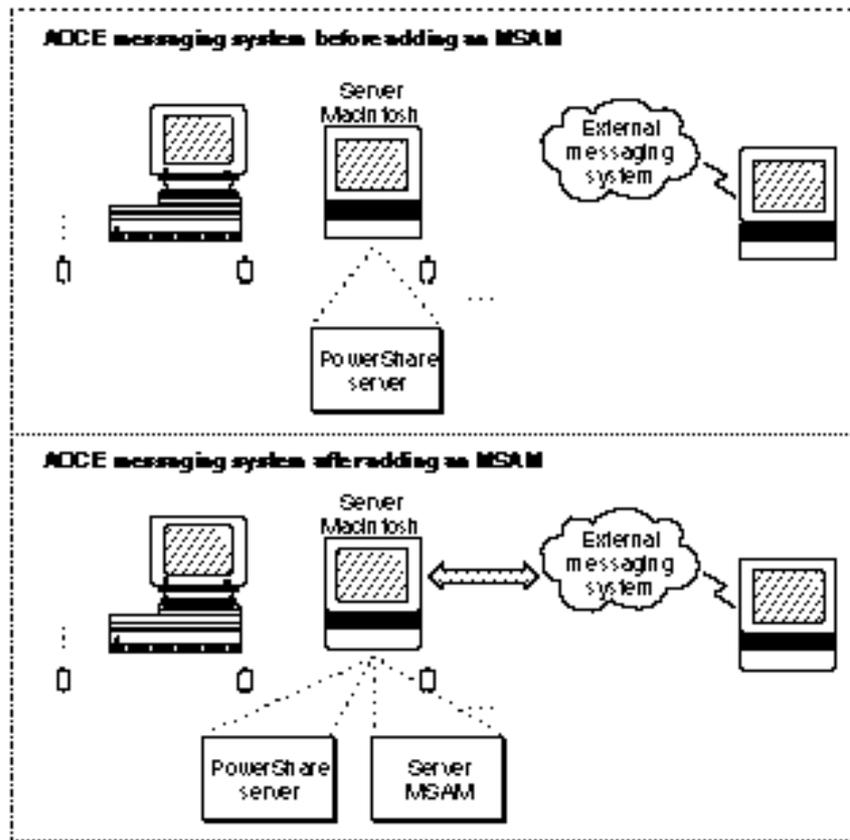
Messaging systems that are not provided automatically with PowerTalk system software and PowerShare servers are collectively referred to as **external messaging systems.** An external messaging system may handle only letters or non-letter messages or both.

The term **mail** refers to letters. Messaging systems that handle only letters are sometimes referred to as *mail systems.*

As a convention, this chapter refers to messages coming into an AOCE system from an external messaging system as **incoming messages** and to those that are leaving an AOCE system to go into an external messaging system as **outgoing messages.**

Throughout the chapter, the text distinguishes between personal and server MSAMs where appropriate. The term *MSAM* is used when the text applies to both personal and server MSAMs, unless it is clear from the context that only a personal or server MSAM is meant.   u

An MSAM is a low-level component in the AOCE software hierarchy. It does not directly provide services to a user or process; rather, it provides services indirectly through either the Standard Mail Package or the Interprogram Messaging (IPM) Manager. Thus, a client has a standard interface to all messaging systems, including those that are accessible via

MSAMs as well as Apple's PowerTalk and PowerShare services, regardless of underlying differences in how messages are accessed and formatted. Figure 2-2 shows the relationship of two clients, the Standard Mail Package, the IPM Manager, an MSAM, and an external messaging system.

Figure 2-2   An MSAM's relationship to AOCE software
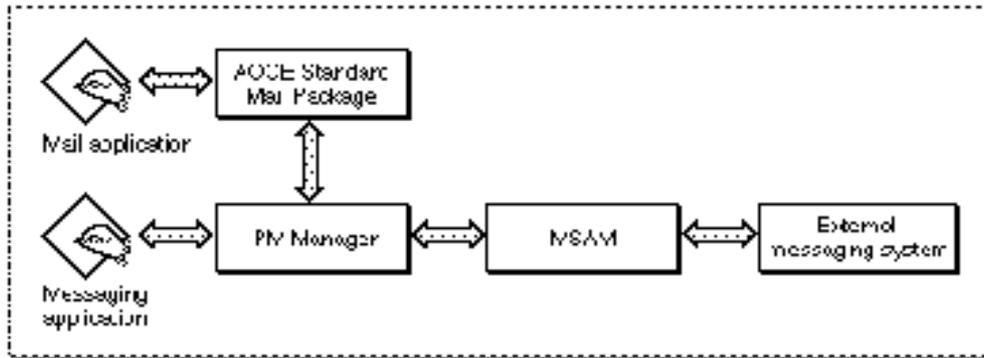
MSAMs interact with the IPM Manager. Either the MSAM or the IPM Manager can initiate communication with the other. Figure 2-3 illustrates the way the IPM Manager and an MSAM initiate communications with each other. An MSAM initiates communication with the IPM Manager by calling one of the functions provided in the MSAM API. These functions are described in detail in the section "MSAM Functions" beginning on page 2-130.

The IPM Manager initiates communication with an MSAM by sending it a high-level event. The events that the IPM Manager may send to an MSAM, which typically instruct the MSAM to take some action or advise it of a status change, are described in the section "High-Level Events" beginning on page 2-220.

Figure 2-3   Communication between the IPM Manager and an MSAM
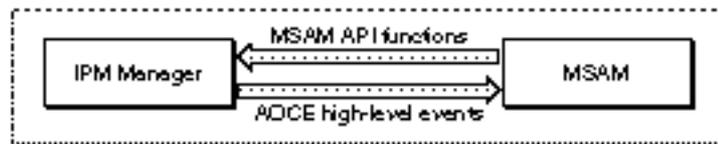
Personal MSAMs

A personal MSAM allows a user or a mail or messaging application to transfer messages between the user's Macintosh and users or applications on one or more external messaging systems. A personal MSAM connects to an external messaging system and transfers messages between the user's Macintosh and the external messaging system. The user or process must have an account on the external messaging system to which the personal MSAM provides access. The user's Macintosh does not need to be connected to an AppleTalk network.

A personal MSAM is a background-only application; that is, it has no user interface.

Every personal MSAM must be accompanied by AOCE templates that allow the user to configure the MSAM and to enter address information. These templates, called the *setup template* and *address template*, are described in the chapter "Service Access Module Setup" in this book. Information that applies to all AOCE templates is provided in the chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces.*

A file containing a personal MSAM must have a file type of either `'msam'` or `'csam'`. If you provide both a personal MSAM and a catalog service access module (CSAM) in the same file, use the file type `'csam'` (for "combined service access module"). If you provide a personal MSAM only, use the file type `'msam'`. You must include your setup and address templates in the same file as your personal MSAM.

Although personal MSAMs and server MSAMs both connect to external messaging systems and translate and transfer messages, there are a number of differences between them. See Table 2-1 on page 2-11 for a list of these differences.

A **slot**, as the term is used in the MSAM API and in this chapter, refers to a collection of information about one account on an external messaging system. The information includes whatever is necessary to allow an MSAM to access the account and retrieve and send messages. Slot information determines what external messaging system the MSAM connects to. The term **mail slot** refers to a slot that allows the transfer of letters. The term **messaging slot** refers to a slot that allows the transfer of non-letter messages.

Slot information is stored in the form of AOCE record attributes in records in the PowerTalk Setup catalog. The record types in which the information is stored differ depending on whether you provide a combined MSAM/CSAM or a stand-alone MSAM. If you provide a combined MSAM/CSAM, slot information and its associated catalog information is stored in a single Combined record. If you provide a stand-alone MSAM, slot information is stored in a Mail Service record (sometimes called a *slot record*) and associated catalog information is stored in a Catalog record. The setup template that you provide with your MSAM writes slot information to some of these records; the PowerTalk Key Chain writes to others. The chapter "Service Access Module Setup" in this book describes the required attributes of the Combined, Mail Service, and Catalog records, and it explains who is responsible for writing those attributes to the different types of records in the Setup catalog.

In addition to the required record attributes, slot information includes whatever is necessary to allow the MSAM to service the slot—for instance, an access telephone number and the line speed. MSAMs can define record attribute types to store slot configuration information.

A personal MSAM can manage more than one slot. For example, if a user had two accounts on an external messaging system of a given type, a personal MSAM would manage two slots, one for each of the user's accounts on that messaging system. A personal MSAM also can connect to more than one external messaging system. For example, if a user has an account on each of two independent messaging systems, the same personal MSAM can connect to each system and manage a slot for the user's account there.

Each mail slot that a personal MSAM manages has two queues: an incoming queue and an outgoing queue.

Each messaging slot that a personal MSAM manages has an outgoing queue. The notion of an incoming queue does not apply to messaging.

An **incoming queue** contains AOCE letters that the personal MSAM translates from mail received from its external messaging system and each letter's associated message summary. (See the section "MSAM Modes of Operation" beginning on page 2-12 for information about message summaries.) An **outgoing queue** contains messages that the personal MSAM must deliver to an external messaging system. A personal MSAM retrieves a message from an outgoing queue, translates it, and delivers it to the intended recipients on the external messaging system.

Note that any given queue contains either letters and message summaries or non-letter messages. It does not contain both. Figure 2-4 shows an example of a personal MSAM with three slots and their associated queues.

Figure 2-4   Personal MSAM with its slots and queues
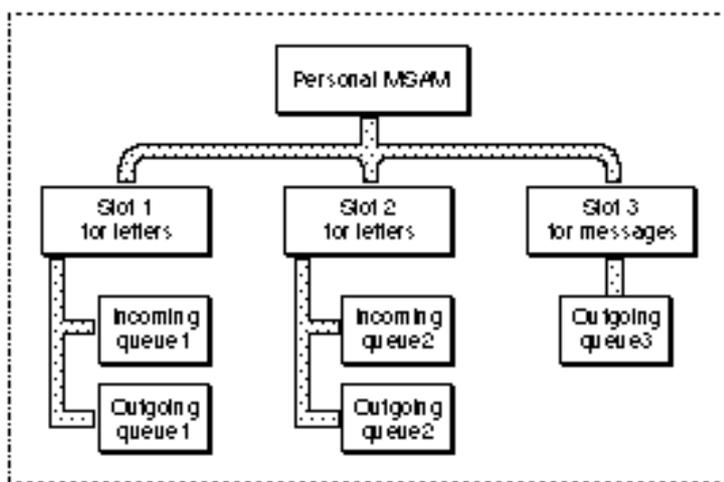
IMPORTANT

In release 1 of the AOCE software, the handling of non-letter messages is
not fully supported for personal MSAMs. Therefore it is not advisable
for a personal MSAM to implement the transfer of non-letter messages
using release 1 of the AOCE software. s

Server MSAMs

A server MSAM allows users and processes on an AppleTalk network to exchange
messages with other users and processes on one or more external messaging systems. It
serves its clients indirectly by acting as a conduit for messages between a PowerShare
mail server and the external systems to which the MSAM is connected. It must run on
the same Macintosh as its PowerShare mail server.

Server MSAMs route messages between different messaging systems rather than between
individual accounts on those systems. Therefore, a server MSAM does not necessarily
need to know about specific accounts on an external messaging system, and, as a result, it
has no concept of slots.

A server MSAM can connect to different types of messaging systems. For instance, a
single server MSAM might connect to one or more Simple Mail Transfer Protocol
(SMTP), X.400, and X.500 systems.

A server MSAM is a foreground Macintosh application. Once a server MSAM is
launched, it should run continuously.

(A server MSAM and its PowerShare mail server do not have to run on a dedicated
Macintosh. However, performance of other applications on the same Macintosh may
suffer when the MSAM and server are very busy.)

Table 2-1 summarizes the differences between personal MSAMs and server MSAMs.
(Not all of the differences have been discussed at this point.) You may want to refer to
this table as you read succeeding sections in this chapter.

Table 2-1   Differences between personal MSAMs and server MSAMs

Characteristic	Personal MSAM	Server MSAM
Application type	Background-only	Foreground
Interconnects	User/process to specific account	Multiple users/ processes to messaging system
Needs specific account information	Yes	No
Uses slots	Yes	No

Table 2-1   Differences between personal MSAMs and server MSAMs (continued)

Characteristic	Personal MSAM	Server MSAM
Queues	1 outgoing queue per slot; 1 incoming queue per mail slot	1 outgoing queue
Writes message summaries	Yes	No
Can write incoming letters on demand	Yes	No
Needs setup template	Yes	No
Needs address template	Yes	Yes
Runs on	A user's Macintosh	A server Macintosh with a PowerShare mail server
Must be connected to an AppleTalk network	No	Yes
Transfers messages for more than one user	No	Yes
Mode of operation	Standard, online, quasi-batch	Standard
File type	`'csam'` or `'msam'`	`'APPL'`
Linked to its catalogs through	Mail Service and Catalog records in the Setup catalog	Foreign dNodes in AOCE catalog
Represented by	MSAM record in the Setup catalog	Forwarder record

MSAM Modes of Operation

In addition to its type (either personal or server), another important characteristic of an MSAM is its mode of operation. *Mode of operation* refers to the degree of control an MSAM retains over messages that it puts into an AOCE system. Some MSAMs function in some respects like a standard store-and-forward gateway; others function as an agent for the user. This section explains these modes in more detail.

The store-and-forward gateway model consists of a source messaging system, a series of one or more store-and-forward gateways, and a destination messaging system. A **store-and-forward gateway** links different systems, providing temporary data storage and, where necessary, address translation. Figure 2-5 illustrates the store-and-forward gateway model. In such a model, the gateway hands off a message to the next link in the store-and-forward chain. Once it transfers a message, its responsibility for (and control of) that message ends. MSAMs that operate in this fashion are said to operate in **standard mode.**

Figure 2-5   Store-and-forward gateway model
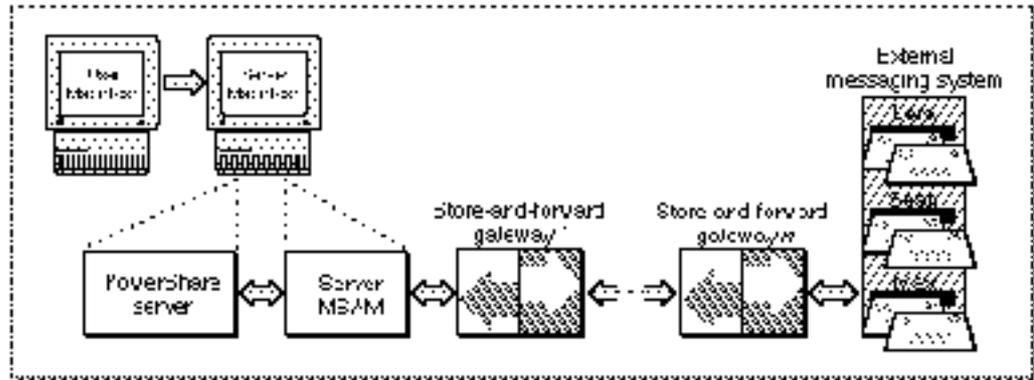
The online model consists of a source messaging system, a destination messaging system, and a personal MSAM that acts as an agent for the user in connecting those systems. In the online model, a personal MSAM does not act simply as a link in a series of store-and-forward gateways. Rather, it actively manages letters in a user's AOCE mailbox and in the user's accounts on external messaging systems, reflecting changes in one to the other, and keeping both ends synchronized as much as possible. Figure 2-6 illustrates the online model. MSAMs that operate in this fashion are said to operate in **online mode.** A personal MSAM operating in online mode can affect the user's experience quite directly, something an MSAM operating in standard mode cannot do.

Figure 2-6   Online model
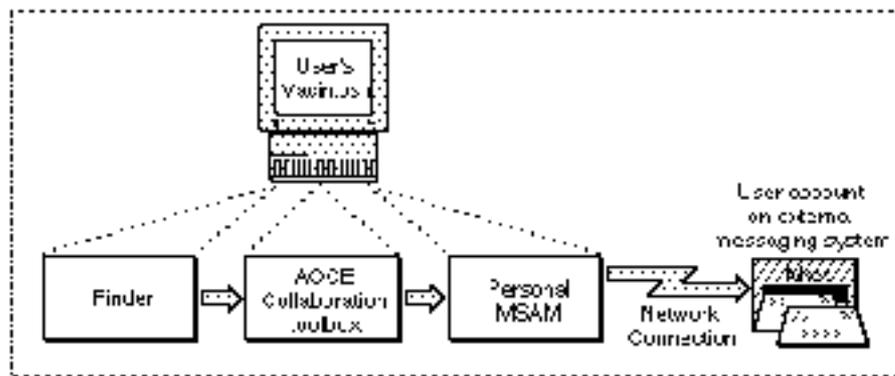
A significant difference between standard mode and online mode is the point at which the MSAM is active. In standard mode, an MSAM is removed from any contact with the user. In online mode, the MSAM is actively involved with the user experience through the MSAM API and Finder interface.

A server MSAM always operates in standard mode. It delivers messages to a PowerShare mail server, at which point the MSAM's responsibility for the message ends. The AOCE system is responsible for delivering the message to its final destination. Similarly, from an AOCE system perspective, a server MSAM is a store-and-forward gateway in that messages sent to a server MSAM are addressed to a particular messaging system, not a specific address within that system.

A personal MSAM may operate in standard mode, online mode, or a variation of online mode referred to as *quasi-batch mode*. A personal MSAM always operates in standard mode when it is dealing with incoming non-letter messages. Much as a server MSAM hands off a message to a PowerShare mail server, the personal MSAM hands off a non-letter message to the IPM Manager resident on the Macintosh. Once it submits such a message to an AOCE system, the personal MSAM has no further control of or responsibility for the message. The AOCE system delivers the message to its final destination on the Macintosh. When a personal MSAM is dealing with incoming letters, however, it operates in online mode or quasi-batch mode.

IMPORTANT

A single personal MSAM may operate in both standard and online or quasi-batch modes; that is, it may handle both letters and non-letter messages. The MSAM API is general enough to cover all variations. As a result, the API contains features that do not apply in every case.

However, as noted earlier, the handling of non-letter messages is not fully supported for personal MSAMs in release 1 of the AOCE software. Therefore it is not advisable for a personal MSAM to implement the transfer of non-letter messages using release 1 of AOCE software. s

The AOCE software architecture allows a personal MSAM to operate in online mode (act as a user agent) by providing it with the means to deliver an incoming letter to a specific queue and to manipulate that letter after placing it in the queue.

The user's AOCE mailbox is a repository for letters from all of the different sources to which the user has access. These sources include an incoming queue for each mail slot managed by a personal MSAM installed on the Macintosh. On any given Macintosh with AOCE software installed, there are some number of destination queues for incoming messages, each of which contains either letters or non-letter messages. An incoming queue is a special type of destination queue for letters. It is special because a personal MSAM can manipulate an incoming queue and its contents. All other destination queues are under the control of the IPM Manager.

A personal MSAM submitting letters to an AOCE system must conform to certain minimal requirements of online mode. These requirements are to create, manage, and delete information blocks about the letters that it puts into an incoming queue. The information blocks are called **message summaries.** The AOCE Mailbox extension to the Finder uses message summaries to display information about the letters to the user. Message summaries are also the means by which a personal MSAM reflects changes in the status of a letter from the local Macintosh computer to the remote system and vice versa. Only personal MSAMs create message summaries for incoming letters.

Before a personal MSAM puts a letter into an incoming queue, it must first create
the letter's message summary and put it into the incoming queue. A message
summary contains

n information that is needed to display the letter to the user (this includes the subject of
   the letter, its timestamp, the sender's name, and so forth)

n status information, such as whether the user has read the letter or deleted the letter (a
   personal MSAM uses the status flags to maintain consistency between the letter's
   status on an AOCE system and on an external system)

n state information about the letter, such as whether the letter itself currently exists in
   the incoming queue

n whatever private data that you wish to attach to this letter (for instance, you may
   want to store the ID or reference number that uniquely identifies the letter on the
   external messaging system)

A message summary is defined by the `MSAMMsgSummary` structure, described on
page 2-127.

After creating and submitting a message summary for a letter, a personal MSAM may
immediately translate the letter into the AOCE letter format and put it into the incoming
queue. Alternately, the MSAM can delay writing the letter until the user actually opens
it. (The MSAM receives a high-level event when a user opens a letter.)

In general, a personal MSAM that connects to an external messaging system over a slow
link should create the message summary and put the letter into the incoming queue at
the same time. This gives a user faster access to the letter when he or she decides to
read the letter. Also, when a link is slow or expensive, the MSAM might keep the copy of
the letter the user has already read to avoid a retransmission if the user wants to read the
letter again.

A personal MSAM that connects to an external messaging system over a fast link such
as a local area network may choose to create just the message summary without auto-
matically translating and transferring the letter itself. The MSAM can retrieve the letter
on demand, that is, only when the user actually wants to read the letter. In these
circumstances, it can delete the letter after the user reads it because retransmission
would not cause much of a delay.

A personal MSAM may implement some features of online mode but not all, and it may
thus operate somewhere in between standard and online modes. **Quasi-batch mode**
represents a continuous gradation between standard and online modes. In quasi-batch
mode, a personal MSAM may simply create a message summary, transfer the letter to an
AOCE system, and do nothing further with regard to the letter. For example, a personal
MSAM for fax transmissions might simply download a fax and put it into the incoming
queue. Such a personal MSAM complies with only the minimal requirements of online
mode and operates as much as possible like a standard store-and-forward gateway.

Table 2-2 shows the types of operating modes available to server and personal MSAMs.

Table 2-2   MSAM operating modes

Operating mode	Type of MSAM
Standard	Personal MSAM (for non-letter messages) and server MSAM
Online	Personal MSAM (for letters)
Quasi-batch	Personal MSAM (for letters)

This section has described the incoming queue as a special queue for incoming letters, available only to personal MSAMs with mail slots. There is no analogous construct on the outgoing side. All MSAMs, personal and server alike, have an outgoing queue from which they obtain outgoing messages. A server MSAM has a single outgoing queue that contains all of the messages addressed to external messaging systems to which it is connected. A personal MSAM, regardless of its operating mode, has one outgoing queue for each of its slots. Each queue contains the outgoing messages for the associated slot.

Types of Messages

The following sections discuss messages, letters, and reports.

Basic Messages

A **message** is the basic unit of communication defined by the Interprogram Messaging (IPM) Manager. A message consists of a message header followed by zero or more **message blocks,** each of which is a sequence of any number of bytes. The **message header** contains control information about the message, such as the message creator and message type, the total length of the message, the time it was submitted, addressing information, and so forth. It also contains the length, creator, and type of each block in the message. For more detailed information on the structure of messages and more information on the IPM Manager and the services it provides, see the chapter "Interprogram Messaging Manager" in *Inside Macintosh: AOCE Application Interfaces.*

Every message has a message creator and a message type. The message creator and type are analogous to a Macintosh file's creator and type. The **message creator** indicates which application created the message. A **message type** indicates the semantics of the message, the type of blocks the message should contain, and the relationships among the various blocks in the message.

Similarly, every block has a block creator and a block type. The **block creator** indicates which application created the block. A **block type** indicates the format of the data contained within the block.

In addition to message types, AOCE software defines the concept of **message families.** A message that belongs to a message family shares a similar form with all other messages that belong to the same message family. Messages of the same family conform to the syntax of a defined set of message block types and their associated semantics. The syntax specifies which block types are optional and which are mandatory and specifies the relationships between the various blocks. Messages that belong to the same message family may also contain additional blocks whose types are not defined as part of the message family.

Apple defines three message families for an MSAM's use. All non-letter messages that an MSAM transfers belong to the `kIPMFamilyUnspecified` family. Letters may belong to either the `kMailFamily` or `kMailFamilyFile` family, both of which are defined in the next section. Although it is possible to distinguish a new class of messages by defining a new message family, it is not recommended that you do so.

IMPORTANT

Apple Computer, Inc., reserves all values for message and block types, message and block creators, and message families that consist entirely of lowercase letters and special characters. You are free to create and use other values except 0 and `'????'`. Apple Computer, Inc., does not provide a registry for message and block types, message and block creators, and message families. s

A message can contain another message. A message that is contained within another message is called a **nested message.**

Letters

A **letter** is a type of message, consisting of a defined set of message blocks, that is intended to be read by a person.

A letter must contain a **letter header block.** A letter header block contains the address of the sender and of each recipient. It also contains the letter's attributes.

Letter attributes are bits of information about a letter. They include such things as the time the letter was sent, the subject of the letter, the priority assigned to the letter by the sender, and so forth.

Note

In this chapter, letter attributes are usually referred to simply as *attributes.* Do not confuse these letter attributes with record attributes. A record attribute refers to a part of an AOCE record. For information about record attributes, see the chapters "AOCE Utilities" and "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces.* u

A letter may have blocks that contain letter content, a nested letter, enclosures, and an image of the letter content. The MSAM API provides functions that you can use to read and write most of these blocks without specifying the block type. For example, the function `MSAMPutContent` automatically creates a block of type `kMailContentType`. However, to add a block of type image (`kMailImageBodyType`) or a private data block

(kMailMSAMType), you need to provide the block type to the MSAMPutBlock function. Table 2-3 lists the AOCE-defined block types that a letter may contain and the functions you use to read and write a block of a given type.

Table 2-3   Predefined letter block types

Block type	Value	Block contents	To read/write
kMailLtrHdrType	'lthd'	Letter header	MSAMGetRecipients MSAMPutRecipient MSAMGetAttributes MSAMPutAttribute
kMailContentType	'body'	Body of letter	MSAMGetContent MSAMPutContent
kMailEnclosureListType	'elst'	List of enclosures	MSAMGetEnclosure MSAMPutEnclosure
kMailEnclosureDesktopType	'edsk'	Desktop Manager information for enclosures	MSAMGetEnclosure MSAMPutEnclosure
kMailEnclosureFileType	'asgl'	A file enclosure	MSAMGetEnclosure MSAMPutEnclosure
kMailImageBodyType	'imag'	Image of letter	MSAMGetBlock MSAMPutBlock
kMailMSAMType	'gwyi'	MSAM-defined information	MSAMGetBlock MSAMPutBlock
kIPMEnclosedMsgType	'emsg'	Nested letter	MSAMOpenNested MSAMBeginNested
kIPMDigitalSignature	'dsig'	Digital signature	MSAMGetBlock MSAMPutBlock

Letter content is that part of the letter that the sender typically wants the recipient to read first, like the body of a conventional hard-copy letter. Letter content may be in three forms:

n   a content block (block type is kMailContentType)

n   an image block (block type is kMailImageBodyType)

n   a content enclosure (block type is kMailEnclosureFileType)

A **content block** contains the body of a letter in one or more data segments. Each segment contains data of one of the following types:

n   Plain text. A text segment contains data in one or more character sets (Roman, Arabic, Kanji, and so on) with 1-byte or 2-byte character codes, depending on the character set.

n   Styled text. The segment contains text and a `StScrpRec` structure containing the style information for that text.

n   Pictures. The segment contains data in PICT format.

n   Sounds. The segment contains data in Audio Interchange File Format (AIFF).

n   Movies. The segment contains data in QuickTime movie file format ('`MooV`').

These five data formats are collectively called *standard content* or **standard interchange format,** sometimes referred to as *AppleMail format.* All MSAMs must support standard content to facilitate interoperability. Any user with AOCE software installed can read and write letters containing standard content using the AppleMail application.

Another way of communicating a letter's content is to include it in an **image block.** Data in an image block is stored in a structure of type `TPfPgDir` followed by picture elements (PICTs). The format of data in an image block is sometimes referred to as *snapshot format.*

The AppleMail application can read image blocks. Thus, by including an image block in a letter, an application that uses formats other than standard interchange format can ensure that a user having the AppleMail application can view the formatted content. A receiver cannot edit image data. MSAMs should support image blocks.

The third form in which letter content may be transmitted or received is a **content enclosure,** sometimes referred to as a *main enclosure.* Such an enclosure is typically in the native format of the sending application. An MSAM is not required to support translations of various application file formats. A recipient must have a copy of the sending application to read a content enclosure. A letter can have only one content enclosure.

The contents (if any) of a letter may be in any or all of these three forms. Typically, you can expect letters to contain a content block as well as a content enclosure.

An **enclosure** is a file or folder sent along with a letter. An enclosure may be either a regular enclosure or a content enclosure. A **regular enclosure** is a file or folder included in a letter like an attachment in a conventional hard-copy letter. That letter may or may not contain a content block.

A letter can have up to 50 enclosures. An enclosure file can be of any type. If an enclosure is a folder, it can contain any number of files of any type, so long as the total number of enclosures does not exceed 50. Each file and folder counts as one enclosure. For example, if a letter had as an enclosure a folder containing three files, the total number of enclosures in the letter is four: one folder and three files. A content enclosure counts when totaling the number of enclosures in a letter.

A **nested letter** is a complete letter included whole within another letter. A letter can
have only one letter nested within it. However, the nested letter itself may contain a
nested letter. Figure 2-7 illustrates this concept.

Figure 2-7   Nested letters
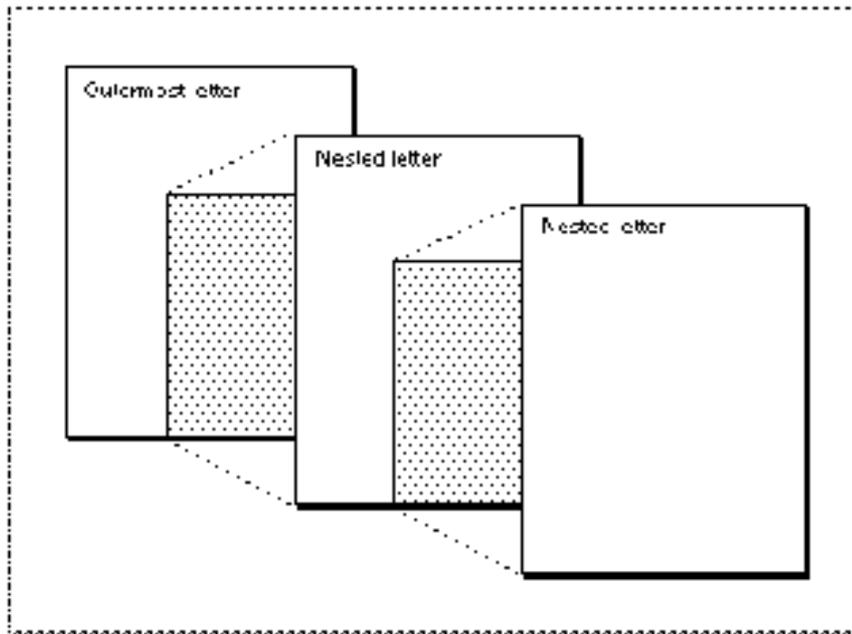
The **nesting level** of a letter indicates how many letters are nested within it. The nesting
level of a letter that contains no nested letters is 0. A letter that contains a letter with a
nesting level of n has a nesting level of $n + 1$. Thus, if a reply letter contains a copy of the
original letter, the nesting level of the reply is one greater than the nesting level of the
original letter. Figure 2-8 illustrates an example of nesting letters. Sue sends a memo to
Dan. Her original memo has a nesting level of 0. Dan replies to Sue and includes a copy
of Sue's original memo in the reply. His reply has a nesting level of 1. Sue sends a
different memo to Tim and includes Dan's reply. The nesting level of her memo to Tim is
2. The theoretical limit to the number of nesting levels is very large.

A forwarded letter is always a nested letter. It is nested within a letter that has no content and no enclosures. The letter that contains the forwarded letter has a nesting level of $n + 1$, where n is the nesting level of the forwarded letter.

Figure 2-8   How the nesting level increments
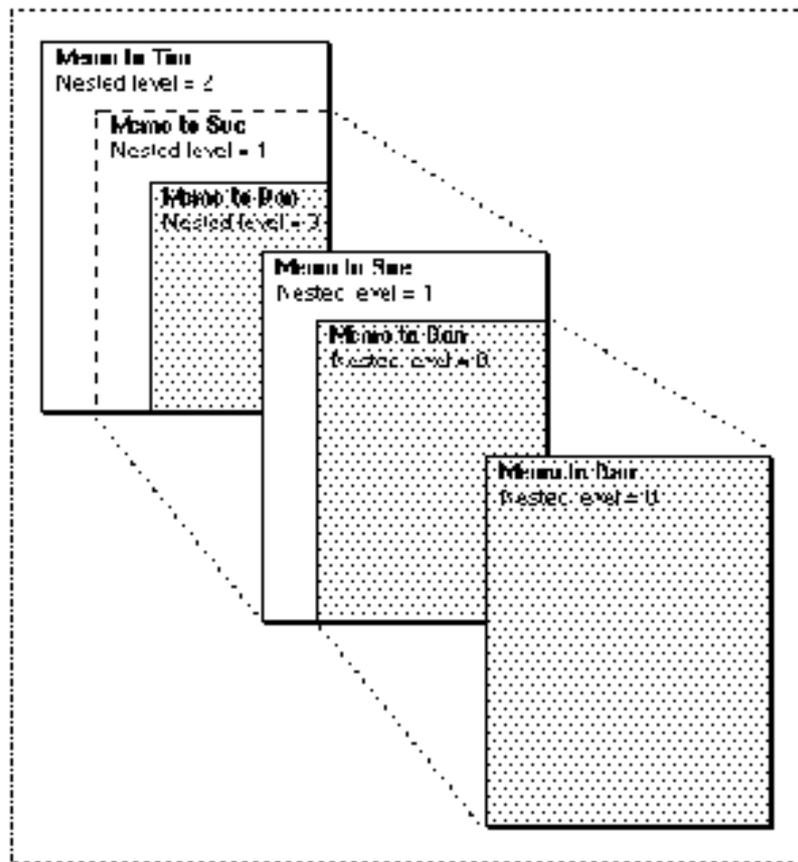
Figure 2-9 illustrates the structure of a hypothetical letter. In the message header, the message creator and type (`'lap2'` and `'lttr'`) indicate that this message is a letter that was created by the AppleMail application. Next is the letter header block. The letter header information includes the letter's nesting level, set to 2, indicating that this letter has two letters nested within it. The letter contains a content block. The blocks of type `kMailEnclosureListType` (`'elst'`) and `kMailEnclosureDesktopType` (`'edsk'`) are private to Macintosh system software. There are two enclosures in the letter, one of which is a content enclosure. An image block is present. It contains an alternate representation of the data in the content block. The letter also contains a nested letter in a

nested letter block. The nested letter is a complete letter consisting of a message header, a letter header block, a content block, and a nested letter block. Its letter header shows that its nesting level is 1. The nested letter block contains a complete letter consisting of a message header, a letter header block, and a content block. Its nesting level is 0.

Figure 2-9   Structure of a letter
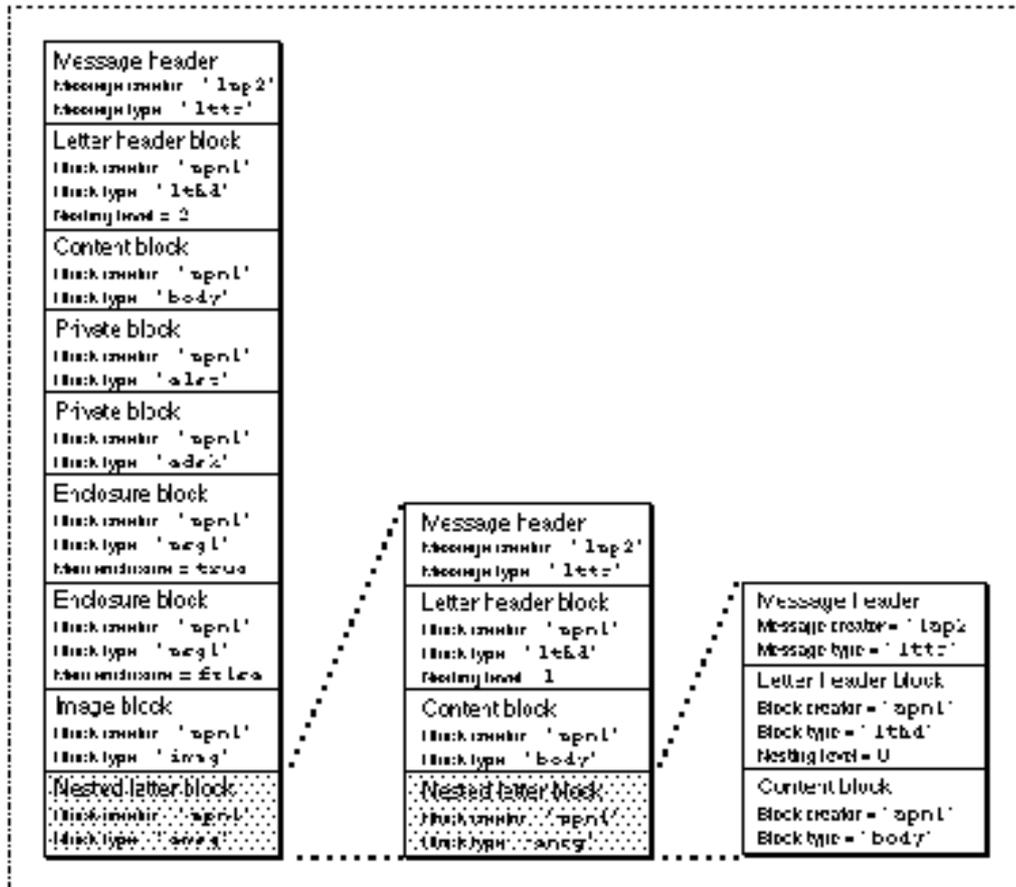
Ordinarily, letters belong to one of two message families defined by AOCE software. A letter that belongs to the kMailFamily family may contain either a content block or any type of enclosure or both. A letter that belongs to the kMailFamilyFile family does not contain a content block or a content enclosure, but may contain a regular enclosure. You should not put a content block into or expect to get a content block from a letter in the kMailFamilyFile family.

Reports

A **report** communicates delivery information about a message to the sender of the message. A report, like a letter, is a message with a defined set of message blocks.

The sender of a message can request information about successful delivery of the message, failure to deliver the message, or both, for a message. The sender's request applies to all of the message's recipients.

A single report may contain information about the outcome of delivery attempts to one or more recipients of a message; that is, it may contain delivery indications, non-delivery indications, or both. A **delivery indication** indicates the successful delivery of a specific message to one or more specified recipients. A **non-delivery indication** indicates failure to deliver a specific message to one or more specified recipients. A delivery or non-delivery indication is sometimes referred to as a *recipient report*.

An MSAM can both create a report about an outgoing message and receive a report about an incoming message.

Note

A report that an MSAM creates or receives (an MSAM report) differs somewhat from a report created or received by other clients of the IPM Manager (an IPM report). An IPM report may contain a copy of the original message, but an MSAM report never does. An IPM report goes directly to an IPM Manager client. An MSAM report goes to an AOCE agent, which interprets the information in the MSAM report and creates an IPM report to send to the ultimate report recipient. u

The sections "Generating a Report" on page 2-61 and "Receiving a Report" on page 2-80 describe how an MSAM generates and receives reports. For information on IPM reports, see the chapter "Interprogram Messaging Manager" in *Inside Macintosh: AOCE Application Interfaces*.

AOCE Addresses

The AOCE software architecture provides for the exchange of messages among different types of messaging systems. The exchange of messages requires a way of uniquely specifying the sender and receiver of a message. This unique specification is called an *address*. This section discusses the syntax and semantics of the AOCE address structure.

To provide connectivity between AOCE messaging systems and other messaging systems, the AOCE address structure is designed to accommodate already existing address formats, in addition to address formats that may be developed for future messaging systems.

One way that messaging systems can be differentiated is by the syntax and semantics of their addresses. Messaging systems that share the same addressing conventions are said to be of the same type.

An address is unique within a messaging system. To exchange messages between messaging systems, a sender must specify an address plus the messaging system in which the address is unique.

At the most general level, you can think of an AOCE address structure as having two parts: a messaging system specifier and an entity specifier that uniquely identifies a person or process within that messaging system. When an address specifies a recipient within an AOCE messaging system, the AOCE software delivers the message to the specific address. When an address specifies a recipient in a non-AOCE messaging system, the AOCE software delivers the message to the MSAM responsible for that messaging system.

For AOCE routing software, the basic problem can be stated as follows: assume an external messaging system is named *System X*. System X contains many addressable entities (users and processes). To send a message to an entity Y in System X, AOCE needs a way to say "Y in System X." AOCE doesn't care what Y is. Y is internal to, and should be unique in, System X.

Figure 2-10 shows an AOCE messaging system, an AppleLink system, and two SMTP systems. (SMTP stands for **Simple Mail Transfer Protocol.** Computers connected to the Internet often use SMTP to exchange messages.) Within this environment, AOCE routing software needs a way to specify each messaging system. Each messaging system is partially described by a four-character extension type. An **extension type** identifies a type of messaging system that uses a specific addressing convention—for example, an AppleLink system or an X.400 system. Because there can be more than one messaging system of a given type, an address based on the extension type alone is not sufficient to distinguish between two or more messaging systems of the same type. In the illustration, AOCE routing software could not distinguish between the two SMTP systems on the basis of type. To solve this problem, AOCE software requires that each messaging system have a unique name by which it is known within an AOCE system. In Figure 2-10, the names Felines and Canines distinguish between the two SMTP messaging systems.

Figure 2-10   AOCE system connected to external messaging systems
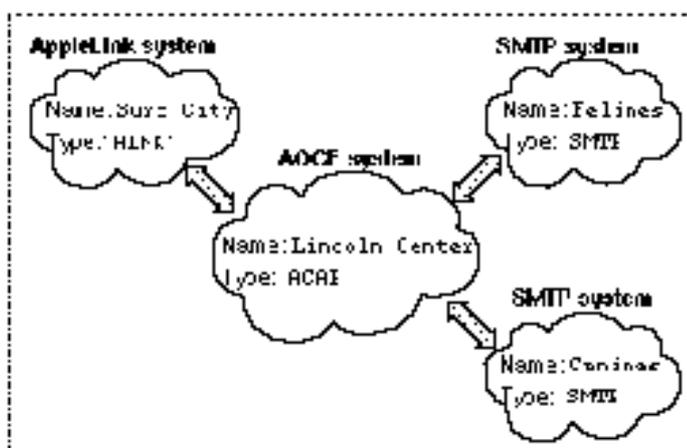
In some cases, there is only one messaging system of a given type, and the messaging system already has a unique, well-known name. The Internet is a good example of this. In cases like this, if your MSAM provides a preassigned name, it should use the well-known name. A unique name for each messaging system is fundamental to AOCE addressing.

Within some messaging systems, multiple address formats are allowed. The Internet, for example, accepts both UUCP and SMTP addresses. An Internet MSAM has one unique name associated with it, but it may service multiple extension types, one for each form of Internet address that it knows how to translate.

Note

There is no registry for extension types. If you want to use an existing extension type, you are responsible for ensuring that the extension type always represents the same address syntax and semantics. If you want to create a new extension type, it is recommended that you use your application's signature type, registered with Macintosh Developer Technical Services, to ensure uniqueness.   u

Before describing an AOCE address structure, it is helpful to understand a little about how the AOCE software implements unique names for messaging systems. Within an AOCE system, each external messaging system is associated with a unique catalog name. The catalog name identifies to AOCE software the messaging system and the set of addresses that belong to that messaging system.

For server MSAMs, the AOCE system administrator creates a reference to an external messaging system by creating a dNode, sometimes called a **foreign dNode,** in an AOCE catalog. Figure 2-11 illustrates the addition of a dNode that represents an external messaging system. The original AOCE configuration has a catalog named Catalog A that contains dNodes named Artists Unlimited and Legal Services. AOCE software routes messages only among addresses in Catalog A. There exists an external messaging system called TriColor Labs. People within the original AOCE messaging system may want to communicate with people who are accessible only via the TriColor Labs messaging system. A server MSAM is installed within the AOCE system to extend the messaging environment to include people within the TriColor Labs messaging system. The AOCE system administrator creates a new dNode representing the TriColor Labs system and gives the dNode a unique name, TCL, within Catalog A. AOCE software still routes messages only among addresses in Catalog A, but Catalog A now includes a new set of addresses represented by the dNode TCL.

Figure 2-11   Adding a dNode for a messaging system
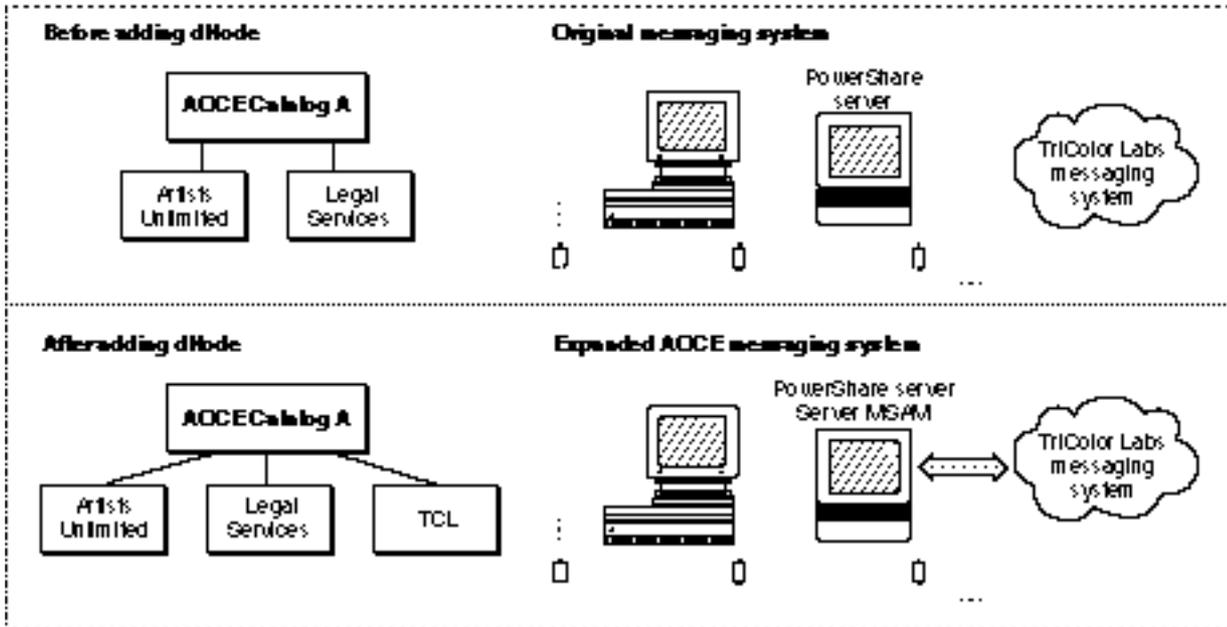
For personal MSAMs, the PowerTalk Key Chain creates a Catalog record in the Setup catalog to represent the set of addresses belonging to a given messaging system. See the chapter "Service Access Module Setup" for more information.

The name that uniquely identifies an external messaging system in an AOCE system is the name of the dNode (for server MSAMs) or the name of the Catalog record in the Setup catalog (for personal MSAMs).

Figure 2-12 illustrates the following points about MSAMs, messaging system names, and extension types:

n   An external messaging system must have a unique name.

n   Different MSAMs may connect to different external messaging systems of the same extension type.

n   A single MSAM may connect to more than one external messaging system, each having a different extension type (it may also connect to more than one external messaging system having the same extension type).

n   A single external messaging system may have more than one extension type.

Figure 2-12   MSAMs, messaging system names, and extension types
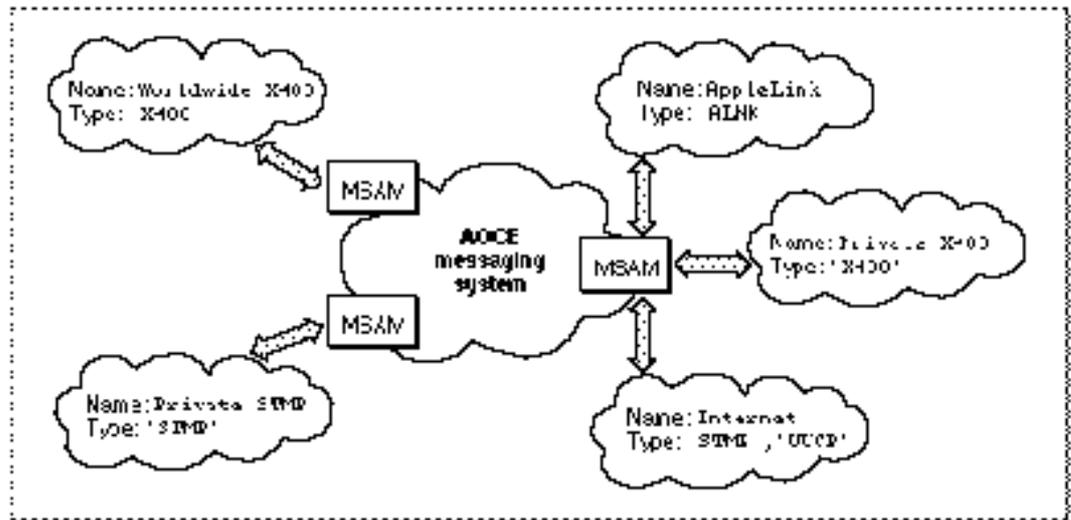
Now look at the AOCE address structure. AOCE software already defines a `RecordID` structure to uniquely identify a record in an AOCE catalog. This structure is adapted and extended for use as an address structure. In an AOCE messaging system, an address is specified as an `OCERecipient` structure, which is identical to a `DSSpec` structure.

```
struct DSSpec {
   RecordID           *entitySpecifier;
   OSType             extensionType;
   unsigned short     extensionSize;
   Ptr                extensionValue;
   };

typedef DSSpec OCERecipient;
```

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

Figure 2-13 shows an exploded view of an `OCERecipient` structure. An AOCE address is a two-level specification that first identifies a messaging system and then identifies an individual entity within it. This is roughly analogous to an address on a piece of hard-copy mail that specifies a large organization and a mailstop within it. The postal service uses part of the address—organization name, street number, city, state, and zip code—to deliver the mail to the organization. The organization itself uses the remainder of the address, the mailstop number, to deliver the mail to a specific internal address. With an AOCE address, the `OCERecipient.entitySpecifier.rli` substructure identifies the messaging system. The value pointed to by the `OCERecipient.extensionValue` field identifies the individual entity within that messaging system.

Figure 2-13   Exploded view of an `OCERecipient` structure
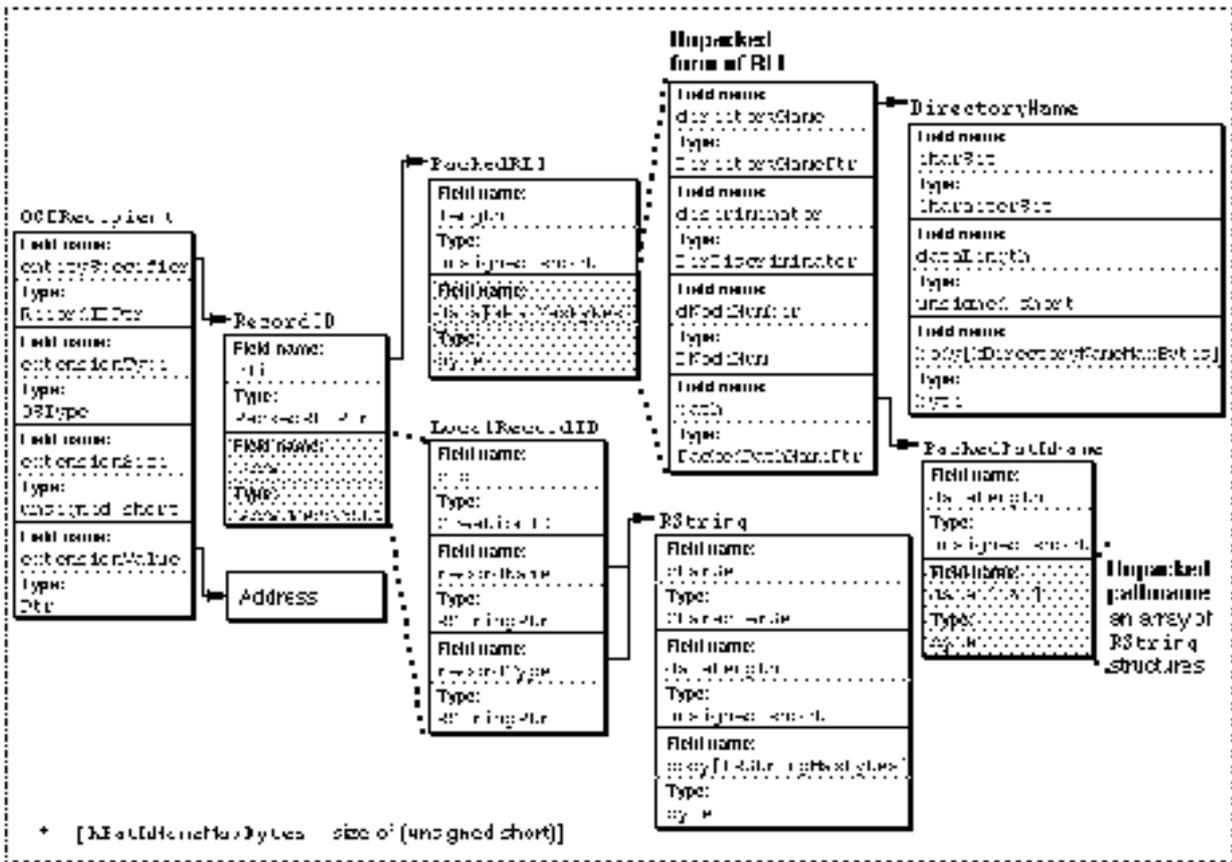
Table 2-4 lists the elemental fields of the address structure and the type of information each field contains when it is used to specify an address on an external system. The structure identifies both the external system and a specific sender or receiver within it that is the source or destination of a message.

Table 2-4 External address: Contents of an `OCERecipient` structure

Field name	Contents
directoryName	A pointer to an `RString` structure containing the unique name of a catalog in the AOCE environment. The name identifies the external messaging system to AOCE. The name is limited to 32 characters.
discriminator	An 8-byte value that further describes the catalog. The first 4 bytes indicate the extension type of the associated messaging system, for example, ALNK or SMTP. It is the same as the value in the `extensionType` field. The second 4 bytes are private to the catalog.
dNodeNumber	Unused. Set to 0.
path	Unused. Set to `nil`.
cid	Unused. Set to 0.
recordName	A pointer to an `RString` structure containing the name of the sender or receiver. This should be a displayable string.
recordType	A pointer to an `RString` structure containing the type of the sender or receiver—for example, "user" or "group". This should be a displayable string.
extensionType	The four-character extension type that specifies a type of messaging system, for example, `'ALNK'` or `'SMTP'`. The extension type is the same as the first 4 bytes of the associated catalog's discriminator value.
extensionSize	The length, in bytes, of the `extensionValue` field.
extensionValue	A pointer to a buffer that contains the address of the sender or receiver on the external system. The address is used only by the MSAM. Its content and format are not examined by AOCE software. However, for the type-in addressing feature in the mailer to work, the address must be a single `RString` structure.

Table 2-5 lists the elemental fields of the `OCERecipient` structure and the type of information each field contains when it is used to specify an address within an AOCE system.

Table 2-5   AOCE address: Contents of an `OCERecipient` structure

Field name	Contents
directoryName	A pointer to an `RString` structure containing the name of the PowerShare catalog that contains the record representing the sender or receiver. The name is limited to 32 characters.
discriminator	The discriminator value of the catalog that contains the record representing the sender or receiver.
dNodeNumber	A value that identifies the dNode that contains the record representing the sender or receiver. Set to 0 if you use the `path` field to specify the dNode.
path	A pointer to a buffer that contains the names of all of the dNodes on the path from the catalog node in which the sender or receiver record resides, up to the catalog root node. Set this field to `nil` if you use the `dNodeNumber` field to identify the dNode.
cid	The creation ID of the record that represents the sender or receiver.
recordName	A pointer to an `RString` structure containing the name of the sender or receiver. This is a displayable string.
recordType	A pointer to an `RString` structure containing the type of the sender or receiver. It tells you what the entity is, such as a user. This is a displayable string.
extensionType	A four-character extension type that specifies the format of the data pointed to by the `extensionValue` field. AOCE defines the following extension types: `kOCEalanXtn`, `kOCEentnXtn`, `kOCEaphnXtn`.
extensionSize	The length, in bytes, of the `extensionValue` field.
extensionValue	A pointer to a buffer that contains the address of the sender or receiver on the AOCE system. The address is used only by the AOCE software. Its content and format need not be examined by the MSAM.

Table 2-6 lists the extension types for addresses within an AOCE messaging system. These extension types are discussed in more detail in the chapter "Interprogram Messaging Manager" in *Inside Macintosh: AOCE Application Interfaces.* You do not need to understand the semantics of the extension types. You do need to be sure that a recipient to whom you transmit a message from an AOCE system can reply to the message. Your MSAM might include the extension information with the outgoing message and reconstruct it when it submits the reply to the AOCE system. Alternatively,

your MSAM might maintain mapping tables to convert between addresses within the AOCE messaging system and external addresses. In this way, it can avoid sending to its external system information that is only relevant inside an AOCE system. This implementation decision is up to you.

Table 2-6   AOCE extension types

Constant	Value	Description
kOCEalanXtn	'alan'	Indicates an `EntityName` structure (an NBP name) plus a queue name in the form of a Pascal string. It is used for an address accessible on the local AppleTalk network.
kOCEentnXtn	'entn'	Indicates a `DSSpec` structure. It is used for an address accessible through a PowerShare mail server.
kOCEaphnXtn	'aphn'	Indicates a structure that specifies an address accessible by telephone.

Before you submit an incoming message to AOCE, you must construct `OCERecipient` structures containing the addresses of the sender and each of the recipients. Table 2-4 on page 2-29 describes the information you must provide in each field of the address structure for (a) the sender from your external messaging system and (b) any recipient in an external messaging system. Table 2-5 on page 2-30 describes the information you must provide in each field of the address structure for a recipient within the AOCE system.

When you read an outgoing message from AOCE, you must translate the `OCERecipient` structures that contain the address information for the sender and each of the recipients into a format that your external messaging system understands. Table 2-4 on page 2-29 describes what information you will find in each field of the address structure when the structure specifies a recipient on an external messaging system. Table 2-5 on page 2-30 describes the information contained in each field of the address structure when the structure specifies the sender of an outgoing message or a PowerTalk recipient.

The address of a recipient in an AOCE messaging system might include only the entity specifier portion of the `OCERecipient` structure; that is, it may not have any data in the `extensionType`, `extensionSize`, and `extensionValue` fields. This form is called an *indirect address* because it is not actually an address but points to a record in an AOCE catalog that contains the address. It uniquely identifies the messaging system and provides a displayable name and type to identify the sender or receiver. The direct form of an address always includes both the entity specifier and the extension information. The extension information gives a more detailed form of address. Addresses in external messaging systems are always in the direct form. Addresses in PowerShare catalogs may be in either the direct or indirect form. For more information about direct and indirect addressing, see the chapter "Interprogram Messaging Manager" in *Inside Macintosh: AOCE Application Interfaces.*

Table 2-7 shows examples of the content of the fields of an `OCERecipient` structure for an indirect AOCE address and an SMTP address.

Table 2-7   Sample addresses

`OCERecipient` fields	AOCE system (indirect address form)	SMTP system
directoryName	Engineering	Finance
discriminator	ACAP1234	SMTP0000
dNodeNumber	6	0
path	nil	nil
creationID	44894489	00000000
recordName	Joe Bernard	Suzy Durksen
recordType	aoce User	aoce User
extensionType	Not applicable	'SMTP'
extensionSize	Not applicable	16
extensionValue	Not applicable	Suzy@finance.com

When the `entitySpecifier` portion of the `OCERecipient` structure contains information about a sender or receiver on an external system, that information does not specify a record in a PowerShare catalog that represents the sender or receiver. However, when the structure contains information on a sender or receiver inside an AOCE messaging system, it does specify an existing record.

With your MSAM, you need to provide a special kind of AOCE template, called an *address template*, that allows a user to enter address information. Basic information about AOCE templates is provided in the chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces*. Specific information about address templates is provided in the chapter "Service Access Module Setup" in this book.

AOCE High-Level Events

Both personal and server MSAMs must be prepared to receive and respond to high-level events defined by AOCE software. The chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* describes the use of high-level events in detail; that information is not repeated in this section.

Personal MSAMs may receive the following high-level events:

Constant	Event ID	Description
kMailEPPCCreateSlot	'crsl'	Slot created
kMailEPPCModifySlot	'mdsl'	Slot modified
kMailEPPCDeleteSlot	'dlsl'	Slot deleted
kMailEPPCMailboxOpened	'mbop'	User opened mailbox
kMailEPPCMailboxClosed	'mbcl'	User closed mailbox
kMailEPPCMsgPending	'msgp'	Messages waiting to be sent
kMailEPPCSendImmediate	'sndi'	Send letter now
kMailEPPCShutDown	'quit'	Shut down operations and quit
kMailEPPCContinue	'cont'	Resume operation after error fixed
kMailEPPCSchedule	'sked'	Time for scheduled activity
kMailEPPCInQUpdate	'inqu'	Incoming queue updated
kMailEPPCMsgOpened	'msgo'	User opened letter
kMailEPPCDeleteOutQMsg	'dlom'	Delete outgoing queue message
kMailEPPCWakeup	'wkup'	Launched due to wakeup
kMailEPPCLocationChanged	'locc'	System location changed

Server MSAMs may receive these high-level events:

Constant	Event ID	Description
kMailEPPCAdmin	'admn'	Server administration function
kMailEPPCMsgPending	'msgp'	Messages waiting to be sent

Detailed descriptions of these events can be found in the section "High-Level Events" beginning on page 2-220.

When an MSAM receives an AOCE high-level event, it manipulates a standard EventRecord structure (defined in the chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*). The fields of an event record associated with an AOCE high-level event have a particular meaning.

```
struct EventRecord {
    short       what;
    long        message;
    long        when;
    long        where;
    short       modifiers;
};
```

Field descriptions

what	Always contains the constant kHighLevelEvent.
message	Always contains the event class kMailAppleMailCreator.

when	Unused.
where	Contains the event ID that identifies a specific event—for example, `kMailEPPCAdmin`.
modifiers	For personal MSAMs, this field contains the slot ID when the event applies to a particular slot; otherwise, it is set to 0. Server MSAMs can ignore this field.

Some AOCE high-level events require more information than that provided in the event record. After you receive such an event, you should call the `AcceptHighLevelEvent` function to get the additional data associated with the event. The additional data is in the form of a `MailEPPCMsg` structure.

A `MailEPPCMsg` structure consists of a version number and a union field. The union field may have any of the following contents: a pointer to an `SMCA` structure; a letter sequence number; a `MailLocationInfo` structure.

The version number indicates the version of the event. The MSAM should compare the version number in the `MailEPPCMsg` structure with `kMailEPPCMsgVersion`. If they are not the same, software incompatibilities may exist between the PowerTalk software and the MSAM, and there is no guarantee that the `MailEPPCMsg` structure used by the MSAM and by the IPM Manager are the same. The MSAM should ignore the event.

Most of the AOCE high-level events are informational in nature. For example, a `kMailEPPCMsgPending` event tells an MSAM that it has a new outgoing message. Informational events sent by the IPM Manager are not guaranteed to be received by the MSAM. The MSAM should consider these events as hints; that is, it should not rely on them as the only mechanism to initiate an action. For example, to make sure it transfers outgoing messages in a timely manner, it could check its outgoing queues every 20 minutes, each time it is launched, and each time it receives a `kMailEPPCMsgPending` event.

A few events are more than informational in nature. An MSAM must receive the `kMailEPPCCreateSlot`, `kMailEPPCModifySlot`, `kMailEPPCDeleteSlot`, `kMailEPPCMsgOpened`, and `kMailEPPCSendImmediate` events in order to take the relevant actions. For these events, the `MailEPPCMsg` structure contains a pointer to an `SMCA` structure. The MSAM needs to set the `result` field of the `SMCA` structure to acknowledge the event or to report the outcome of its effort to handle the event. Additionally, the IPM Manager informs the client if the event does not reach the MSAM. (An MSAM cannot acknowledge or set a result for an event whose `MailEPPCMsg` structure does not contain a pointer to an `SMCA` structure.)

Once the MSAM sets the `result` field to acknowledge the event or to signal completion, the `SMCA` structure is no longer valid.

An MSAM defines the error codes that it returns in response to the `kMailEPPCCreateSlot`, `kMailEPPCModifySlot`, `kMailEPPCDeleteSlot`, and `kMailEPPCMsgOpened` events. For the `kMailEPPCSendImmediate` event, it typically should return the `kMailSlotSuspended` or `kMailTooManyErr` result code.

System Location

The concept of location serves users with mobile Macintosh computers. Personal MSAMs must understand the concept of location, whereas server MSAMs need not. A personal MSAM, residing on a user's Macintosh, must be aware of the possibility that the system location may change. For instance, a personal MSAM installed on a PowerBook may be launched at different locations, such as the user's business office, the user's home, a customer site, an airport, and so forth. The personal MSAM is likely to be affected by such changes of location. A fax MSAM, for example, would use different telephone numbers when running at home or in the office; an Internet MSAM cannot work if a TCP/IP network connection is not available.

After it is launched, a personal MSAM gets the current system location from the Setup record in the Setup catalog. Then it determines, for each slot, whether the slot is active at that location by checking the location flags in the slot's standard slot information. See the section "Initializing a Personal MSAM" on page 2-37 for a description of how you do this.

If a slot is not active at the current location, the personal MSAM should not perform any work on behalf of that slot. If none of the personal MSAM's slots are active at the current location, the MSAM should quit.

If the system location changes, the IPM Manager sends the MSAM one `kMailEPPCLocationChanged` high-level event for each slot. The event tells the MSAM the slot to which it applies, the current system location, and the location flags for the slot. If the location flags show that the slot is inactive at the current location, the MSAM should immediately stop performing any activity on behalf of the slot, such as downloading or sending letters.

A user can activate or deactivate a mail slot in a given location. In response, the IPM Manager updates the location flags in the `MailStandardSlotInfoAttribute` structure for that slot and sends a `kMailEPPCLocationChanged` high-level event to the MSAM. At that point, the MSAM needs to determine if the slot is active at the current location. If the slot is active, the MSAM should continue to act for the slot; if it is not, the MSAM should cease acting for the slot.

Using the MSAM API

This section shows you how to

n   determine whether the Collaboration toolbox is available

n   launch a personal MSAM

n   initialize personal and server MSAMs

n   transfer an outgoing letter from an AOCE system to another messaging system

n transfer an incoming letter from another messaging system to an AOCE system

n delete a message

n translate addresses

n log personal MSAM operation errors

Determining Whether the Collaboration Toolbox Is Available

Before calling any of the functions in the MSAM API, a server MSAM should verify that the Collaboration toolbox is available by calling the `Gestalt` function with the selector `gestaltOCEToolboxAttr`. If the Collaboration toolbox is present but not running (for example, if the user deactivated it from the PowerTalk Setup control panel), the `Gestalt` function sets the bit `gestaltOCETBPresent` in the `response` parameter. If the Collaboration toolbox is running and available, the function sets the bit `gestaltOCETBAvailable` in the `response` parameter. The Gestalt Manager is described in the chapter "Gestalt Manager" in *Inside Macintosh: Operating System Utilities*. Because a personal MSAM is launched by the IPM Manager, it can assume that the Collaboration toolbox is available.

If you want to be informed when the IPM Manager starts up or shuts down, you can install an entry in the AppleTalk Transition Queue (ATQ). Then the AppleTalk Link-Access Protocol Manager calls your ATQ routine with the transition selector `ATTransIPMStart` when the IPM Manager has finished starting up and with the selector `ATTransIPMShutdown` when the IPM Manager has started to shut down. The ATQ is described in the "Link-Access Protocol (LAP) Manager" chapter in *Inside Macintosh: Networking*.

Determining the Version of the IPM Manager

To determine the version of the IPM Manager that is available, call the `Gestalt` function with the selector `gestaltOCEToolboxVersion`. The function returns the version number of the Collaboration toolbox in the low-order word of the `response` parameter. For example, a value of 0x0101 indicates version 1.0.1. If the Collaboration toolbox is not present and available, the `Gestalt` function returns 0 for the version number. You can use the constant `gestaltOCETB` for AOCE Collaboration toolbox version 1.0.

Launching a Personal MSAM

A personal MSAM must be launched by the IPM Manager. If you launch a personal MSAM in any other manner, it will not work properly with the IPM Manager.

If a personal MSAM is not already running, the IPM Manager launches it in response to any of the following events:

n The MSAM's setup template calls the `MailCreateMailSlot` or `MailModifyMailSlot` function.

n An application calls the `MailWakeupPMSAM` function.

n The MSAM's scheduled send or receive time occurs, or its send/receive time
   interval elapses.

Initializing a Personal MSAM

Before the IPM Manager launches a personal MSAM for the first time, the setup template
you provide with your personal MSAM must obtain information about the MSAM,
the accounts on external messaging systems to which it will connect, and the catalogs
associated with those external messaging systems. It gets this information from the user
and stores it in the Setup catalog.

Once launched, a personal MSAM needs to obtain a variety of information, much of it in
the Setup catalog. The information includes:

n the current system location

n information about each slot for which it is responsible (each slot represents one
   account on a messaging system)

n the incoming and outgoing queue references for each of its slots

n any additional configuration or private information it may require

A personal MSAM obtains much of the necessary information by reading records in the
Setup catalog. It then often copies this information into private structures.

The following steps illustrate a typical sequence of actions your MSAM can take to
obtain the necessary startup information after it has been launched:

1. Get the creation ID of the MSAM's record in the Setup catalog by calling the
    `PMSAMGetMSAMRecord` function. Build a record ID that contains your MSAM's
    record creation ID.

2. Get the local identity by calling the `AuthGetLocalIdentity` function. If the user
    hasn't set up a local identity yet, the function returns the `kOCESetupRequired`
    result code. If the local identity is locked, the function returns the
    `kOCELocalAuthenticationFail` result code. In either case, call the
    `AuthAddToLocalIdentityQueue` function to be notified when the local identity
    is set up and unlocked. If the `AuthGetLocalIdentity` function returned
    `kOCELocalAuthenticationFail`, you can pass the locked local identity provided
    by the function to the `DirLookupGet` and `DirLookupParse` functions. Therefore,
    you should proceed with the initialization process.

3. Get the reference number of the Setup catalog and the creation ID of the Setup
    record by calling the `DirGetOCESetupRefnum` function. You need to provide the
    catalog's reference number in the `dsRefNum` field of the `DirLookupGet` and
    `DirLookupParse` parameter blocks when you want to read the records in the
    Setup catalog. You need the creation ID to build a record ID for the Setup record.

4. Get the current location from the Setup record in the Setup catalog by calling the
    `DirLookupGet` and `DirLookupParse` functions. As the target of the `aRecordList`
    field in the `DirLookupGet` parameter block, specify the record ID of the Setup record.
    You can set all fields of the record ID except the creation ID to `nil`. Set the creation ID

to the value you obtained in the previous step. Instead of providing record location information, you provide the catalog's reference number in the `dsRefNum` field of the `DirLookupGet` function's parameter block. As the target of the `attrTypeList` field in the parameter block, specify the `AttributeType` structure referenced by the attribute type index `kLocationAttrTypeNum`. The function reads the Setup record and places the location information into a buffer in a private data format.

Call the `DirLookupParse` function to read the data in the buffer. The function calls a callback routine that you provide and passes it a pointer to an `Attribute` structure containing the location information (type `OCESetupLocation`) that you requested.

5. Get a reference to each Mail Service or Combined record that belongs to the MSAM by calling the `DirLookupGet` and `DirLookupParse` functions. If you provide a stand-alone MSAM, attributes for a slot and its associated catalog are stored in a Mail Service and a Catalog record, respectively. If you provide a combined MSAM/CSAM, attributes for a slot and its associated catalog are stored in a single Combined record.

As the target of the `aRecordList` field in the `DirLookupGet` parameter block, specify the `RecordID` structure that you created that contains the creation ID of your MSAM record. As the target of the `attrTypeList` field in the parameter block, specify the `AttributeType` structure referenced by the attribute type index `kMailServiceAttrTypeNum`. The function reads the MSAM record and places the packed record ID of each Mail Service or Combined record that it finds into a buffer in a private data format.

Call the `DirLookupParse` function to read the data in the buffer. The function calls your callback routine and passes it a pointer to an `Attribute` structure containing a packed record ID that points to either a Mail Service or a Combined record. The `DirLookupParse` function calls your callback routine once for each packed record ID in the buffer, each of which corresponds to a slot for which your MSAM is responsible. Now you know how many slots you are responsible for and in what records their specific information is stored.

6. Unpack the packed record IDs of the Mail Service or Combined records by calling the `OCEUnpackRecordID` utility function.

7. Get the slot ID, standard slot information, and associated catalog information for each slot by calling the `DirLookupGet` and `DirLookupParse` functions. As the target of the `aRecordList` field in the `DirLookupGet` parameter block, specify the unpacked record IDs that point to your Mail Service or Combined records. As the target of the `attrTypeList` field in the parameter block, specify `AttributeType` structures that are referenced by the following attribute type indexes: `kSlotIDAttrTypeNum`, `kStdSlotInfoAttrTypeNum`, and `kAssoDirectoryAttrTypeNum`.

Call the `DirLookupParse` function. It repeatedly calls your callback routine and passes it a pointer to an `Attribute` structure containing one of the record attributes you requested for each of your Mail Service or Combined records.

The value of each `kSlotIDAttrTypeNum` attribute is the slot ID you previously assigned to the slot while processing the `kMailEPPCCreateSlot` high-level event for that slot. It is a number (type `MailSlotID`) that uniquely identifies the slot. (If you have never received and processed a `kMailEPPCCreateSlot` high-level event, no `kSlotIDAttrTypeNum` attributes exist.)

The value of each `kStdSlotInfoAttrTypeNum` attribute is a
`MailStandardSlotInfoAttribute` structure that indicates if the slot is active
and provides its send and receive timer information. For each slot, you must
determine if the slot is active at the current system location. The `active` field of the
`MailStandardSlotInfoAttribute` structure is a bit array; each bit corresponds to
a possible system location. If the slot is active at that location, the bit is set. You can
test the bits with the `MailLocationMask` macro (see page 2-115)

The value of each `kAssoDirectoryAttrTypeNum` attribute is a packed record ID
that points to the Catalog record associated with this slot or to the Combined record.

8. If you provide a stand-alone MSAM, unpack the packed record ID for each slot's
   associated Catalog record by calling the `OCEUnpackRecordID` utility function. (If
   you provide a combined MSAM/CSAM, attributes for the slot and catalog are both
   stored in the Combined record— you already unpacked the Combined record IDs.)

9. Get information about the catalog associated with each slot by calling the
   `DirLookupGet` and `DirLookupParse` functions. As the target of the `aRecordList`
   field in the `DirLookupGet` parameter block, specify the unpacked record IDs that
   point to your Catalog or Combined records. As the target of the `attrTypeList`
   field in the parameter block, specify `AttributeType` structures that are
   referenced by the following attribute type indexes: `kCommentAttrTypeNum`,
   `kRealNameAttrTypeNum`, and `kDiscriminatorAttrTypeNum`. If you provide a
   combined MSAM/CSAM, also specify `kSFlagsAttrTypeNum`.

   Call the `DirLookupParse` function. It repeatedly calls your callback routine and
   passes it a pointer to an `Attribute` structure containing one of the attributes you
   requested from each Catalog or Combined record. Table 2-8 on page 2-40 describes
   the information contained in those attributes.

10. Get the user's account name and decrypted password by calling the
    `OCESetupGetDirectoryInfo` function. If the local identity is still locked, this
    function returns an error. You cannot proceed until the local identity is unlocked.

    Note that the value of the `nativeName` field returned by the
    `OCESetupGetDirectoryInfo` function is the value of the Real Name attribute
    (`kRealNameAttrTypeNum`) in the Catalog or Combined record. The content and use
    of the Real Name attribute and the `nativeName` field are defined by the personal
    MSAM and its setup template. A setup template can store the user's account name
    in the Real Name attribute.

    At this point, you have obtained all of the standard information stored in your MSAM
    and Combined records (or MSAM, Mail Service, and Catalog records) in the Setup
    catalog. Using the `DirLookupGet` and `DirLookupParse` functions, you may read
    other attributes of private types that your setup or address template has added to
    the records.

11. Get the incoming and outgoing queue references for each of the slots by calling the
    `PMSAMOpenQueues` function for each slot.

Now the personal MSAM can begin performing its primary functions of translating and transferring messages between an AOCE system and external messaging systems.

Table 2-8   Selected Catalog record attributes

Attribute type	Data type of attribute value	Description
kDiscriminatorAttrTypeNum	DirDiscriminator	Discriminator value for this catalog.
kSFlagsAttrTypeNum	long	Bit array indicating the features supported by this catalog. Present for combined MSAM/CSAM only.
kCommentAttrTypeNum	RString	Displayable string describing this catalog/external messaging system.
kRealNameAttrTypeNum	RString	Defined by the MSAM and its setup template. For example, it may be the user's account (logon) name or the name of the external messaging system and its address catalog.

The chapter "Service Access Module Setup" in this book describes the information that your setup template obtains from the user and stores in the Setup catalog as well as the process it uses to do so. See the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* for descriptions of the DirGetOCESetupRefnum, DirLookupGet, and DirLookupParse functions. For a description of the OCEUnpackRecordID function and the record and attribute type indexes, see the chapter "AOCE Utilities" in *Inside Macintosh: AOCE Application Interfaces*. The OCESetupGetDirectoryInfo function is described in the chapter "Authentication Manager" in *Inside Macintosh: AOCE Application Interfaces*.

Initializing a Server MSAM

The first time a server MSAM is launched, it needs to solicit user input to obtain information about itself. Then it initializes itself within the AOCE system by calling the SMSAMSetup and SMSAMStartup functions.

The SMSAMSetup function creates the server MSAM's Forwarder record. The **Forwarder record** (record type index kMnMForwarderRecTypeNum) contains information about the server MSAM. The Forwarder record name is the name of the server MSAM. The record contains the record ID of the MSAM's PowerShare mail server, an optional comment string describing the server MSAM, and a list of the foreign dNodes to which the server MSAM is connected. (See the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* for information about PowerShare catalogs, dNodes, and foreign dNodes, as well as other concepts that pertain to AOCE catalogs.)

After being launched for the first time, a server MSAM must find out its name, password, messaging system extension type, and a descriptive comment string about the extension type. The MSAM should display one or more dialog boxes to obtain its name and password from the system administrator. Typically, an MSAM has built-in knowledge of the extension type it supports and a descriptive comment string about the extension type; if it does not, it must obtain that information from the system administrator.

Once a server MSAM has all this information, it calls the SMSAMSetup function to create its Forwarder record. Prior to calling the function, the MSAM must allocate a RecordID structure for its Forwarder record. Then the MSAM sets the recordName field to its name that the user provided, and the recordType field to the constant kMnMForwarderRecTypeNum. The MSAM passes to the function a pointer to the RecordID structure, the MSAM's password, its extension type, and a string describing its extension type. In the RecordID structure, the function returns the creation ID for the newly created Forwarder record and the record location information. In the catalogServerHint field, the function returns the AppleTalk address (an AddrBlock structure) of the PowerShare catalog server that created the Forwarder record. The MSAM can pass this address to a Catalog Manager function (in the serverHint field of the function's parameter block) if it wants to direct the request to that particular catalog server. This can be helpful in preventing failures in the setup process due to delays in replicating the MSAM's Forwarder record.

During the execution of the SMSAMSetup function, the PowerShare mail server prompts the user for the system administrator's name and password. You may find it helpful to consult the *PowerShare System Manager's Guide*, which describes the setup process from the system administrator's perspective.

If the system administrator does not provide this information, the function returns an error. The function will also return an error if

n   the PowerShare catalog server was unreachable

n   the MSAM's name is not unique

n   the disk is full

n   an error occurred in creating the Forwarder record (any record creation error)

If an error occurs, the MSAM must display an appropriate dialog box telling the user about the error. If the PowerShare catalog server was unreachable, the MSAM should give the user the option of trying the operation again and, if the user chooses to try again, the MSAM should call the SMSAMSetup function once more. If the user chooses not to try again, the MSAM should quit. If the MSAM's name was not unique, the MSAM should allow the user to enter another name. In any error, the MSAM should fix the problem when it can or quit when it cannot. Until the SMSAMSetup function executes successfully, the MSAM cannot proceed with its initialization process.

When the SMSAMSetup function completes successfully, the server MSAM must save knowledge of this fact so that if it is launched again in the future, it does not call the SMSAMSetup function again. It is recommended that the server MSAM create a preferences file in the Preferences folder and save the record ID of its Forwarder record in its preferences file.

Once the `SMSAMSetup` function completes successfully, the server MSAM should call the `AuthBindSpecificIdentity` function, providing the record ID of its Forwarder record and its encrypted password, to obtain its authentication identity. Once a server MSAM has obtained its authentication identity, it should provide that information on subsequent calls to AOCE functions that require an identity.

At this point, the server MSAM may present dialog boxes to the user to obtain any additional configuration information it needs to function within an AOCE system and to connect to its external messaging system, such as an IP address, a telephone number, how often it should connect, and so forth. In general, an MSAM should ask for more generic information first—that is, information that applies independently of a messaging system. Then it should prompt for specific information for each messaging system that it supports. It should then add this information to its Forwarder record in MSAM-defined attribute types.

Note

In addition to its Forwarder record, a server MSAM should store a copy of its configuration information in its preferences file for quick, efficient access.

A server MSAM should keep a backup copy of its preferences file in case the file is lost or damaged. If its preferences file is lost or damaged and a server MSAM does not have a backup copy, it can retrieve the information stored in the MSAM's Forwarder record and rebuild the file. To read its Forwarder record, an MSAM must have the Forwarder record ID (which it obtains from the `SMSAMSetup` function). u

As the final step in the server MSAM's initialization process, the MSAM calls the `SMSAMStartup` function to obtain a reference number for its outgoing queue. After the `SMSAMStartup` function completes successfully, the PowerShare mail server may send high-level events to the server MSAM. The MSAM should respond to high-level events, connect to external messaging systems, and begin to translate and transfer messages.

A server MSAM must run on the same Macintosh computer as its PowerShare mail server. If the PowerShare mail server is not running, the `SMSAMStartup` function returns the `corErr` result code. You can detect when the PowerShare mail server becomes available by

n   repeatedly calling the `Gestalt` function and using the `gestaltOCESFServerAvailable` mask on its `response` parameter to determine if a PowerShare mail server is running on the local Macintosh computer

n   repeatedly calling the `SMSAMStartup` function

n   adding an entry to the AppleTalk Transition Queue and waiting to receive a notification that the PowerShare mail server is available

Using the AppleTalk Transition Queue is the recommended approach. The transition event code `ATTransSFStart` indicates that the PowerShare mail server has finished starting up, and the code `ATTransSFShutdown` indicates that the PowerShare mail server has started to shut down.

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

If the PowerShare mail server quits, your queue reference becomes invalid. You know that the PowerShare mail server is not running when any of the MSAM API functions return the `corErr` result code or you receive notification of the `ATTransSFShutdown` AppleTalk transition event. If the PowerShare mail server quits unexpectedly, you do not receive an AppleTalk transition event.

When it starts up again, the PowerShare mail server does not know that your server MSAM exists. You need to call the `SMSAMStartup` function again to get a new queue reference. You detect the restarting of the PowerShare mail server by any of the three methods listed previously.

If the PowerShare mail server quits, your server MSAM can keep running. Although you can no longer retrieve messages from your outgoing queue, you can continue to process any outgoing messages you queued separately. You can mark recipients and send reports for those messages after the PowerShare mail server resumes operations. If you have a separate spool area to hold them, you can continue to process incoming messages while the PowerShare mail server is not running.

Handling Outgoing Messages

This section describes what you need to do with messages in an outgoing queue. It assumes you have already initialized your MSAM. Each subsection addresses a specific task, such as

- enumerating messages in an outgoing queue

- opening and closing messages

- determining the message family

- determining what is in a message

- reading letter attributes

- reading addresses

- reading letter content

- reading nested messages

- marking recipients

- generating reports

There are some differences between how you read letters and how you read non-letter messages. These differences are noted in the sections that address the specific tasks. For convenience, Table 2-9 lists the tasks you perform while handling messages in an outgoing queue and the functions you use to accomplish the task for a letter and a non-letter message.

Table 2-9   Outgoing tasks and functions

Task	Letters	Non-letter messages
Enumerate a queue	`MSAMEnumerate`	`MSAMEnumerate`
Open a message	`MSAMOpen` `MSAMOpenNested`	`MSAMOpen` `MSAMOpenNested`
Read header information	`MSAMGetAttributes` `MSAMGetRecipients`	`MSAMGetMsgHeader` `MSAMGetRecipients`
Read letter content	`MSAMGetContent`	Not applicable
Read an enclosure	`MSAMGetEnclosure`	Not applicable
Enumerate a block	`MSAMEnumerateBlocks`	`MSAMEnumerateBlocks`
Read a block	`MSAMGetBlock`	`MSAMGetBlock`
Close a message	`MSAMClose`	`MSAMClose`

Generate a report	`MSAMCreateReport`
	`MSAMPutRecipientReport`
	`MSAMSubmit`
		`MSAMCreateReport`
		`MSAMPutRecipientReport`
		`MSAMSubmit`

Mark a recipient	`MSAMnMarkRecipients`	`MSAMnMarkRecipients`
Set message status (personal MSAMs only)	`PMSAMSetStatus`	`PMSAMSetStatus`

The order in which functions are listed in Table 2-9 corresponds to the sequence in which you would call the functions to process a message in an outgoing queue. You first enumerate the messages in the queue. Then you open a specific message and read its header information. Header information consists of such items as the message creator and type, and address (recipient) information. Next, you read the substance of the message—for a letter, its content block, other blocks it may contain, and enclosures; for a non-letter message, its blocks. When you have finished reading the message, you close it. After you have transmitted the message to the recipients for which you are responsible, you indicate the outcome of your delivery attempts—that is, you generate a report containing delivery and non-delivery indications if required and mark the recipients. Setting the status of a message is a task that you perform at several points while you are processing the message.

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

Enumerating Messages in an Outgoing Queue

Before you can read a message from an outgoing queue, you must obtain its sequence number. A sequence number uniquely identifies the message in the queue. You provide it when you open the message. You get the sequence number of a message by calling the `MSAMEnumerate` function.

To make sure it transfers outgoing messages in a timely manner, an MSAM should enumerate an outgoing queue on a regular basis. The MSAM should enumerate each

time it is launched and each time it receives a `kMailEPPCMsgPending` event. It should also enumerate at periodic intervals—for instance, every 20 minutes. If an MSAM puts itself into an idle state, it should enumerate before entering the idle state. A personal MSAM should also enumerate when it receives a `kMailEPPCSchedule` event.

Listing 2-1 illustrates one way that you can enumerate messages in an outgoing queue. For convenience, the function `DoEnumerateOutgoingMessages` in Listing 2-1 defines the type `MyEnumOutQReplyType`, a structure that contains a buffer that can hold a 2-byte count value plus exactly one `MSAMEnumerateOutQReply` structure. As a result, each time `DoEnumerateOutgoingMessages` calls the `MSAMEnumerate` function, `MSAMEnumerate` returns exactly one `MSAMEnumerateOutQReply` structure, which provides identifying information about one message in the queue, including its sequence number.

Before the `DoEnumerateOutgoingMessages` function calls the `MSAMEnumerate` function, it always initializes the fields of the parameter block. It sets the queue reference to an outgoing queue reference previously obtained from the `PMSAMOpenQueues` function. The first time through the loop, `DoEnumerateOutgoingMessages` sets the starting sequence number to 1 to start with the first message in the queue. On subsequent executions of the loop, it sets the starting sequence number to the sequence number of the next message in the queue, which is returned by `MSAMEnumerate`.

The `DoEnumerateOutgoingMessages` function calls the `MSAMEnumerate` function once for each message in the queue. Into your buffer, `MSAMEnumerate` places the count of the number of `MSAMEnumerateOutQReply` structures followed by the reply structures themselves. In Listing 2-1, the count is always 1.

Listing 2-1   Enumerating outgoing messages

```
OSErr DoEnumerateOutgoingMessages(MSAMQueueRef myOutgoingQRef)
{
typedef struct MyEnumOutQReplyType {
   MailReply             reply;   /* number of structures returned */
   MSAMEnumerateOutQReply  message; /* enumerate reply structure */
} MyEnumOutQReplyType;

   OSErr              myErr;
   MSAMEnumeratePB      myParamBlock;
   MyEnumOutQReplyType  myEnumOutQReply;
   long               myNextMsgSeq;

   myNextMsgSeq   = 1;
   myErr          = noErr;
```

```
do {
    myParamBlock.ioCompletion          = (ProcPtr)DoMSAMCompletion;
    myParamBlock.queueRef              = myOutgoingQRef;
    myParamBlock.startSeqNum           = myNextMsgSeq;
    myParamBlock.buffer.bufferSize     = sizeof(MyEnumOutQReplyType);
    myParamBlock.buffer.buffer         = (Ptr)&myEnumOutQReply;

    MSAMEnumerate((MSAMParam *)&myParamBlock,true);
    /* poll for completion */
    myErr          = DoWaitPBDone(&myParamBlock);
    myNextMsgSeq   = myParamBlock.nextSeqNum;

    /* save the MSAMEnumerateOutQReply structure */
    DoSaveData((Ptr)&myEnumOutQReply);

}
while (myErr == noErr && myNextMsgSeq != 0);

return myErr;
}
```

The `DoWaitPBDone` function, called here and in the listings in the following sections, polls the `ioResult` field to determine when an asynchronous request has completed. While it is polling, it also yields time to other processes running on the computer by calling the `WaitNextEvent` function. When the `MSAMEnumerate` function completes, `DoWaitPBDone` returns the `MSAMEnumerate` result code as its result code.

The `DoMSAMCompletion` completion routine, called when the `MSAMEnumerate` function completes execution, calls the `WakeUpProcess` function. Then `WakeUpProcess` makes the MSAM process, which suspended itself by calling the `WaitNextEvent` function, eligible to receive CPU time.

After the `MSAMEnumerate` function completes, `DoEnumerateOutgoingMessages` saves the enumeration information elsewhere by calling its `DoSaveData` function. It needs to do this because `MSAMEnumerate` overwrites the `MyEnumOutQReplyType` structure each time through the loop.

Opening and Closing a Message

Before you can read any part of an outgoing message, you must open it. To open a specific message, you call the `MSAMOpen` function and provide the queue reference of the outgoing queue in which the message is located and the sequence number of the message. The `MSAMOpen` function returns a reference number for the opened message that you use when you call other functions to read the various parts of the message, such as the message header, recipient information, and the content data in the message. If the message is a letter, you can also read the letter's attributes. You cannot modify a message in an outgoing queue.

When you have finished reading a message, call the `MSAMClose` function to close it. Closing a message reduces PowerTalk software memory requirements. Once you have closed a message, the message reference number is no longer valid, even though the message itself remains in the outgoing queue. If you want to read any part of the message again, you must call the `MSAMOpen` function and get a new reference number. You can open and close a message as many times as you wish.

Determining the Message Family

You must determine if a message that you want to read is a letter or a non-letter message because the functions you use to read a letter or a non-letter message differ somewhat (see Table 2-9 on page 2-44). You determine the message family to which a message belongs by examining the `msgFamily` field in the `MSAMEnumerateOutQReply` structure. Letters may belong to either the `kMailFamily` or `kMailFamilyFile` family. Non-letter messages belong to the `kIPMFamilyUnspecified` family. Once you know the message family, you can call the appropriate MSAM functions to read the attributes, addresses, and contents of the letter or non-letter message.

Determining What Is in a Message

Typically, when you read a letter, you call the `MSAMGetContent`, `MSAMGetBlock`, `MSAMGetEnclosure`, and `MSAMOpenNested` functions to read the letter's content block, image block, enclosures, and nested letter, respectively.

When you want to read a non-letter message, you need to enumerate the blocks in the message. The `MSAMEnumerateBlocks` function returns each block's creator and type, its offset in bytes from the beginning of the message, and its length in bytes. When you want to read a given block, you call the `MSAMGetBlock` function and provide the block's creator and type.

Reading Letter Attributes

Every letter contains attributes that provide information about the letter, such as whether the sender wants to receive a report containing delivery or non-delivery indications, when the letter was sent, and so forth. You should read this information and include in the letter as much of the information as is meaningful in your messaging system. You can read most letter attributes with the `MSAMGetAttributes` function. However, to read the recipients of a letter—the `from`, `to`, `cc`, and `bcc` attributes—you call the `MSAMGetRecipients` function.

To the `MSAMGetAttributes` function, you provide a set of bit flags, known as the *request mask*, that represents the attributes whose values you want to read and a buffer to hold the attribute values. The `MailAttributeBitmap` structure, described on page 2-100, defines the attributes that the bit flags in the request mask represent. The function returns a second set of bit flags, known as the *response mask*, that indicates which of the requested attribute values it has returned in your buffer.

The function `DoReadLetterAttributes` in Listing 2-2 shows how you can request attribute values, test for their presence in your buffer, and save the value in a file. The `DoReadLetterAttributes` function defines the structure type `MaximumLetterAttributes` that is large enough to hold a value for each of the attributes that the `MSAMGetAttributes` function can return. The `DoReadLetterAttributes` function declares a variable of that type, `myAttribBuf`, and sets a pointer, `myAttribPtr`, to point to the start of the buffer. Next, it initializes the request mask to 0 and then sets the request mask to specify every attribute that the `MSAMGetAttributes` function can return. If the messaging system to which you provide access does not use some of this information, don't ask for it. For instance, if you know that your messaging system does not understand a reply ID, do not set the bit for the reply ID in the attribute request mask.

Note

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

After the `DoReadLetterAttributes` function sets its attribute request mask, it calls the `MSAMGetAttributes` function. The `MSAMGetAttributes` function returns the attributes that you request (if they are present in the letter header) packed into your buffer, starting with the attribute specified by the least significant bit in the request mask. The `MSAMGetAttributes` function also sets the bits in the response mask corresponding to those attributes for which it returned a value.

Next, `DoReadLetterAttributes` tests the bits in the response mask to find out which attributes are in the buffer. Initially, `myAttribPtr` points to the beginning of the `myAttribBuf` buffer. For each bit in the response mask that is set, `DoReadLetterAttributes` writes the corresponding attribute value to a file and adds the size of the attribute value's data type to `myAttribPtr` to position the pointer to the start of the next attribute value in `myAttribBuf`.

Listing 2-2   Reading letter attributes

```
OSErr DoReadLetterAttributes(MailMsgRef myMailRef)
{
 /* maximum size structure for calling MSAMGetAttributes */
typedef struct MaximumLetterAttributes {
    MailIndications     indications;
    OCECreatorType      msgType;
    MailLetterID        letterID;
    MailTime            sendTimeStamp;
    MailNestingLevel    nestingLevel;
    OSType              messageFamily;
    MailLetterID        replyID;
```

```
   MailLetterID       conversationID;
   RString            subject;
} MaximumLetterAttributes;

   OSErr                   myErr;
   MSAMGetAttributesPB     myParamBlock;
   MailAttributeBitmap     myRequestBitmap;
   MaximumLetterAttributes myAttribBuf;
   char                    *myAttribPtr;
   long                    *myClearBitmap;


   myAttribPtr = (char *)&myAttribBuf;    /* point to start of buffer */

   /* initialize the request mask to 0 */
   myClearBitmap  = (long *)&myRequestBitmap;
   *myClearBitmap = 0L;

   /* set bits for the attributes you want */
   myRequestBitmap.indications = myRequestBitmap.msgType =
       myRequestBitmap.letterID = myRequestBitmap.sendTimeStamp =
       myRequestBitmap.nestingLevel = myRequestBitmap.msgFamily =
       myRequestBitmap.replyID = myRequestBitmap.conversationID =
       myRequestBitmap.subject = 1;

   /* fill in the fields of the parameter block */
   myParamBlock.ioCompletion    = (ProcPtr)DoMSAMCompletion;
   myParamBlock.mailMsgRef       = myMailRef;
   myParamBlock.requestMask      = myRequestBitmap;
   myParamBlock.buffer.bufferSize = sizeof(MaximumLetterAttributes);
   myParamBlock.buffer.buffer    = myAttribPtr;
   myParamBlock.more             = false;

   /* call function to get the attributes */
   MSAMGetAttributes((MSAMParam *)&myParamBlock,true);
   myErr = DoWaitPBDone(&myParamBlock);
   if (myErr!=noErr)
      return myErr;

   /* save returned attributes to disk */
   if (myParamBlock.responseMask.indications) {
      myErr = DoWriteToFile(kMailIndicationsMask, myAttribPtr,
                        sizeof(MailIndications));
```

```
   if (myErr!=noErr)
      return myErr;
   myAttribPtr += sizeof(MailIndications);
}

if (myParamBlock.responseMask.msgType) {
   myErr = DoWriteToFile(kMailMsgTypeMask, myAttribPtr,
                         sizeof(OCECreatorType));
   if (myErr!=noErr)
      return myErr;
   myAttribPtr += sizeof(OCECreatorType);
}

if (myParamBlock.responseMask.letterID) {
   myErr = DoWriteToFile(kMailLetterIDMask, myAttribPtr,
                         sizeof(MailLetterID));
   if (myErr!=noErr)
      return myErr;
   myAttribPtr += sizeof(MailLetterID);
}

/*
   Test for presence of the send time stamp, nesting level, message
   family, reply ID, and conversation ID attributes. If present, write
   them to file.
*/

if (myParamBlock.responseMask.subject) {
   myErr = DoWriteToFile(kMailSubjectMask, myAttribPtr, sizeof(RString));
   if (myErr!=noErr)
      return myErr;
   myAttribPtr += sizeof(RString);
}
}
```

You can read information such as the message creator and message type from the message header of non-letter messages by calling the `MSAMGetMsgHeader` function.

Interpreting Creator and Type for Messages and Blocks

An outgoing message may have any message creator and any message type. Typically, an application that generates a message uses its own application signature as the message creator and its document type as the message type.

The message creator value `'lap2'` indicates that the AppleMail application created the message.

If the message type of an outgoing message is `kMailLtrMsgType`, the message is a letter that contains any or all of the following: data in standard interchange format, data in image format, or a regular enclosure.

Each block in an outgoing message has a block creator and block type. The AppleMail application sets the block creator to `kMailAppleMailCreator` for blocks that it creates. The block types that you may find in a letter are listed in Table 2-3 on page 2-18.

Reading Addresses

When you read the addresses associated with an outgoing message, you must get both the original and the resolved recipients for that message. That gives you complete addressing information for both display and routing purposes.

An **original recipient** can be a To, From, cc, or bcc recipient. These four types of original recipients are defined as follows:

n   From: the sender of a message

n   To: a primary recipient of a message

n   cc: a secondary recipient receiving a copy of a letter

n   bcc: a secondary recipient whose address does not appear on the letter as received by the To and cc recipients and other bcc recipients

An original recipient may be a group address (distribution list).

A **resolved recipient** is a recipient to which you are responsible for delivering the message. Usually, a resolved recipient is an individual address; sometimes it can be a group address.

Reading Original Recipients

To get a list of original recipients, you call the `MSAMGetRecipients` function. You need to get original recipients so that you can properly display them as From, To, cc, or bcc recipients in the message you send to an external messaging system. The function returns information about one type of original recipient. You specify the type of original recipient you want by setting the `attrID` field of the `MSAMGetRecipientsPB` parameter block appropriately. You can set the `attrID` field to any of the following constants:

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

If you are reading a letter, you need to get each original recipient type so that when you translate the letter, it includes display information about all of the recipients. Display address information refers to an address that may not be usable for routing within a given messaging system but nevertheless shows that the letter went to the addressee.

(A bcc recipient is an exception, as it should be displayed only to the sender and the bcc recipient itself.)

If you are reading a non-letter message, the only original recipient types that apply are From and To. You may not need to get display information. If that is the case, do not call the `MSAMGetRecipients` function to retrieve the To recipients. You may still want to call it to get the From recipient. (You could also get the From recipient by calling the `MSAMGetMsgHeader` function.)

When a letter has a bcc recipient, you must make every attempt to conform to the following AOCE guidelines for bcc recipients: A bcc recipient must know that he or she is a bcc recipient. A To or a cc recipient must not see any bcc recipient. It is less desirable, but acceptable, for a bcc recipient to see other bcc recipients.

To support these guidelines, your MSAM may need to generate a separate copy of the letter for each bcc recipient for which it is responsible or employ other implementations that are less straightforward or more expensive than usual. As a last resort, if your MSAM cannot support AOCE guidelines, it must reject bcc recipients. In that case, it must still apply the guidelines to the letter—that is, no other recipient must know of the bcc recipients.

Reading Resolved Recipients

To get a list of resolved recipients, call the `MSAMGetRecipients` function and specify the `kMailResolvedList` constant in the `attrID` field of the `MSAMGetRecipientsPB` parameter block. You need to get a list of resolved recipients so that you know to which recipients you must send the message.

As you read the `MailResolvedRecipient` structures that the `MSAMGetRecipients` function places in your buffer, you must save the ordinal-position value for each resolved recipient. The first recipient's ordinal-position value is 1; the second recipient's ordinal-position value is 2; and so forth. The `MSAMnMarkRecipients` function requires you to provide the ordinal-position value to identify a recipient for whom you have completed delivery attempts. If you need to call `MSAMGetRecipients` more than once to get all of the resolved recipients, you must increment the ordinal-position value continuously so that each resolved recipient is associated with a unique ordinal-position value.

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

Server MSAMs may find more recipients in the resolved list than in the displayable lists for this reason: the PowerShare mail server expands PowerShare group addresses into individual addresses for the resolved list, but the original recipient lists may have included PowerShare group addresses that were not expanded. The `MSAMGetRecipients` function does not expand external group addresses.

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

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

Listing 2-3 illustrates a dispatch routine that calls the DoReadGenericAddress function (shown in Listing 2-4 on page 2-55) to get a list of resolved recipients and lists of the original recipients that are appropriate to a letter or a non-letter message.

Listing 2-3   Getting resolved and original recipients

```
OSErr DoReadAddress(MailMsgRef myMsgRef)
{
   FSSpec    myTempFileSpec;
   OSErr     myErr;

   /* initialize the file specification */

   myErr = DoReadGenericAddress(&myTempFileSpec, myMsgRef,
                                kMailResolvedList);
   if (myErr!= noErr)
      return myErr;

   myErr = DoReadGenericAddress(&myTempFileSpec, myMsgRef, kMailFromBit);
   if (myErr!= noErr)
      return myErr;

   myErr = DoReadGenericAddress(&myTempFileSpec, myMsgRef, kMailToBit);
   if (myErr!= noErr)
      return myErr;

   if (myMsg->msgFamily == kMailFamily) { /* it's a letter */
      myErr = DoReadGenericAddress(&myTempFileSpec, myMsgRef, kMailCcBit);
      if (myErr!= noErr)
         return myErr;

      myErr = DoReadGenericAddress(&myTempFileSpec, myMsgRef, kMailBccBit);
      if (myErr!= noErr)
         return myErr;
   }

   return myErr;
}
```

The function `DoReadGenericAddress` shown in Listing 2-4 actually reads the addresses from an outgoing message and writes them to a disk file. The `DoReadGenericAddress` function takes three parameters: the file system specification of a temporary disk file to which it writes the addresses, the message reference number for a given message, and an attribute ID that identifies the type of address that the caller wants to retrieve from the message.

First `DoReadGenericAddress` allocates a buffer, pointed to by the `addressBuffer` field, that it uses to hold addresses returned by the `MSAMGetRecipients` function. It sets the size of the buffer to 1024 bytes. Your MSAM should determine the buffer size that is appropriate for your needs.

Next, `DoReadGenericAddress` determines if it is handling a request to get resolved or original recipients and sets the `doingResolved` Boolean variable accordingly. If it is handling resolved recipients, `DoReadGenericAddress` initializes its local variable `ordinalPosition` to 0. It uses `ordinalPosition` to save the ordinal position of each resolved recipient. It needs this information to mark a recipient when it has finished its efforts to deliver the letter to the recipient. The ordinal-position value must be unique for each recipient.

Then, `DoReadGenericAddress` fills in all but one of the fields of the local variable `myParamBlock`, which is an `MSAMGetRecipientsPB` parameter block. It sets the `myParamBlock.mailMsgRef` field to its message reference number parameter (`myMailRef`) to identify the message and sets the `myParamBlock.attrID` field to its attribute ID parameter (`attrID`) to indicate which type of address (To, From, cc, bcc, or resolved) it wants the `MSAMGetRecipients` function to return. Although the `nextIndex` and `more` fields are outputs of the `MSAMGetRecipients` function, `DoReadGenericAddress` sets them here to execute the `for` statement that follows and to initialize the `myParamBlock.startIndex` field properly the first time through the loop.

To accomplish its main work, `DoReadGenericAddress` uses two `for` loops, one nested inside the other. Note that the outer `for` statement contains only the logical expression controlling the iteration of the loop. The loop executes as long as the value of `myParamBlock.more` is `true` and no error has occurred. The `MSAMGetRecipients` function sets the `more` field to `true` when there are more addresses to return than it could fit into the caller's buffer.

The outer `for` loop sets the `myParamBlock.startIndex` field to the value of the `myParamBlock.nextIndex` field, which it previously set to 1. This tells the `MSAMGetRecipients` function that it should begin returning addresses starting with the first address of the specified type. Then `DoReadGenericAddress` calls `MSAMGetRecipients` asynchronously and polls for its completion.

If no error has occurred, `DoReadGenericAddress` initializes two variables used by the inner `for` loop. The `MSAMGetRecipients` function always puts at the beginning of your buffer the count of the number of addresses it placed in your buffer, followed by the addresses themselves. Therefore, `DoReadGenericAddress` sets `recipientPtr` to point into the address buffer at the byte where address information actually begins, skipping over the count. It next sets the variable `numRecipients` to the count of the

number of addresses in the buffer. Then, it executes the inner `for` loop to manipulate the addresses returned in the buffer.

The inner `for` loop extracts an address from the buffer and writes it to a disk file. It executes until all of the addresses have been extracted and written or until an error occurs. For convenience, `DoReadGenericAddress` defines two new types, `MailOriginalRecipientExt` and `MailResolvedRecipientExt`. Each consists of a `MailOriginalRecipient` or `MailResolvedRecipient` structure, respectively, followed by an `OCEPackedRecipient` structure. The new types enable `DoReadGenericAddress` to manipulate all of the relevant information associated with a particular address using a single structure.

If it is extracting resolved recipients, `DoReadGenericAddress` first increments the `ordinalPosition` local variable. Then it sets the pointer `resolvedPtr` to `recipientPtr`, which in turn points to the beginning of the first resolved address. The `DoReadGenericAddress` function writes the `MailResolvedRecipientExt` structure to a disk file, tagging it with its address type (attribute ID) and ordinal-position value for later identification. Once that is done, `DoReadGenericAddress` advances the `recipientPtr` pointer to the next address in the buffer. It moves `recipientPtr` past the `MailResolvedRecipient` structure, past the `dataLength` field in the `OCEPackedRecipient` structure, and then past the number of bytes specified in the `dataLength` field. If `recipientPtr` points to an odd byte address, `DoReadGenericAddress` increments it by 1 to point to an even byte boundary. At this point, the `for` loop is ready to execute again.

Because of differences in the sizes of the applicable structures, the `for` loop has separate but parallel logic to extract and write resolved and original recipients.

The logic of `DoReadGenericAddress` assumes that after it writes the addresses to disk, the MSAM translates them from AOCE address format into the format of the destination messaging system.

Listing 2-4   Reading addresses from an outgoing message

```
OSErr DoReadGenericAddress(FSSpec *myTempFileSpec, MailMsgRef myMailRef,
                           MailAttributeID attrID)
{
    typedef struct MailOriginalRecipientExt {
        MailOriginalRecipient    prefix;
        OCEPackedRecipient       packedRecip;
    } MailOriginalRecipientExt;

    typedef struct MailResolvedRecipientExt {
        MailResolvedRecipient    prefix;
        OCEPackedRecipient       packedRecip;
    } MailResolvedRecipientExt;
```

```
OSErr                     myErr;
MSAMGetRecipientsPB       myParamBlock;
short                     count, numRecipients, ordinalPosition;
MailOriginalRecipientExt  *origPtr;
MailResolvedRecipientExt  *resolvedPtr;
Ptr                       addressBuffer, recipientPtr;
Boolean                   doingResolved;

addressBuffer = NewPtr(1024L);
if (MemError()!= noErr)
   return MemError();

if (attrID == kMailResolvedList) {
   doingResolved    = true;
   ordinalPosition  = 0;
else
   doingResolved = false;

myParamBlock.ioCompletion        = (ProcPtr)DoMSAMCompletion;
myParamBlock.mailMsgRef          = myMailRef;
myParamBlock.attrID              = attrID;
myParamBlock.buffer.buffer       = addressBuffer;
myParamBlock.buffer.bufferSize   = 1024L;
myParamBlock.more                = true; /* to get into "for" loop */
myParamBlock.nextIndex           = 1;

myErr = noErr;
for ( ; myParamBlock.more == true && myErr == noErr; ) {
   myParamBlock.startIndex = myParamBlock.nextIndex;
   MSAMGetRecipients((MSAMParam *)&myParamBlock,true);
   myErr = DoWaitPBDone(&myParamBlock);
   if (myErr != noErr) {
      DisposPtr(addressBuffer);
      return myErr;
   }  /* end if */
   recipientPtr  = addressBuffer + sizeof(short);
   numRecipients = (MailReply *) addressBuffer->tupleCount;
   for (count = 0; count < numRecipients && myErr == noErr;
        count++) {
      if (doingResolved) {
         resolvedPtr  = (MailResolvedRecipientExt *)recipientPtr;
         ordinalPosition++;
         myErr = WriteRecipient(myTempFileSpec, attrID, resolvedPtr,
                             ordinalPosition);
```

```
        recipientPtr += (sizeof(MailResolvedRecipient) + sizeof(short)
                        + resolvedPtr->packedRecip.dataLength);
        if ((unsigned long)recipientPtr % 2)/*pad to even boundary */
            recipientPtr++;
    }  /* end if */
    else {
        origPtr      = (MailOriginalRecipientExt *)recipientPtr;
        myErr = WriteRecipient(myTempFileSpec, attrID, origPtr, 0);
        recipientPtr += (sizeof(MailOriginalRecipient) + sizeof(short)
                        + origPtr->packedRecip.dataLength);
        if ((unsigned long)recipientPtr % 2)/*pad to even boundary */
            recipientPtr++;
    }  /* end else */
    }  /* end inner for loop */
}  /* end outer for loop */

DisposPtr(addressBuffer);
return myErr;
}
```

Reading Letter Content

You read a letter's content block by calling the MSAMGetContent function. A content block consists of a series of data segments. A segment contains data in any of these formats: plain text, styled text, pictures, sound, and QuickTime movies. You select which types of segment you want to read by setting the segmentMask field in the function's parameter block appropriately.

To read the segments sequentially, set the segmentID field to 0. The MSAMGetContent function returns data from the first segment of a type that you requested in your segment mask. Continue resetting the segmentID field to 0 on subsequent calls to the MSAMGetContent function to read the segments of interest sequentially.

To access the segments in any order you choose, set the segmentID field to a given segment's segment ID. You can obtain the segment ID for each segment in a letter's content block by scanning the segments without actually reading in any data. To do this, set the segmentMask and segmentID fields to 0 before calling the MSAMGetContent function. This tells the function that you do not want it to return data for any segment type and that you want it to return information about the segments starting with the first segment in the block. Save the values of the segmentType, segmentLength, and segmentID fields that the function returns. Reset the segmentID field to 0 and call the function again to get information about the next segment in the block. Continue saving the values of the segmentType, segmentLength, and segmentID fields, resetting the segmentID field to 0, and calling the function. The function provides information about the next segment in the content block. When it returns information about the last segment in the content block, the function returns true in the endOfContent field.

At this point, you know the order of the segments in the block, the type of data each contains, the number of bytes in the segment, and the segment IDs. You can then read the data in the segments in any order you choose. Set the segmentMask field to indicate the types of segments from which you want to retrieve data. The types of segment data you request depends on the capabilities of your messaging system. For instance, if your messaging system understands only plain text data, there is no point in reading segments that contain QuickTime movie data.

The function DoReadLetterContent in Listing 2-5 reads a letter's content block. It allocates buffer space for the segment data. In the MSAMGetContentPB parameter block, it sets the segment mask to request data from segments containing plain text, pictures, and sound. Then it repeatedly calls the MSAMGetContent function until the function returns true in the endOfContent field, always resetting the segment ID to 0 to proceed sequentially through the blocks. If MSAMGetContent completes successfully, DoReadLetterContent writes the segment data to a file. Later, it can read this file and build its message in the format acceptable to its external messaging system.

Listing 2-5   Reading a letter's content block

```
#define   kMaxBufferSize      32767L

OSErr DoReadLetterContent(FSSpec *myTempFileSpec, MailMsgRef myMailRef)
{
   MSAMGetContentPB      myParamBlock;
   Ptr                   dataBuffer;
   OSErr                 myErr;
   Boolean               startOfBlock;
   unsigned short        blockIndex;


   /* allocate data buffer */
   dataBuffer = NewPtr(kMaxBufferSize);
   if (MemError() != noErr)
      return MemError();

   /* fill in parameter block */
   myParamBlock.ioCompletion         = (ProcPtr)DoMSAMCompletion;
   myParamBlock.mailMsgRef           = myMailRef;
   myParamBlock.buffer.buffer        = dataBuffer;
   myParamBlock.buffer.bufferSize    = kMaxBufferSize;
   myParamBlock.segmentMask          = kMailTextSegmentMask |
                           kMailPictSegmentMask | kMailSoundSegmentMask;
   myParamBlock.textScrap            = nil;
```

```
/* read letter content */
startOfBlock  = true;
blockIndex    = 0;
do {
    myParamBlock.segmentID = 0;
    MSAMGetContent((MSAMParam *)&myParamBlock,true);
    myErr = WaitPBDone(&myParamBlock);
    if ((myErr == noErr) && (myParamBlock.buffer.dataSize > 0)) {
        if (startOfBlock) {
            DoWriteContentToFile(myTempFileSpec, myParamBlock.segmentType,
                                 myParamBlock.buffer.buffer,
                                 myParamBlock.buffer.dataSize, blockIndex);
            startOfBlock = false;
        }
        else
            DoAppendContentToFile(myTempFileSpec, myParamBlock.segmentType,
                                  myParamBlock.buffer.buffer
                                  myParamBlock.buffer.dataSize, blockIndex);
        if (myParamBlock.endOfSegment == true) {
            startOfBlock = true;
            blockIndex++;
        }
    }
} while ((myErr == noErr) && (myParamBlock.endOfContent == false));

DisposPtrChk(dataBuffer);

return myErr;
}
```

Reading a Nested Message

A message can have other messages nested within it. If you are reading a letter, you can determine if the letter contains nested letters by calling the MSAMGetAttributes function and requesting the nestingLevel attribute. A nesting level of 0 means there are no nested letters; a nesting level of 1 means there is one nested letter, and so forth. If you are reading a non-letter message, you can determine if it contains a nested message by calling the MSAMEnumerateBlocks function and looking for a block of type kIPMEnclosedMsgType. Such a block contains a complete message. That nested message may in turn contain a message block of type kIPMEnclosedMsgType that contains a complete message, and so on.

To open a nested message, you call the MSAMOpenNested function, which returns a reference number to the nested message. To read the nested message, you pass this nested message reference number to functions. An MSAM can call MSAMOpenNested repeatedly to open a hierarchy of nested messages.

You can close a nested message explicitly by calling the `MSAMClose` function or you can close it implicitly when you close the parent message.

Note

A letter can have only one nested letter per nesting level, although each nested letter can itself contain a nested letter, and so forth. A non-letter message may actually have more than one nested message per nesting level. The IPM Manager API allows applications to create such messages. However, the MSAM API restricts you to reading one nested message per nesting level. You can read only the first occurrence of a nested message in a sequence of message blocks. u

Marking Recipients

Once you have read a message from the outgoing queue, translated it into the format understood by your external messaging system, and transmitted it, you can mark one or more recipients. Marking a recipient indicates that you have completed your efforts to deliver the message to that recipient. You mark a recipient by calling the `MSAMnMarkRecipients` function.

Marking a recipient does not indicate that you have successfully delivered the message, but only that you are finished with your efforts to deliver it to that recipient.

You can use the `MSAMnMarkRecipients` function to help you keep track of your delivery status for a message. The function clears the `responsible` flag in the `MailResolvedRecipient` structure for the recipients you specify. Thus, if you later call the `MSAMGetRecipients` function to get the resolved recipients for the message, the `responsible` flag indicates those recipients you have already processed.

You identify a recipient that you want to mark by its ordinal position in the buffer returned by the `MSAMGetRecipients` function. That is, when you call the `MSAMGetRecipients` function to get your resolved recipients, it places recipient information in your buffer, and you must save the ordinal-position value of each resolved recipient as you retrieve the recipient information from the buffer. The first recipient's ordinal-position value is 1; the second recipient's ordinal-position value is 2; and so forth. It is this value that you provide to the `MSAMnMarkRecipients` function to identify the recipient. If you use the recipient's absolute index, contained in a `MailResolvedRecipient` structure, the `MSAMnMarkRecipients` function does not work correctly.

After you mark all of the recipients for a given message, the function sets the `done` field in the `MSAMEnumerateOutQReply` structure to `true`. If you later call the `MSAMEnumerate` function to check the messages in your outgoing queue, you can determine if you have finished processing a given message by checking the `done` field.

You can call the `MSAMnMarkRecipients` function as many times as necessary for a given message, specifying one or more recipients each time as you complete your delivery efforts for those recipients.

Generating a Report

When you have completed your delivery attempts for an outgoing message, you may need to generate a report to the sender. An MSAM determines whether it must create a report for an outgoing message by reading information in the message header. An MSAM should create a report about an outgoing message only in response to the sender's request.

If the message is a letter, an MSAM calls the `MSAMGetAttributes` function to read the `MailIndications` structure. In the `MailIndications` structure, the `kMailNonReceiptReportsBit` bit and the `kMailReceiptReportsBit` bit, if set, indicate that the letter's sender requested non-delivery and delivery indications, respectively.

If the message is not a letter, an MSAM calls the `MSAMGetMsgHeader` function with the constant `kIPMFixedInfo` as the value of the `selector` field. The `IPMFixedHdrInfo` structure returned by `MSAMGetMsgHeader` contains the `notification` field, which contains the `kIPMNonDeliveryNotificationBit` bit and the `kIPMDeliveryNotificationBit` bit. These bits, if set, indicate that the sender of the message requested non-delivery and delivery indications, respectively. Test these bits to determine if you need to create a report.

If a sender asks for delivery indications, non-delivery indications, or both, an MSAM must provide information on the outcome of delivery attempts (a delivery or non-delivery indication) for every recipient for which the MSAM is responsible. It is important that an MSAM provide delivery information on all of the MSAM's recipients whenever a sender requests any type of delivery information because an MSAM report does not go directly to the report requestor. Instead, the report goes to an AOCE agent that uses the MSAM report information to prepare an IPM report according to the requestor's specifications. If an MSAM fails to provide delivery information on all of its recipients, the requestor may receive inaccurate IPM reports.

An MSAM should ignore the bit fields having to do with including a copy of the original message in the report. If necessary, a copy of the original is added by the AOCE agent.

To create a report, an MSAM must

1. call the `MSAMCreateReport` function

2. call the `MSAMPutRecipientReport` function to add delivery and non-delivery indications for recipients for which it was responsible

3. call the `MSAMSubmit` function to deliver its finished report

An MSAM must have certain information about a message in order to create a report about the message. The `MSAMCreateReport` function requires the letter or message ID of the message to which the report applies and the address of the sender. You obtain this information from either the `MSAMGetAttributes` and `MSAMGetRecipients` functions (for a letter) or the `MSAMGetMsgHeader` function (for a non-letter message). The `MSAMPutRecipientReport` function requires the recipient index to identify which recipient is being reported upon. You obtain this information from the `MSAMGetRecipients` function.

Depending on how the external messaging system works, an MSAM may save this information in its own data store or include it with the message. If, for example, more than one MSAM connects to the same external messaging system, and the system might acknowledge receiving the message to any of those MSe s, an MSAM should include the information with the message. This enables the external messaging system to extract the information from the message and then include the information with the acknowledgment of the message. As a result, any MSAM that receives the acknowledgment has the information necessary to create a report for that message. You decide how to make sure that the information required to create a report is available, given the characteristics of the external messaging system to which your MSAM connects.

Your MSAM and its external messaging system define what constitutes successful or failed delivery for outgoing messages.

Writing Incoming Messages

This section describes how you create and submit an incoming letter for delivery to its AOCE recipients. It assumes you have already initialized your MSAM. Each subsection addresses a specific task, such as

n   creating a message summary for an incoming letter (for personal MSAMs only)

n   creating a letter

n   creating a non-letter message

n   writing letter attributes

n   writing addresses

n   writing letter content

n   submitting a letter for delivery

n   receiving a report

The differences between writing letters and writing non-letter messages are noted in the sections that address the specific tasks. For convenience, Table 2-10 lists the tasks you perform while handling incoming messages and the functions you use to accomplish each task for a letter and a non-letter message.

The order in which functions are listed in Table 2-10 corresponds to the sequence in which you would call the functions to process an incoming message. A personal MSAM first creates a message summary if it is dealing with a letter. Then all MSAMs create the message itself and begin adding information to it. First, you write header information consisting of message attributes, such as the priority of the message, and address (recipient) information. Next, you write the substance of the message—for a letter, its content block, other blocks it may contain, and enclosures; for a non-letter message, its blocks. You can include an entire message within another message by defining its beginning and end with the `MSAMBeginNested` and `MSAMEndNested` functions and

Table 2-10   Incoming tasks and functions

Task	Letters	Non-letter messages
Create a messaging summary (personal MSAMs only)	PMSAMCreateMsgSummary	Not applicable
Create a message	MSAMCreate	MSAMCreate
Write header information	MSAMPutAttribute MSAMPutRecipient	MSAMPutMsgHeader MSAMPutRecipient
Write letter content	MSAMPutContent	Not applicable
Write an enclosure	MSAMPutEnclosure	Not applicable
Write a block	MSAMPutBlock	MSAMPutBlock
Write a nested letter	MSAMBeginNested MSAMEndNested	MSAMBeginNested MSAMEndNested
Submit a message	MSAMSubmit	MSAMSubmit
Delete a message (personal MSAMs only)	MSAMDelete	Not applicable
Set message status (personal MSAMs only)	PMSAMSetStatus	Not applicable
Enumerate a queue (personal MSAMs only)	MSAMEnumerate	Not applicable

calling the appropriate functions to write the nested message's header information, blocks, enclosures, and so forth. When you have finished writing the message, you submit it to the AOCE system for delivery to its recipients.

A personal MSAM may also delete a letter, or both the letter and the letter's message summary, from an incoming queue. For example, the MSAM may delete a letter (but not the message summary) if it no longer wants the letter to be cached locally. If the personal MSAM is mirroring the letter's status on the external messaging system, it can delete the letter and message summary when the letter is removed from the external messaging system.

A personal MSAM may also set the status of a letter and enumerate an incoming queue. Setting the status of a letter is a task that the MSAM performs at several points while it is processing the letter. Enumerating an incoming queue is a task it may do in response to receiving a kMailEPPCInQUpdate high-level event.

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

The sample code in Listing 2-6 through Listing 2-15 illustrates one way a personal MSAM can write a letter to an incoming queue. Most of the sample code and the text also apply to a server MSAM. The text notes differences between the operation of personal and server MSAMs where applicable.

Most of these listings contain code fragments from the `DoIncomingLetter` function, but only Listing 2-6 on page 2-67 shows the `DoIncomingLetter` function definition and its local variables.

Choosing Creator and Type for Messages and Blocks

When you create an incoming message, you set the message creator to indicate the application that should open the message. If you set the message creator for a letter to `'lap2'`, the signature of the AppleMail application, the AppleMail application opens the letter when the user double-clicks the letter's icon. If the letter contains a content enclosure, you can set the message creator to the signature of the application that created the content enclosure. In this case, if the user has that application, that application will open the letter.

The message type `kMailLtrMsgType` designates an AOCE letter that contains data in standard interchange format or image format, or a regular enclosure. When you create an incoming letter, you should use this message type when the letter contains data in standard interchange format or image format, or when it contains a regular enclosure. If the letter also contains a content enclosure or a private block, and you set the message creator to the signature of the application that created the enclosure or private block, then you can use a message type that you define that is consistent with the message creator.

When you create a non-letter message, you typically use an application-defined message creator and message type.

Each block in an incoming message has a block creator and block type. When you create blocks such as header, content, enclosure, and report blocks by calling the appropriate MSAM function, the function sets the block creator to `kMailAppleMailCreator` and the block type to the correct predefined type. (Letter block types are listed in Table 2-3 on page 2-18.)

When you call the `MSAMPutBlock` function to add a block to an incoming message, you set the block creator and block type to values that you select. If you are writing a block of a predefined type such as an image block or a private block, be sure to set the block type to `kMailImageBodyType` or `kMailMSAMType`, respectively.

Creating a Letter's Message Summary

A personal MSAM must create a message summary for an incoming letter before creating the letter itself. Server MSAMs do not create message summaries at any time, and personal MSAMs do not create message summaries for non-letter messages. The need to create a message summary is related to the mode of operation in the personal MSAM. See the section "MSAM Modes of Operation" beginning on page 2-12 for information on this topic.

The function `DoIncomingLetter` shown in Listing 2-6 on page 2-67 illustrates how you can create a message summary for an incoming letter. It assumes that you previously read the letter from an external messaging system, translated it into AOCE data formats,

and saved it to disk. (Note that this method is just one way an MSAM can handle incoming letters.)

The `DoIncomingLetter` function first allocates the buffer `dataBuffer` that it uses to hold a variety of data throughout the function's execution. Then it initializes all of the fields of the message summary structure to 0 prior to setting the fields that a personal MSAM should set. At the top level of the message summary structure, `DoIncomingLetter` sets only the `version` field. You always set it to the constant `kMailMsgSummaryVersion`.

You set the bits in the attribute mask that correspond to the attributes that are present in the letter. In the `attrMask` field of the `masterData` substructure, `DoIncomingLetter` sets the bits for the send timestamp, indications, the sender of the letter, the subject of the letter, the message type, and the message family. Each external messaging system may differ in the attribute information it routinely provides. In the sample code, the external messaging system always provides a timestamp and does not provide a reply ID. For this reason, the corresponding bits in the attribute mask in the message summary are set and not set accordingly.

Once you have set the bits in the attribute mask, you write the attributes to the message summary. At a minimum, you must write the message type, send timestamp, sender, and subject attributes to the message summary. The `DoIncomingLetter` function first writes the send timestamp to the message summary by calling its `DoGetTimeStamp` routine. Next, it calls its `DoGetLetterLength` utility routine to get the approximate size of the letter.

In the `coreData` substructure, `DoIncomingLetter` explicitly provides a value for all of the fields except `agentInfo` and `letterFlags`. (The `DoIncomingLetter` function implicitly set the `letterFlags` field to 0 when it initialized the entire message summary structure to 0.) In the `letterIndications` field, it sets those bits that indicate the letter has normal priority and that it has a content block. This technique assumes that the incoming letter has no priority setting, so `DoIncomingLetter` supplies a default value here. (The `DoIncomingLetter` function also supplies a default value for content if the letter has no content. See Listing 2-11 on page 2-78.)

The `DoIncomingLetter` function sets the message type to the constant `kMailLtrMsgType` to indicate a standard AOCE letter. It sets the message creator to `kLetterCreator`, a constant for `'lap2'`, the signature of the AppleMail application. As a result, when a user double-clicks the letter, the Finder launches the AppleMail application to open the letter. Usually, an MSAM does not set a letter's creator to its own signature because the MSAM cannot open the letter and allow the user to view and edit it. However, if your MSAM is associated with a particular letter application, you should use that application's signature so that the application will launch when the user opens the letter.

The `DoIncomingLetter` function sets the message family to `kMailFamily`, indicating that the letter falls into the general class of mail messages. Next, it sets the `messageSize` field to the value returned by the `DoGetLetterLength` utility routine. The Finder uses this value when a user chooses the Get Info command from the File menu.

The `sender` and `subject` fields in the message summary deserve special attention. Each is declared as an `RString32` structure in the `MailCoreData` structure in the message summary. However, those declarations only serve to allocate space and indicate the relative order of the sender and subject data. They do not represent the actual data layout. You should treat these two fields as a common buffer containing variable-length sender and subject data. The correct order of information in the common buffer is an `RString32` structure containing the sender information (character set, data length, and sender data), padded to an even byte boundary if necessary, and followed immediately by an `RString32` structure containing the subject information. (You should also pad the subject information to an even byte boundary if necessary.) Thus, sender information always starts at a fixed place whereas subject information does not. Neither subject nor sender information may exceed `kRString32Size` bytes although either, of course, may be smaller.

The `DoIncomingLetter` function illustrates one way to write the sender and subject information to a message summary. The `DoIncomingLetter` function calls its `DoReadFromFile` utility routine to read a `PackedDSSpec` structure containing the sender's address information from the letter stored on disk. (The `DoReadFromFile` routine reads a file in which an incoming letter is stored and returns in a buffer the requested letter component and the number of bytes it placed in the buffer.) If the read operation succeeds, `DoIncomingLetter` unpacks the packed address and calls its `DoCopyFitRString` utility routine. The `DoCopyFitRString` routine copies the displayable string that identifies the sender from the `recordName` field of the unpacked address into the `sender` field of the message summary, truncating it if it is longer than `kRString32Size` bytes.

Next, `DoIncomingLetter` reads into its local variable `subject` an `RString` structure containing the subject from the stored letter. Every AOCE letter must have a subject. If the read operation fails, `DoIncomingLetter` converts a constant C string containing a default value for the subject into an `RString` and writes it to its local variable `subject`. Finally, it calls its `DoCopyFitRString` routine to copy its local variable `subject` into the message summary, truncating it if it is longer than `kRString32Size` bytes. (The `DoIncomingLetter` function copies the subject into its local variable `subject` instead of directly into the message summary because it uses the local variable when adding the subject attribute to the letter header. See Listing 2-8 on page 2-72.)

Now that both the subject and sender information are in a common buffer in the message summary, `DoIncomingLetter` adjusts the byte position at which the subject information begins. The subject information must start immediately after the sender information. `DoIncomingLetter` calculates the total length of the sender `RString`, including the fields for length and character set. If the total is an odd number, it adds 1 to get an even word boundary, then calls the `BlockMove` routine to move the subject information immediately after the end of the sender information.

IMPORTANT

Because the `sender` and `subject` fields form one common buffer into which the information is packed, using the `subject` field to access the subject information does not produce the desired result. You must compute the beginning of the subject information in the common buffer. s

At this point, the `DoIncomingLetter` function has filled in the relevant fields of the message summary. Next, it sets up the fields of the parameter block for the `PMSAMCreateMsgSummary` function. One of the parameters to `DoIncomingLetter` is a `MySlotSpec` structure, a data type defined by the personal MSAM that contains information about a slot. The personal MSAM of which `DoIncomingLetter` is a part previously stored the incoming queue reference that it obtained from the `PMSAMOpenQueues` function in the `MySlotSpec` structure. The `DoIncomingLetter` function uses that incoming queue reference to fill in the `queueRef` field of the `MSAMCreate` parameter block. Next, it sets the `msgSummary` field of the parameter block to the address of the message summary structure it has just initialized. Although `DoIncomingLetter` does not do it, you can add up to `kMailMaxPMSAMMsgSummaryData` bytes of private data in the buffer structure pointed to by the `buffer` field of the `PMSAMCreateMsgSummary` parameter block. It is a convenient way for you to store additional information related to a specific letter. Then `DoIncomingLetter` calls the `PMSAMCreateMsgSummary` function, which returns a sequence number for the letter. The `DoIncomingLetter` function must use this sequence number when it calls the `MSAMCreate` function to create the letter itself.

Listing 2-6   Creating a message summary

```
OSErr DoIncomingLetter(FSSpec *myTempFileSpec, MySlotSpec *slotSpec)
{
    OSErr             myErr;
    MSAMParam         myParamBlock;
    MSAMMsgSummary    myMsgSum;
    Ptr               dataBuffer;
    unsigned long     bufferLen;
    unsigned long     contentLength;
    RString           subject;
    RecordID          entitySpecifier;
    OCERecipient      fromAddress;
    MailMsgRef        letterRef;
    long              letterSeqNum;
    char              defaultText[256];
    unsigned char     *subjectOffset;

    #define  kLetterCreator    'lap2'   /* signature of AppleMail app */
    #define  kDefaultSubject   "<no subject>"
```

```
#define   kDefaultBody       "<no message>"
#define   kMaxBufferSize     32767L
/* constants to identify components of stored letter on disk */
#define   kFromType          '2FRM'
#define   kToType            '2MTO'
#define   kCCType            '2MCC'
#define   kBCCType           '2BCC'
#define   kTextContent       '2TXT'
#define   kPictContent       '2PIC'
#define   kSoundContent      '2SND'
#define   kContentSectionType  '2RTY'
#define   kSubjectType       '2SUB'



/* allocate buffer for reading from disk */
bufferLen   = kMaxBufferSize;
dataBuffer  = NewPtr(bufferLen);
if (MemError() != noErr)
   return MemError();

/* initialize the message summary structure to 0 */
DoClearBuffer(&myMsgSum,sizeof(MSAMMsgSummary));

/* set the version and attribute mask fields */
myMsgSum.version                             = kMailMsgSummaryVersion;
myMsgSum.masterData.attrMask.sendTimeStamp   = true;
myMsgSum.masterData.attrMask.indications     = true;
myMsgSum.masterData.attrMask.from            = true;
myMsgSum.masterData.attrMask.subject         = true;
myMsgSum.masterData.attrMask.msgType         = true;
myMsgSum.masterData.attrMask.msgFamily       = true;

/* get the timestamp and write it to message summary */
DoGetTimeStamp(myTempFileSpec,&myMsgSum.coreData.sendTime);

/* get length of stored letter data in bytes */
contentLength = kMaxBufferSize;
contentLength = DoGetLetterLength(myTempFileSpec);

/* set other core data fields */
myMsgSum.coreData.letterIndications.priority    = kIPMNormalPriority;
myMsgSum.coreData.letterIndications.hasContent  = true;
myMsgSum.coreData.letterIndications.hasStandardContent= true;
```

```
   myMsgSum.coreData.messageType.msgType            = kMailLtrMsgType;
   myMsgSum.coreData.messageType.msgCreator         = kLetterCreator;
   myMsgSum.coreData.messageFamily                  = kMailFamily;
   myMsgSum.coreData.messageSize                    = contentLength;
   myMsgSum.coreData.addressedToMe                  = kAddressedAs_TO;


/* get sender name from stored letter and write it to message summary */
   bufferLen = kMaxBufferSize;
   myErr = DoReadFromFile(myTempFileSpec, kFromType, dataBuffer,
                            &bufferLen);
   if (myErr != noErr) {
      DisposPtr(dataBuffer);
      return myErr;
   }
   OCEUnpackDSSpec((PackedDSSpec*)dataBuffer, &fromAddress,
                    &entitySpecifier);
   DoCopyFitRString(entitySpecifier.local.recordName,
                  (RStringPtr)&myMsgSum.coreData.sender, kRString32Size);



/* get subject from stored letter and write it to message summary */
   bufferLen = kMaxBufferSize;
   myErr = DoReadFromFile(myTempFileSpec, kSubjectType, &subject,
                            &bufferLen);
   if (myErr != noErr)
      OCECToRString(kDefaultSubject, smRoman, &subject, kRStringMaxBytes);
   DoCopyFitRString(&subject, (RStringPtr)&myMsgSum.coreData.subject,
                  kRString32Size);

/* calculate subject offset and move subject flush with sender */
   subjectOffset = ((unsigned char *)&myMsgSum.coreData.sender) +
                    myMsgSum.coreData.sender.dataLength + sizeof(long);
   if ((unsigned long)subjectOffset % 2)
      subjectOffset++;
   BlockMove(&myMsgSum.coreData.subject, subjectOffset,
              myMsgSum.coreData.subject.dataLength + sizeof(long));

/*
   All required fields have been set. Create the message summary. Save the
   letter's sequence number.
*/
   myParamBlock.header.ioCompletion             = (ProcPtr)DoMSAMCompletion;
   myParamBlock.pmsamCreateMsgSummary.inQueueRef   = slotSpec->inQueue;
```

```
myParamBlock.pmsamCreateMsgSummary.msgSummary    = &myMsgSum;
myParamBlock.pmsamCreateMsgSummary.buffer        = nil;
PMSAMCreateMsgSummary(&myParamBlock,true);
myErr = DoWaitPBDone(&myParamBlock);
if (myErr != noErr) {
    DisposPtr(dataBuffer);
    return myErr;
}
letterSeqNum = myParamBlock.pmsamCreateMsgSummary.seqNum;
```

Creating a Letter

After creating a message summary, a personal MSAM may write the letter associated with the message summary to the incoming queue immediately or at a later time. The choice of methods should depend on the speed of the link connecting your personal MSAM to its external messaging system. If the link is fast, you can download the letter on demand—that is, when the user opens it. If the link is slow, you should cache the letter locally so that there is no untimely delay when the user opens it. The function `DoIncomingLetter` writes the letter immediately. Listing 2-7 is a code fragment from `DoIncomingLetter` that shows how you create a letter.

The `DoIncomingLetter` function sets up the fields of the parameter block for the `MSAMCreate` function. It checks whether the letter has a blind copy recipient and sets the `bccRecipients` field accordingly. It uses the incoming queue reference originally obtained from the `PMSAMOpenQueues` function to fill in the `queueRef` field of the parameter block. Then `DoIncomingLetter` sets the `asLetter` field to `true` to indicate that the message it is creating is a letter. Because it is creating a letter, it must set the `msgType.format` field to `kIPMOSFormatType`. This setting indicates that the rest of the `IPMMsgType` structure contained in the `msgType.format` field consists of an `OCECreatorType` structure. Then `DoIncomingLetter` sets the letter's creator and type to the same values it used when it created the letter's message summary. It sets the `seqNum` field to the sequence number it obtained from the `PMSAMCreateMsgSummary` function.

Once `DoIncomingLetter` has finished initializing the parameter block, it calls the `MSAMCreate` function. The function returns a reference to the new letter, which `DoIncomingLetter` saves. The `DoIncomingLetter` function must provide the reference to all subsequent functions that add various components to the letter.

Listing 2-7   Creating a letter

```
/* check for bcc recipients */
bufferLen = kMaxBufferSize;
myErr = DoReadFromFile(myTempFileSpec, kBCCType, dataBuffer, &bufferLen);
myParamBlock.msamCreate.bccRecipients  = (myErr == noErr);
```

```
/* fill in the rest of the parameter block and create the letter */
myParamBlock.header.ioCompletion        = (ProcPtr)DoMSAMCompletion;
myParamBlock.msamCreate.queueRef        = slotSpec->inQueue;
myParamBlock.msamCreate.asLetter        = true;
myParamBlock.msamCreate.msgType.format = kIPMOSFormatType;
myParamBlock.msamCreate.msgType.theType.msgOSType.msgCreator =
                                    kLetterCreator;
myParamBlock.msamCreate.msgType.theType.msgOSType.msgType =
                                    kMailLtrMsgType;
myParamBlock.msamCreate.seqNum          = letterSeqNum;
myParamBlock.msamCreate.tunnelForm      = false;
MSAMCreate(&myParamBlock, true);
myErr = DoWaitPBDone(&myParamBlock);
if (myErr != noErr) {
   DisposPtr(dataBuffer);
   return myErr;
}
letterRef = myParamBlock.msamCreate.newRef;
```

A server MSAM does basically the same things to create a letter, with the following differences. A server MSAM uses the queue reference that it obtained from the SMSAMStartup function to fill in the queueRef field. Because server MSAMs do not create message summaries, there is no need to ascertain that the values provided to the MSAMCreate function for the creator and type exactly match those in the message summary. A server MSAM does not supply a value in the seqNum field of the MSAMCreate parameter block.

Creating a Non-Letter Message

When you create a non-letter message instead of a letter, the following differences apply for both personal and server MSAMs:

n You must set the myParamBlock.msamCreate.asLetter field to false.

n You can set the myParamBlock.msamCreate.msgType.format field to either kIPMOSFormatType (which specifies that the message creator and message type information is formatted as type OCECreatorType) or kIPMStringFormatType (which specifies that the message creator and message type information is formatted as type Str32). Typically, you use type OCECreatorType; type Str32 is included for compatibility with the Program-to-Program Communications (PPC) Toolbox.

n You may set the myParamBlock.msamCreate.refCon field to a private value. The MSAMCreate function stores that value in the message header. A recipient can retrieve the value with the MSAMGetMsgHeader function.

n You do not supply a value in the myParamBlock.msamCreate.bccRecipients field.

In addition, a personal MSAM does not supply a value in the myParamBlock.msamCreate.seqNum field.

Writing Letter Attributes

Once you have created a letter, you add the component parts to the letter. To add information to a letter's header, you use the MSAMPutAttribute function. Listing 2-8, a code fragment from the DoIncomingLetter function, shows how you add attributes to a letter header.

The MSAMPutAttribute function allows you to add one attribute each time you call it. The DoIncomingLetter function adds the send timestamp, indications, message family, and subject attributes to the letter's header by copying the values it previously stored in the letter's message summary. Each time it calls the MSAMPutAttribute function, DoIncomingLetter sets the mailMsgRef field to indicate the letter to which it wants to add the attribute. It sets the attrID field to a constant that indicates the type of attribute it wants to add. Then it specifies the buffer in which the attribute data is located, specifies the buffer size, and calls the MSAMPutAttribute function to add the attribute to the letter header. Note that when it writes the subject, DoIncomingLetter does not use the C function sizeof to get the size of the subject attribute because that would return the size of an RString structure. Instead, it computes the exact size of the subject string in the buffer by using the actual length of the subject, which is specified in the subject.dataLength field, and then adding 4 bytes for the dataLength and charSet fields of the RString structure. If the number of bytes turns out to be odd, it adds 1 to make an even length.

The DoIncomingLetter function does not add the letter creator and type to the letter header. That information was already added when DoIncomingLetter called the MSAMCreate function.

Once the parameter block is initialized, DoIncomingLetter calls the MSAMPutAttribute function. If the function returns an error, DoIncomingLetter calls its DoCancelOnSubmit function, which disposes of the data buffer, calls the MSAMSubmit function to delete the unfinished letter, and calls the MSAMDelete function to delete the message summary.

Listing 2-8   Adding attributes to a letter header

```
/* add the time */
myParamBlock.msamPutAttribute.mailMsgRef         = letterRef;
myParamBlock.msamPutAttribute.attrID             = kMailSendTimeStampBit;
myParamBlock.msamPutAttribute.buffer.buffer      =
                                    (Ptr)&myMsgSum.coreData.sendTime;
myParamBlock.msamPutAttribute.buffer.bufferSize = sizeof(MailTime);
MSAMPutAttribute(&myParamBlock, true);
myErr = DoWaitPBDone(&myParamBlock);
if (myErr != noErr) {
   DoCancelOnSubmit(letterRef, letterSeqNum, slotSpec->inQueue,
                  dataBuffer);
   return myErr;
}
```

```
/* add the indications */
myParamBlock.msamPutAttribute.mailMsgRef        = letterRef;
myParamBlock.msamPutAttribute.attrID            = kMailIndicationsBit;
myParamBlock.msamPutAttribute.buffer.buffer     =
                                (Ptr)&myMsgSum.coreData.letterIndications;
myParamBlock.msamPutAttribute.buffer.bufferSize = sizeof(MailIndications);
MSAMPutAttribute(&myParamBlock, true);
/*
   Call DoWaitPBDone and check for error. Then use the same logic used
   to add the time and indications to add the message family.
*/


/* add the subject */
myParamBlock.msamPutAttribute.mailMsgRef        = letterRef;
myParamBlock.msamPutAttribute.attrID            = kMailSubjectBit;
myParamBlock.msamPutAttribute.buffer.buffer     = (Ptr)&subject;
myParamBlock.msamPutAttribute.buffer.bufferSize = subject.dataLength + 4;
if ((myParamBlock.msamPutAttribute.buffer.bufferSize % 2) != 0)
   myParamBlock.msamPutAttribute.buffer.bufferSize++;
MSAMPutAttribute(&myParamBlock, true);
/* call DoWaitPBDone and check for error */
```

A server MSAM does not have a message summary from which to copy attribute values, so it would extract the attribute values from the incoming letter itself.

Note
The `MSAMPutAttribute` function does not apply to non-letter messages. In dealing with an incoming non-letter message, both personal and server MSAMs can add attributes to the message header by calling the `MSAMPutMsgHeader` function. u

Writing Addresses

Although the different types of recipients—From, To, cc, and bcc—are letter attributes, you do not add them to a letter using the `MSAMPutAttribute` function. Instead, you use the `MSAMPutRecipient` function. Each time you call the `MSAMPutRecipient` function, you can add one recipient to a letter. This function requires you to add all of the recipients of one type before adding any recipient of another type. The code fragment from the `DoIncomingLetter` function shown in Listing 2-9 demonstrates how you can add recipients to a letter.

The `DoIncomingLetter` function calls its `DoAddTheRecipients` function four times, once for each type of recipient, to actually add the recipient information to the letter. It passes several parameters to `DoAddTheRecipients`:

n   the reference number of the letter to which it wants to add a recipient

n   a pointer to the file specification of the temporary file containing the translated incoming letter

n   a constant that identifies the disk file component for a given type of recipient

n   the type of recipient to add (an attribute ID)

n   a pointer to its buffer

n   the size of the buffer

If `DoAddTheRecipients` returns an error for any type of recipient,
`DoIncomingLetter` terminates writing the letter.

Listing 2-9   Adding recipients to a letter

```
/*
    Add the recipients. Check for error after calling DoAddTheRecipients
    for each recipient type.(Shown only the first time in the following
    code.)
*/
myErr = DoAddTheRecipients(letterRef, myTempFileSpec, kFromType,
                            kMailFromBit, dataBuffer, kMaxBufferSize);
if (myErr != noErr) {
    DoCancelOnSubmit(letterRef, letterSeqNum, slotSpec->inQueue,
                    dataBuffer);
    return myErr;
}

myErr = DoAddTheRecipients(letterRef, myTempFileSpec, kToType, kMailToBit,
                            dataBuffer, kMaxBufferSize);

myErr = DoAddTheRecipients(letterRef, myTempFileSpec, kCcType, kMailCcBit,
                            dataBuffer, kMaxBufferSize);

myErr = DoAddTheRecipients(letterRef, myTempFileSpec, kBccType,
                            kMailBccBit, dataBuffer, kMaxBufferSize);
```

The `DoAddTheRecipients` function is shown in Listing 2-10. It is a utility routine that
can add any type of recipient to a given letter. It assumes that the MSAM has previously
written the letter's recipient information to a file in the form of a `PackedDSSpec`
structure. For a given type of recipient, `DoAddTheRecipients` reads one recipient at a
time, and places the information in a buffer. Then it unpacks the `PackedDSSpec`
structure and fills in the fields of the parameter block for the `MSAMPutRecipient`
function.

The `DoAddTheRecipients` function sets the `mailMsgRef` and `attrID` fields to the
values it was passed by `DoIncomingLetter` for the letter's reference number and the
recipient type attribute ID, respectively. It sets the `recipient` field to the unpacked
`DSSpec` structure it got by calling the `OCEUnpackDSSpec` routine. Then it sets the
`responsible` field to `false`.

A personal MSAM always sets the `responsible` field of the parameter block for
`MSAMPutRecipient` to `false` when it is adding a recipient to a letter. For a non-letter
message, however, it should set the `responsible` field to `false` only when the
recipient address is not local to the computer on which the personal MSAM is running.
Setting the `responsible` field to `true` for a non-letter message indicates that you want
the AOCE system to be responsible for delivering the message to its destination on the
local computer.

A server MSAM should set the `responsible` field to `true` to indicate that the AOCE
system should deliver the message to the recipient. This applies to both letter and
non-letter messages.

Finally, `DoAddTheRecipients` calls the `MSAMPutRecipient` function. The
`DoAddTheRecipients` function repeats this cycle until either the `MSAMPutRecipient`
function returns an error or there are no more recipients of a given type for the letter.

Listing 2-10   Adding a specific type of recipient

```
OSErr DoAddTheRecipients(MailMsgRef mailRef, FSSpec *myTempFileSpec,
                    OSType recipType, MailAttributeID attrID,
                    Ptr dataBuffer, unsigned long bufferLen)

{
   OSErr          myErr;
   Boolean        moreRecipients = true;
   unsigned long  gotLength;
   OCERecipient   recipient;
   RecordID       entitySpecifier;
   MSAMParam      myParamBlock;


   do {
      gotLength = bufferLen;
      myErr = DoReadFromFile(myTempFileSpec, recipType, dataBuffer,
                             &gotLength);
      if (myErr == noErr && gotLength > 0) {

         /* unpack a recipient, initialize the parameter block,
            add the recipient */
         OCEUnpackDSSpec((PackedDSSpec*)dataBuffer, &recipient,
                      &entitySpecifier);
         myParamBlock.msamPutRecipient.ioCompletion   =
                                            (ProcPtr)DoMSAMCompletion;
         myParamBlock.msamPutRecipient.mailMsgRef  = mailRef;
         myParamBlock.msamPutRecipient.attrID      = attrID;
```

```
        myParamBlock.msamPutRecipient.recipient   = &recipient;
        myParamBlock.msamPutRecipient.responsible = false;
        MSAMPutRecipient(&myParamBlock, true);
        myErr = DoWaitPBDone(&myParamBlock);
    }
    else {
        moreRecipients    = false;
        myErr             = noErr;
    }

} while (myErr == noErr && moreRecipients);

return myErr;
}
```

Writing Letter Content

A letter's content block consists of a series of one or more segments, each containing data of one of the following types: plain text, styled text, pictures, sounds, and QuickTime movies. To add a content block to an incoming letter, you call the `MSAMPutContent` function.

You provide the function with a buffer containing data of a given type and tell it what type of data is in the buffer. The first time you call the `MSAMPutContent` function, set the `append` field to `false` to tell the function to begin a new segment. On subsequent calls to the function, you set the `append` field to `true` or `false`, depending on whether you want your data placed in a new segment or appended to the current one.

When you add a text segment, you must specify values for the `startNewScript` and `script` fields. The value of the `startNewScript` field (`true` or `false`) tells the `MSAMPutContent` function whether the data in your buffer uses a different character set than that of text data you previously wrote. You set the `script` field to a code that indicates the character set of your data. (See *Inside Macintosh: Text* for a list of script codes.)

When you add a styled text segment, you provide the style information in a style scrap structure (`StScrpRec` structure). You should allocate the `StScrpRec` structure dynamically because it is a very large structure. See the `MSAMPutContent` function description on page 2-186 for more information on adding styled text.

You must add all of a letter's content sequentially. For instance, you cannot call `MSAMPutContent` to add some of the content, call `MSAMPutBlock` to add a private block, and then call `MSAMPutContent` again to add the remainder of the content. Once you call `MSAMPutContent`, calling any other function in the MSAM API terminates the content block for the letter. If you call the `MSAMPutContent` function again for the same letter, it returns the `kMailInvalidOrder` result code. The `MSAMPutContent` function adds the segments to the letter in the order you provide them.

The `DoWriteLetterContent` function in Listing 2-11 shows one way to add content to an incoming letter. It assumes the MSAM has previously stored a letter from its external messaging system in a disk file. The file is composed of a series of sections corresponding to different components of the letter. The content component of the stored letter consists of a series of sections, similar to the segments in a letter's content block, each of which contains a single type of data.

The `DoWriteLetterContent` function starts by initializing the fields of the `MSAMPutContent` function's parameter block that won't change regardless of what it reads from its file. It sets the `mailMsgRef` field to the letter's reference number. It sets the `textScrap` field to `nil` because it does not handle styled text. Because this MSAM handles just one character set, `DoWriteLetterContent` sets the `script` field to `smRoman` and never changes this setting. It sets the `append` field to `false` because it intends that each block of data that it previously stored on disk be written to a separate segment in the letter's content block.

The `DoWriteLetterContent` function initializes its local variable `contentType` to indicate that it wants to read the content section of its stored letter. It sets the local variable `contentWritten` to `false` because it has not yet written a segment to the incoming letter.

Then `DoWriteLetterContent` reads sequentially through the content sections of the stored letter. It repeatedly calls the `DoReadFromFile` utility routine to read a buffer of data from the file. The `DoReadFromFile` function returns one content section from the file each time it is called. The buffer is large enough to hold any content section that the MSAM previously stored. After reading each section, `DoWriteLetterContent` determines the type of data in the section and sets the `segmentType` field accordingly. Because this MSAM handles only plain text, picture, or sound data, the content sections can contain only these types of data. If `DoReadFromFile` returns plain text data, `DoWriteLetterContent` sets the `startNewScript` field to `true`. This tells the `MSAMPutContent` function to examine the `script` field to discover the character set of the text in the buffer. Typically, you set this field to `true` when you first add a plain text segment and thereafter whenever the character set of the text changes (which does not apply to this MSAM) or you've called `MSAMPutContent` to add some other type of segment. Last, `DoWriteLetterContent` sets the `bufferSize` field to the number of bytes it read from its disk file and calls the `MSAMPutContent` function to write the data to the letter's content block. If the `MSAMPutContent` function returns successfully, `DoWriteLetterContent` sets the local variable `contentWritten` to `true`. The `DoWriteLetterContent` function continues to read from its file and write segments to the letter's content block until it has read all the content sections in the file or it encounters an error.

When `DoWriteLetterContent` has finished reading the content sections, it tests the local variable `contentWritten`. If it failed to write any data successfully, `DoWriteLetterContent` copies a default string into its buffer and calls the `MSAMPutContent` function. It must do this to provide some content since it set the `hasContent` bit in the `indications` attribute in the letter's header. (See Listing 2-6 on page 2-67.)

Listing 2-11 Writing letter content

```
OSErr DoWriteLetterContent(FSSpec *myTempFileSpec, MailMsgRef myMailRef,
                           Ptr dataBuffer)
{
   unsigned long  bufferLen;
   OSType         contentType;
   Boolean        contentWritten;
   MSAMParam      myParamBlock;
   OSErr          myErr,myErr2;

   myParamBlock.header.ioCompletion            = (ProcPtr)MSAMCompletion;
   myParamBlock.msamPutContent.mailMsgRef     = myMailRef;
   myParamBlock.msamPutContent.textScrap      = nil;
   myParamBlock.msamPutContent.buffer.buffer = dataBuffer;
   myParamBlock.msamPutContent.script         = smRoman;
   myParamBlock.msamPutContent.append         = false;

   contentType      = kContentSectionType;
   contentWritten   = false;

   do {        /* for each content section in the temp file */
      bufferLen = kMaxBufferSize;
      myErr = DoReadFromFile(myTempFileSpec, contentType, dataBuffer,
                             &bufferLen);
      switch (contentType) {  /* determine segment type */
         case kTextContent:
            myParamBlock.msamPutContent.segmentType = kMailTextSegmentType;
            myParamBlock.msamPutContent.startNewScript = true;
            break;
         case kPictContent:
            myParamBlock.msamPutContent.segmentType = kMailPictSegmentType;
            break;
         case kSoundContent:
            myParamBlock.msamPutContent.segmentType = kMailSoundSegmentType;
            break;
      } /* endswitch */
      myParamBlock.msamPutContent.buffer.bufferSize= bufferLen;
      if (myErr == noErr) {
         MSAMPutContent(&myParamBlock,true);
         myErr2 = WaitPBDone(&myParamBlock);
         if (myErr2 != noErr)
            return myErr2;
         contentWritten = true; /* don't need default content */
```

```
      }  /* endif */
} while (myErr != noErr);

if (myErr == kEndOfContentSections)
   myErr = noErr;

/* if no content written, write default content */
if (contentWritten == false) {
   strcpy(dataBuffer,kDefaultBody);
   myParamBlock.msamPutContent.segmentType = kMailTextSegmentType;
   myParamBlock.msamPutContent.buffer.bufferSize = strlen(kDefaultBody);
   MSAMPutContent(&myParamBlock,true);
   myErr = WaitPBDone(&myParamBlock);
}

return myErr;
}
```

You call the MSAMPutContent function to add content to letters only. You do not call it to write data to a non-letter message.

Submitting a Message

After composing a message, an MSAM calls the MSAMSubmit function to submit the message to the AOCE system for delivery. A message must be complete before you submit it because, when the MSAMSubmit function completes execution, the message's reference number is invalid and you cannot change the message in any way.

Listing 2-12 is a code fragment from the DoIncomingLetter function that shows how you can submit a letter for delivery. The DoIncomingLetter function sets the mailMsgRef field to the letter's reference number and the submitFlag field to true to indicate that the letter is ready for delivery. If you set the submitFlag field to false, the function deletes the letter. Then DoIncomingLetter calls the MSAMSubmit function.

If MSAMSubmit returns an error, DoIncomingLetter calls the MSAMDelete function to delete the message summary associated with the letter. The DoIncomingLetter function sets the queueRef field to the reference value that identifies the incoming queue in which the message summary is located. (It originally obtained this value from the PMSAMOpenQueues function.) Then it sets the seqNum field to the sequence number that identifies the message summary. Last, DoIncomingLetter sets the msgOnly field to false. This tells MSAMDelete to delete the letter and its message summary. In this case, there is no letter to delete. The MSAMDelete function deletes the message summary and returns the result code noErr.

Listing 2-12 Submitting a letter

```
/* submit the letter */
myParamBlock.msamSubmit.mailMsgRef = letterRef;
myParamBlock.msamSubmit.submitFlag = true;
myErr = MSAMSubmit(&myParamBlock);
if (myErr != noErr) {                            /* delete message summary */
    myParamBlock.msamDelete.queueRef = slotSpec->inQueue;
    myParamBlock.msamDelete.seqNum   = msgSeqNum;
    myParamBlock.msamDelete.msgOnly  = false;
    myParamBlock.msamDelete.result   = noErr;
    MSAMDelete(&myParamBlock, true);
    DoWaitPBDone(&myParamBlock);
}
DisposPtr(dataBuffer);

return myErr;
```

If `DoIncomingLetter` had been dealing with a non-letter message, it would not need to delete a message summary, because a personal MSAM only creates a message summary for a letter. A server MSAM, of course, does not need to delete a message summary because it never creates one.

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.

Receiving a Report

An MSAM can receive reports about incoming messages. Server MSAMs can receive reports on both letters and non-letter messages. Personal MSAMs can receive reports on non-letter messages only.

To request a report on a non-letter message, an MSAM should set the appropriate bits in the `deliveryNotification` field when it calls the `MSAMPutMsgHeader` function. You set the bits by using the `kIPMDeliveryNotificationMask` or `kIPMNonDeliveryNotificationMask` masks to request delivery and non-delivery indications.

To request a report on a letter, a server MSAM should set the `receiptReports` bit, the `nonReceiptReports` bit, or both in the letter's `MailIndications` attribute.

Because personal MSAMs do not receive reports on letters, the IPM Manager ignores the setting of the `receiptReports` and `nonReceiptReports` bits in a letter's `MailIndications` attribute for any letter submitted by a personal MSAM. Instead, the result code of the `MSAMSubmit` function tells a personal MSAM if the letter delivery attempt was successful or not.

The report that an MSAM receives never includes a copy of the original message. Thus, the IPM Manager ignores the bits in a letter's indications attribute and a non-letter message's header that have to do with enclosing a copy of the original with the report.

An MSAM can identify a report from the IPM Manager in its outgoing queue because all such reports have a message creator of `kIPMSignature` and a message type of `kIPMReportNotify`.

An MSAM reads a report by calling the `MSAMOpen`, `MSAMGetMsgHeader`, and `MSAMGetBlock` functions. Reports consist of a recipient report block (type `kMailReportType`) and possibly a private data block (type `kMailMSAMType`). The recipient report block contains a report header and information about some number of recipients. (See the chapter "Interprogram Messaging Manager" in *Inside Macintosh: AOCE Application Interfaces* for a description of the report header `IPMReportBlockHeader` and the recipient report information structure `OCERecipientReport`.) If an MSAM added a private data block to a message, the IPM Manager includes a copy of that block in the report.

A report may contain information on one or more AOCE recipients. The IPM Manager attempts to report as quickly as possible on each recipient. If there is some difficulty in reporting, it sends a report on the recipients about which it has information and sends another report about the remaining recipients at a later time. Therefore, if a message that the MSAM put into an AOCE system has several recipients, the MSAM may get several reports. If the MSAM plans to forward that information to its external messaging system, it may want to consolidate the information from the reports before forwarding it.

Note
The AOCE software defines successful delivery to mean that the message was placed in the recipient's incoming queue. It does not imply that the message was actually opened or read. u

Deleting a Message

A personal MSAM should not delete messages from its outgoing queues. Messages should stay in an outgoing queue so that the user can look at 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 MSAM should delete the letter. A server MSAM does delete messages from its outgoing queue.

A personal MSAM can delete letters from an incoming queue. It can delete only a letter or both a letter and the associated message summary. For example, the MSAM 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 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.

Note

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. u

The `MSAMDelete` function removes a message from the queue that you specify. You identify the message by its sequence number, which you obtain from the `MSAMEnumerate` function. Once you have deleted a message, it is no longer available to you on the Macintosh computer on which your MSAM is running. (The message may still exist on the external messaging system.)

Translating Addresses

One of an MSAM's primary tasks is translating address information from AOCE format to the format of its external messaging system and vice versa. Within AOCE software, an address is defined by an `OCERecipient` structure, a complex structure that contains other structures and elemental fields. It is described on page 2-106. Figure 2-13 on page 2-28 illustrates the fields in an `OCERecipient` structure and their relationship to each other. Table 2-4 on page 2-29 lists what each field should contain for a non-AOCE address. Table 2-5 on page 2-30 lists the contents of each field when the `OCERecipient` structure contains an AOCE address. If you are already familiar with the information in Figure 2-13, Table 2-4, and Table 2-5, you'll find the listings and descriptions in the sections "Translating From an AOCE Address" and "Translating to an AOCE Address" easier to understand.

Note that an `OCERecipient` structure is identical to a `DSSpec` structure.

Within this chapter and the MSAM API, an address is often referred to as an *xxx recipient*, where *xxx* specifies a type of recipient—To, From, cc, or bcc.

A non-letter message contains only From and To recipients. A letter may contain any type of recipients.

An address can become known to an AOCE system by any of the following methods:

n   the user provides the address information by means of an address template (see the chapter "Service Access Module Setup" in this book for an explanation of address templates)

n   the address is read from an incoming message

n   the user types in the address when using a mailer (this works only if the extension value portion of the address is formatted as a single `RString`; see the chapter "Standard Mail Package" in *Inside Macintosh: AOCE Application Interfaces* for an explanation of the mailer and type-in addressing)

n   the address exists in a catalog and can be retrieved by the user or an application

The MSAM whose code is shown in the sections that follow is a personal MSAM that connects to an SMTP messaging system. The address format understood by the SMTP messaging system is a string of this form: *username@systemlocation*. The information presented applies to server MSAMs as well.

Translating From an AOCE Address

Prior to transmitting a letter to its external messaging system, an MSAM must convert the address information from AOCE format (an `OCERecipient` structure) to the format understood by its external messaging system.

The function `DoBuildSMTPAddressInfo` in Listing 2-13 provides an example of building a non-AOCE address from an `OCERecipient` structure. The `DoBuildSMTPAddressInfo` function first allocates a buffer pointed to by `addressBuf`. This address buffer will eventually hold all of the SMTP address information for a given letter except the bcc recipients, which are stored in a separate buffer. The `DoBuildSMTPAddressInfo` function sets the first byte in the address buffer to 0 to indicate an empty string.

When it is launched, this MSAM creates and maintains a `MySlotSpec` structure for each mail slot for which it is responsible. This privately defined structure contains all the information relevant to a individual slot. To build the From address, the `DoBuildSMTPAddressInfo` function begins by copying the user name from the `MySlotSpec` structure for the slot it is processing into the local variable `fromAddr`. Then the function appends to the user name the `@` character and the SMTP server name, which it also copies from the `MySlotSpec` structure. Once it has finished building the string holding the actual From address, `DoBuildSMTPAddressInfo` builds a second string in the address buffer that includes formatting information. First, it copies the constant `kMyFromHeader` into `addressBuf` to label the address. The constant's value is `"From: "`. Next, it appends the From address in `fromAddr` to the contents of the address buffer. Finally, it appends a carriage return. At this point, the contents of the address buffer look like this:

```
From: username@systemLocation(CR)0
```

Next, `DoBuildSMTPAddressInfo` adds the To addresses. To the address buffer, it adds the string `"To: "` to label the address. It initializes the `hasRecipient` Boolean variable to `false` to indicate that at this point it has found no To recipients. Then it repeats the following procedure until it encounters an error:

n Read a To address from a temporary file. The MSAM created this file when it read the letter from AOCE. If there are no more To addresses, it will get an error here.

n If the read succeeded

    n call the `DoAOCEToSMTPAddress` function (see Listing 2-14 on page 2-87), which converts an AOCE address into an SMTP address

    n append the SMTP address and a comma to the contents of the address buffer

    n set the `hasRecipient` Boolean to `true`

At this point, `DoBuildSMTPAddressInfo` completes the formatting. If it added any To addresses to the address buffer, it overwrites the last comma with the string terminator 0 and then appends a carriage return. The contents of the address buffer now look like this:

```
From: username@systemLocation(CR)To: recipient1@location,
recipient2@location,...,recipientN@location(CR)0
```

If it has not added any To addresses to the address buffer, it positions the string terminator 0 immediately before the `"To: "` label, in effect erasing it.

The `DoBuildSMTPAddressInfo` function processes a letter that has no To recipient for two reasons. First, AOCE software considers valid a letter whose header has at least one To, cc, or bcc recipient. Therefore, it is possible for an MSAM to get a letter from its AOCE system that has no To recipient. Second, as you will see in Listing 2-14 on page 2-87, this MSAM translates only SMTP addresses. It is possible that all of the To recipients for a given letter are non-SMTP addresses, but that one or more of the cc or bcc addresses are SMTP addresses. This topic is discussed in more detail in the explanation of Listing 2-14.

The `DoBuildSMTPAddressInfo` function adds the cc addresses to the address buffer in exactly the same manner as it added the To address. At this point, the address buffer contains a string that includes the From, To, and cc addresses, formatted with commas and carriage returns, and terminated by a `NULL` character.

For bcc addresses, `DoBuildSMTPAddressInfo` uses the same procedure but a separate buffer, `bccBuf`. Typically, an SMTP messaging system does not display a bcc address even to a bcc recipient. Therefore, `DoBuildSMTPAddressInfo` places any bcc addresses in a separate buffer so they can be handled separately. In code not shown in Listing 2-13, the `DoBuildSMTPAddressInfo` function uses the information in the address buffer for both routing and display purposes, but it uses the address information in the bcc buffer for routing only.

When `DoBuildSMTPAddressInfo` has finished building its two address buffers, it adds them to the letter.

Listing 2-13   Building SMTP addresses

```
OSErr DoBuildSMTPAddressInfo(FSSpec *myTempFileSpec, MySlotSpec *slotSpec)
{
    #define   kMyMaxAddrBufSize      4096     /* this MSAM's limit on address
                                                 info */
    #define   kMyFromHeader          "From: "
    #define   kMyToHeader            "To: "
    #define   kMyCCHeader            "Cc: "
    #define   kMyBCCHeader           "Bcc: "
    #define   kMyAddressDelimiter    ", "
    #define   kMyCRStr               "\r"

    OSErr          myErr;
    char           tmpString[256];
    char           bccBuf[256];
    char           fromAddr[256];
    char           *addressBuf;
    unsigned long  tmpLen;
```

```
char          packedRecip[kMaxRecipSize];
Boolean       hasRecipient;

/* allocate memory to hold addresses in external form */
addressBuf = NewPtr(kMyMaxAddrBufSize);
if (MemError() != noErr) {
   return (MemError());
}
addressBuf[0] = 0;

/* build 'from' address */
strcpy(fromAddr, slotSpec->dirIdentity.userName);
strcat(fromAddr, "@");
strcat(fromAddr, slotSpec->specInfo.smtpServer);
strcpy(addressBuf, kMyFromHeader);
strcat(addressBuf, fromAddr);
strcat(addressBuf, kMyCRStr);

/* build 'To' address */
hasRecipient = false;
strcat(addressBuf, kMyToHeader);
for (myErr = noErr; myErr == noErr; ) {
   tmpLen = kMaxRecipSize;
   myErr = DoReadFromFile(myTempFileSpec, kToType, (Ptr)packedRecip,
                          &tmpLen);
   if (myErr == noErr) {
      if (DoAOCEToSMTPAddress(
                       (OCEPackedRecipient *)packedRecip, tmpString)) {
         strcat(addressBuf, tmpString);
         strcat(addressBuf, kMyAddressDelimiter);
         hasRecipient = true;
      }
   }
}
if (hasRecipient) {
   addressBuf[strlen(addressBuf) - strlen(kMyAddressDelimiter)] = 0;
   strcat(addressBuf, kMyCRStr);
}
else {
   addressBuf[strlen(addressBuf) - strlen(kMyToHeader)] = 0;
}

/* not shown here -- build 'cc' address just like 'To' address */
```

```
/* build 'bcc' address just like 'To' address but in separate buffer */
hasRecipient = false;
strcpy(bccBuf,kMyBCCHeader);
for (myErr=noErr; myErr==noErr; ) {
    tmpLen = kMaxRecipSize;
    myErr = DoReadFromFile(myTempFileSpec,kBCCType, (Ptr)packedRecip,
                           &tmpLen);
    if (myErr==noErr) {
        if (DoAOCEToSMTPAddress(
                            (OCEPackedRecipient *)packedRecip,tmpString)) {
            strcat(bccBuf,tmpString);
            strcat(bccBuf,kMyAddressDelimiter);
            hasRecipient = true;
        }
    }
}
if (hasRecipient) {
    bccBuf[strlen(bccBuf)-strlen(kMyAddressDelimiter)] = 0;
    strcat(bccBuf,kMyCRStr);
}

/* not shown here -- add address information to the letter */

DisposPtr(addressBuf);
return noErr;
}
```

The `DoAOCEToSMTPAddress` function in Listing 2-14 converts an SMTP address contained in an `OCEPackedRecipient` structure into string format. It returns `true` when it produces an SMTP address from an `OCEPackedRecipient` structure.

The `DoAOCEToSMTPAddress` function calls the `OCEUnpackDSSpec` AOCE utility routine to unpack the packed recipient information pointed to by its `packedRecip` parameter. If the extension type of the unpacked address specifies an SMTP address, it calls the `BlockMove` function to copy the value from the `extensionValue` field into the `RString` structure `recipRString`, converts the `RString` in `recipRString` into a C string, and stores the C string in the buffer pointed to by its `unixRecip` parameter. Then it returns `true`. If the extension type specifies some other type of address, the `DoAOCEToSMTPAddress` function makes no effort to translate the address and simply returns `false`.

A user can send a single letter to recipients in different types of messaging systems; thus, a single AOCE letter header may contain addresses with different extension types. This creates a potential problem for an MSAM, which is illustrated in the following example. The SMTP messaging system to which our sample MSAM is connected understands

Listing 2-14 Converting from AOCE to SMTP address

```
Boolean DoAOCEToSMTPAddress(OCEPackedRecipient *packedRecip,
                                    char *unixRecip)
{
    #define   kMySMTPAddrType'SMTP'

    OCERecipient    recip;
    RecordID        entitySpecifier;
    OSType          recipType;
    RString         recipRString;


    OCEUnpackDSSpec((PackedDSSpec*)packedRecip, &recip, &entitySpecifier);
    recipType = recip.extensionType;
    switch (recipType) {
        case kMySMTPAddrType:
            BlockMove(recip.extensionValue, &recipRString, recip.extensionSize);
            DoRToCString(&recipRString, unixRecip);
            break;
        default:                        /* if not SMTP address, don't convert it */
            return false;
            break;
    }
    return true;
}
```

only SMTP addresses. When the messaging system receives a letter, it tries to route the letter to all of the addresses in the letter header. If it cannot do this, it generates an error reply to the sender. Suppose an AOCE user sends a letter to a fax address and sends a copy to a recipient with an SMTP address. Our sample MSAM is responsible for this SMTP address and must deliver the letter to the SMTP recipient. How should the MSAM handle the fax address? It cannot add the fax address as the To recipient because the SMTP messaging system will complain. Yet, it should provide the SMTP recipient with a letter that shows that the letter's primary recipient was a fax address.

The solution to this dilemma is up to the MSAM and its messaging system. For instance, the MSAM can copy the displayable strings from the recordName and recordType fields of an address into a display area in the letter header. A messaging system does not interpret information in the header's display area. If no such display area exists, the MSAM can append the displayable strings to the body of the letter and note that the letter was also sent to that address.

An MSAM can add an actual address for which it is not responsible instead of the displayable strings from the recordName and recordType fields of the address. To do this, it must know the address format specified by a given extension type and how an

address of that type is stored in an `OCERecipient` structure. Knowing this, the MSAM can translate the extension value into an actual address. (Apple does not define the syntax and semantics for non-AOCE address extension types. MSAM developers must work together to define agreed-upon extension types, and the associated address syntax and semantics.)

Suppose, for example, an AppleLink MSAM knows how an SMTP address is stored in an `OCERecipient` structure. If an AOCE user sends a letter to an AppleLink address and to an SMTP address, the AppleLink MSAM can translate the SMTP address to its proper SMTP form and add it to the letter header as a display address.

Remember that an MSAM only delivers a letter to those recipients for which it is responsible. All other recipient information with the letter is for display purposes only, regardless of whether the other recipient information is included in actual address format or as displayable strings, and regardless of where the information is stored (a display area in the letter header or the body of the letter).

Note

Given that an MSAM routes a letter only to those recipients for which it is responsible, a recipient on the MSAM's messaging system cannot necessarily reply to all other recipients. An MSAM must consider what to do when a recipient wants to reply to addresses that the MSAM cannot reach. Regardless of how it handles this situation, the MSAM should avoid sending the AOCE user a reply that looks as if it went to all recipients of the original message if in fact it did not.   u

Although an MSAM is limited by the characteristics of the messaging system to which it is connected, it should always attempt to represent all recipients of an outgoing letter that it translates and transmits.

Translating to an AOCE Address

When an MSAM receives a message from its external messaging system, it must translate the addresses associated with the message before it can deliver the message to an AOCE system.

The function `DoConvertToAOCEAddress` in Listing 2-15 on page 2-90 provides an example of building an AOCE `OCERecipient` address structure from a non-AOCE address. The `DoConvertToAOCEAddress` function takes an address from a letter it received from its SMTP system and puts that address into AOCE format. The `DoConvertToAOCEAddress` function calls several AOCE utility routines to facilitate the process of constructing an AOCE address; the utility routines are described in the chapter "AOCE Utilities" in *Inside Macintosh: AOCE Application Interfaces*.

Listing 2-15 picks up at the point where `DoConvertToAOCEAddress` begins assembling the pieces of an `OCERecipient` structure. The `DoConvertToAOCEAddress` function begins by constructing the record ID part of the `OCERecipient`. A record ID, in turn, consists of a local record ID and record location information. It makes an `RLI` structure that contains the record location information by calling the AOCE utility routine `OCENewRLI` and providing it with an `RLI` structure's component parts: a catalog name, a discriminator, a dNode number, and a path. The `OCENewRLI` function returns

the `RLI` structure. The MSAM retrieves the catalog name from the private slot specification structure (type `MySlotSpec`) that the MSAM builds when it is launched. Because dNode numbers and paths are not used with non-AOCE addresses, `DoConvertToAOCEAddress` passes `OCENewRLI` a null dNode number and a `nil` pointer to a path. After `OCENewRLI` returns the `RLI` structure, `DoConvertToAOCEAddress` calls the AOCE utility routine `OCEValidRLI` to check its validity.

Next, `DoConvertToAOCEAddress` calls the `OCEPackRLI` utility routine to convert the `RLI` structure into packed form and calls the `OCEValidPackedRLI` utility routine to check the validity of the packed form.

Having prepared the record location information, `DoConvertToAOCEAddress` next prepares the local record ID, which consists of a creation ID, a record name, and a record type. A creation ID is not used in a non-AOCE address, so `DoConvertToAOCEAddress` calls the `OCESetCreationIDtoNull` utility routine to set the `CreationID` structure to 0. The buffer pointed to by the local variable `realName` contains a displayable form of the sender or receiver's name in C string format. The `DoConvertToAOCEAddress` function converts the C string into an `RString` and stores the `RString` in the local variable `recordName`. It tells the `OCECToRString` utility routine what character set the string uses and how many bytes, at maximum, it should place in the data portion of the `RString`, which in this example is the maximum number of bytes. Then `DoConvertToAOCEAddress` calls the `OCECToRString` utility routine again to get an `RString` that contains the sender or receiver's type. In this example, the type is always set to the constant `kUserRecTypeBody`, indicating a user.

At this point, `DoConvertToAOCEAddress` calls the `OCENewLocalRecordID` utility routine to build a local record ID from the creation ID, record name, and record type. The `DoConvertToAOCEAddress` function then calls the `OCENewRecordID` utility routine to build a record ID from its packed RLI and local record ID.

At last, `DoConvertToAOCEAddress` is ready to build the `OCERecipient` itself. It sets the `entitySpecifier` field to point to the record ID it has just constructed. Then it sets the extension fields. It specifies its extension type in the `extensionType` field. The buffer pointed to by the local variable `startAddr` contains the SMTP address in C string format. The `DoConvertToAOCEAddress` function converts the C string into an `RString` and stores the `RString` in the local variable `xtnValueRString`. (The `DoConvertToAOCEAddress` function converts the extension value from C string to `RString` format so that the mailer can correctly display the SMTP address to the user.) Then, `DoConvertToAOCEAddress` sets the `extensionSize` field to the number of bytes in the `body` field of `xtnValueRString` plus 4 more to account for the `dataLength` and `charSet` fields in an `RString` structure. This produces a count of the total number of bytes in `xtnValueRString`. Last, `DoConvertToAOCEAddress` sets the `extensionValue` field to point to `xtnValueRString`.

Before writing the address to a disk file, `DoConvertToAOCEAddress` converts the address into packed form. It calls the `OCEPackedDSSpecSize` utility routine, passing it the unpacked structure. In response, `OCEPackedDSSpecSize` returns the size of the packed structure into which the unpacked structure could be converted. Then `DoConvertToAOCEAddress` calls the `OCEPackDSSpec` utility routine and passes the

size value to it. Finally, `DoConvertToAOCEAddress` **writes the packed structure to a
disk file.**

Listing 2-15   Building an `OCERecipient` structure

```
OSErr DoConvertToAOCEAddress(FSSpec *myTempFileSpec, MySlotSpec *slotSpec)
{
   #define kMySMTPAddrType     'SMTP'
   #define kMyDirectoryType    'SMTP'
   #define kMyDiscriminator    {kMyDirectoryType,0L}

   OSErr             myErr;
   char              *startAddr, *realName;
   RLI               myRLI;
   PackedRLI         myPackedRLI;
   DirDiscriminator  discriminator = kMyDiscriminator;

   CreationID        cid;
   RString           recordName,recordType;
   LocalRecordID     localRID;
   RecordID          RID;

   OCERecipient      theRecipient;
   char              packedRecipient[kMaxRecipSize];
   unsigned long     packedRecipLength;
   RString           xtnValueRString;


   /*
      Not shown here -- parse the address information in the letter from the
      external messaging system. Put the SMTP address into a buffer pointed
      to by startAddr. Put the displayable string that identifies the sender
      or receiver into a buffer pointed to by realName.
   */

   /* make an RLI and check it for validity */
   OCENewRLI(&myRLI, (DirectoryNamePtr)&slotSpec->directoryName,
                     &discriminator, kNULLDNodeNumber, nil);
   if (!OCEValidRLI(&myRLI))
      return kUnexpectedOCECondition;

   /* pack the RLI and check it for validity */
   myErr = OCEPackRLI(&myRLI, &myPackedRLI, kRLIMaxBytes);
```

```
if (myErr != noErr)
    return myErr;
if (!OCEValidPackedRLI(&myPackedRLI))
    return kUnexpectedOCECondition;

/* prepare name and type rstrings and creation ID for local RID */
OCESetCreationIDtoNull(&cid);                /* set cid to null */
OCECToRString(realName, smRoman, &recordName, kRStringMaxBytes);
OCECToRString(kUserRecTypeBody, smRoman, &recordType, kRStringMaxBytes);

/* the components have been prepared; make the local RID and the RID */
OCENewLocalRecordID (&recordName, &recordType, &cid, &localRID);
OCENewRecordID(&myPackedRLI, &localRID, &RID);

/* build the OCERecipient address structure */
theRecipient.entitySpecifier  = &RID;
theRecipient.extensionType     = kMySMTPAddrType;
OCECToRString(startAddr, smRoman, &xtnValueRString, kRStringMaxChars);
theRecipient.extensionSize     = xtnValueRString.length+4;
theRecipient.extensionValue    = (Ptr)&xtnValueRString;

/* pack the OCERecipient and write it to a disk file */
packedRecipLength = OCEPackedDSSpecSize(&theRecipient);
OCEPackDSSpec(&theRecipient, (PackedDSSpec *)&packedRecipient,
              packedRecipLength);
myErr = DoWriteAddressToFile(myTempFileSpec, (Ptr)&packedRecipient,
                             packedRecipLength);
}
```

Note

If a personal MSAM receives an incoming letter that contains more than
one AOCE recipient, the MSAM translates all of the addresses. However,
a personal MSAM cannot forward letters from the user's Macintosh to
other AOCE users. A personal MSAM can deliver an incoming letter
only to the owner of the local Macintosh computer, even if the letter
contains the addresses of other AOCE users. u

Logging Personal MSAM Operational Errors

When an operational error occurs, such as a modem not functioning properly or an
access number being out of service, the personal MSAM should log the error by calling
the PMSAMLogError function.

You can log four general classes of information: informational messages, warnings,
errors that are not correctable by the user, and errors that are correctable by the user.

These classes are referred to as *error types*; they are represented by four enumerated constants. You use one of these constants in the `errorType` field of the `MailErrorLogEntryInfo` structure when you log an error:

```
enum {
    kMailELECorrectable     = 0,  /* error correctable by user */
    kMailELEError           = 1,  /* error not correctable by user */
    kMailELEWarning         = 2,  /* warning requiring no user intervention */
    kMailELEInformational   = 3   /* informational message */
};
```

For example, you would log an error of type `kMailELEInformational` if you wanted to inform the user that it took 12 connection attempts before a connection with the external messaging system was actually achieved. If you wanted to warn the user that his or her password on the external messaging system was about to expire, you would log an error of type `kMailELEWarning`. You use the `kMailELEError` error type to log an error that cannot be fixed by the user, for example, a missing resource in the personal MSAM. If an error occurs that requires user intervention, you log an error of type `kMailELECorrectable`.

In general, you should log all errors that require user intervention, but you should be selective about logging other types of errors. Logging many warnings and informational messages can fill the error log and cause problems at the user interface.

An error may apply to a specific slot or to the personal MSAM as a whole. When you log an error, you set the `msamSlotID` field of the `MailErrorLogEntryInfo` structure to 0 if the error applies to the personal MSAM as a whole. Otherwise, you set it to the slot ID of the affected slot.

When you log an error of type `kMailELECorrectable`, the IPM Manager considers either the personal MSAM or the affected slot to be suspended. While a 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, the function 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.

For example, suppose a user configures an SMTP personal MSAM to start up every night at midnight. At midnight, the IPM Manager launches the MSAM, and the MSAM fails to connect to its external messaging system because MacTCP, which is required for this MSAM, is not installed. The MSAM should log an error of type `kMailELECorrectable`. The IPM Manager will not try to launch the SMTP personal MSAM again until the user has installed MacTCP.

Because logging an error of type `kMailELECorrectable` implies that the problem is not transient in nature, the `PMSAMLogError` function does not provide you with a mechanism for canceling these errors or accessing logged entries. Correctable errors, by their definition, require a user's attention, and you should not log them unless absolutely necessary.

AOCE software defines the following error codes:

```
enum {                   /* predefined values of MailLogErrorCode */
   kMailMSAMErrorCode   = 0,     /* MSAM-defined error */
   kMailMiscError       = -1,    /* miscellaneous error */
   kMailNoModem         = -2     /* modem required, but missing */
};
```

Because a personal MSAM is a background application, it has no user interface and therefore cannot notify the user of runtime errors. Because each MSAM can potentially encounter errors specific to its implementation, the Finder cannot adequately notify the user of these errors without help from the MSAM. To solve this problem, an MSAM needs to provide two 'STR#' string list resources. The first 'STR#' resource contains a list of the MSAM's error messages, each describing a problem that may occur. This resource must have a resource ID of kMailMSAMErrorStringListID. The second 'STR#' resource contains a list of strings specifying the action that the user can take to fix a specific error. It must have a resource ID of kMailMSAMActionStringListID.

To cause the Finder to display one of your error messages, you must set the errorCode field of the MailErrorLogEntryInfo structure to kMailMSAMErrorCode and set the errorResource field. The errorResource field is an index into the list of your error messages in the 'STR#' resource. The index of the first message in the string list is 1.

When you log an error that requires user intervention (kMailELECorrectable), you must specify an action that the user should take to correct the error. You provide the action messages in a 'STR#' resource (resource ID = kMailMSAMActionStringListID). You set the actionResource field to an index into the list of your action messages in the 'STR#' resource. The index of the first message in the string list is 1.

The Finder displays all errors to the user, regardless of the error type. A user reports that an error is corrected by clicking the Resolve button on a problem report in his or her In Tray. (See the *PowerTalk User's Guide* for a description of the PowerTalk user interface.)

The IPM Manager reinstates a suspended personal MSAM or slot when the user reports 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 user reports that the problem has been corrected, the IPM Manager launches it. If the personal MSAM is running, it gets a kMailEPPCContinue high-level event.

Messaging Service Access Module Reference

This section describes the structures and functions that constitute the messaging service access module API. It also includes descriptions of the high-level events an MSAM might receive.

Data Types and Constants

This section describes the data structures in the MSAM API. The chapters "AOCE
Utilities" and "Interprogram Messaging Manager" in *Inside Macintosh: AOCE Application
Interfaces* contain descriptions of other structures that you use.

The MSAM Parameter Block

Every function in the MSAM API takes a pointer to an MSAMParam parameter block as
input. The parameter block has a standard header followed by function-specific fields.
Each function description in the section "MSAM Functions" describes the fields of that
function's parameter block.

MailParamBlockHeader

The parameter block header for an MSAMParam structure has the following definition:

```
# define MailParamBlockHeader
   Ptr            qLink;         /* reserved */\
   long           reservedH1;    /* reserved */\
   long           reservedH2;    /* reserved */\
   ProcPtr        ioCompletion;  /* your completion routine */\
   OSErr          ioResult;      /* result code */\
   long           saveA5;        /* location of app global variables */\
   short          reqCode;       /* reserved */
```

Field descriptions

qLink	Reserved.
reservedH1	Reserved.
reservedH2	Reserved.
ioCompletion	Pointer to a completion routine that you can provide. When a function that you called asynchronously completes execution, it calls your completion routine. See page 2-219 for a description of the completion routine. Set this field to nil if you do not wish to provide a completion routine. This field is ignored if you call a function synchronously.
ioResult	The result of a function. You can poll the ioResult field to determine when a function has finished executing. When you execute the function asynchronously, the function sets this field to 1 as soon as the function has been queued for execution. When the function completes execution, it sets this field to the actual result code.
saveA5	The contents of your application's A5 register.
reqCode	Reserved.

MSAMParam

The `MSAMParam` structure is a union of function-specific substructures, each of which contains standard header fields.

```
union MSAMParam{
    struct {MailParamBlockHeader} header;

    PMSAMGetMSAMRecordPB        pmsamGetMSAMRecord;
    PMSAMOpenQueuesPB           pmsamOpenQueues;
    PMSAMSetStatusPB            pmsamSetStatus;
    PMSAMLogErrorPB             pmsamLogError;
    SMSAMSetupPB                smsamSetup;
    SMSAMStartupPB              smsamStartup;
    SMSAMShutdownPB             smsamShutdown;
    MSAMEnumeratePB             msamEnumerate;
    MSAMDeletePB                msamDelete;
    MSAMOpenPB                  msamOpen;
    MSAMOpenNestedPB            msamOpenNested;
    MSAMClosePB                 msamClose;
    MSAMGetMsgHeaderPB          msamGetMsgHeader;
    MSAMGetAttributesPB         msamGetAttributes;
    MSAMGetRecipientsPB         msamGetRecipients;
    MSAMGetContentPB            msamGetContent;
    MSAMGetEnclosurePB          msamGetEnclosure;
    MSAMEnumerateBlocksPB       msamEnumerateBlocks;
    MSAMGetBlockPB              msamGetBlock;
    MSAMMarkRecipientsPB        msamMarkRecipients;
    MSAMnMarkRecipientsPB       msamnMarkRecipients;
    MSAMCreatePB                msamCreate;
    MSAMBeginNestedPB           msamBeginNested;
    MSAMEndNestedPB             msamEndNested;
    MSAMSubmitPB                msamSubmit;
    MSAMPutMsgHeaderPB          msamPutMsgHeader;
    MSAMPutAttributePB          msamPutAttribute;
    MSAMPutRecipientPB          msamPutRecipient;
    MSAMPutContentPB            msamPutContent;
    MSAMPutEnclosurePB          msamPutEnclosure;
    MSAMPutBlockPB              msamPutBlock;
    MSAMCreateReportPB          msamCreateReport;
    MSAMPutRecipientReportPB    msamPutRecipientReport;
    PMSAMCreateMsgSummaryPB     pmsamCreateMsgSummary;
    PMSAMPutMsgSummaryPB        pmsamPutMsgSummary;
    PMSAMGetMsgSummaryPB        pmsamGetMsgSummary;
```

```
   MailWakeupPMSAMPB          wakeupPMSAM;
   MailCreateMailSlotPB       createMailSlot;
   MailModifyMailSlotPB       modifyMailSlot;
};

typedef union MSAMParam MSAMParam;
```

The Mail Buffer

You use the `MailBuffer` structure to pass data between your MSAM and the
IPM Manager.

MailBuffer

The mail buffer structure is defined by the `MailBuffer` data type.

```
struct MailBuffer {
    long     bufferSize; /* size of your buffer */
    Ptr      buffer;     /* pointer to your buffer  */
    long     dataSize;   /* amount of data returned in or read out
                            of your buffer */
};

typedef struct MailBuffer MailBuffer;
```

Field descriptions

`bufferSize`   When reading, you set this field to the size of your buffer in bytes.
                When writing, you set this field to the number of bytes that you
                want to write.

`buffer`   A pointer to your buffer. You allocate a buffer of whatever size
                you need.

`dataSize`   When it successfully completes execution, the function sets this
                field to the actual number of bytes that it read or wrote.

The Mail Reply Structure

A `MailReply` structure is a model. Many functions in the MSAM API format the data
they place in a `MailBuffer` structure according to the `MailReply` model format.

MailReply

A structure of type `MailReply` consists of a single field, `tupleCount`, that contains a count. It is followed immediately by `tupleCount` occurrences of a data item or structure. The format of the data item or structure depends on the particular function that returns the data in the `MailReply` structure format. For instance, the `MSAMEnumerate` function returns `MSAMEnumerateOutQReply` or `MSAMEnumerateInQReply` structures.

```
struct MailReply {
   unsigned short tupleCount;
   /* tuple[tupleCount] */
};

typedef struct MailReply MailReply;
```

The Enumeration Structures

The enumeration structures, `MSAMEnumerateOutQReply` and `MSAMEnumerateInQReply`, return information about messages in an outgoing or incoming queue, respectively. The `MSAMEnumerate` function returns a list of one or the other of these structures. Each structure gives enough information about a message for you to know what to do next with the message.

MSAMEnumerateOutQReply

When a personal or server MSAM calls the `MSAMEnumerate` function to enumerate an outgoing queue, the function returns information about the messages in the outgoing queue in a list of `MSAMEnumerateOutQReply` structures, one for each message.

```
struct MSAMEnumerateOutQReply {
   long            seqNum;     /* sequence number of message */
   Boolean         done;       /* resolution of message */
   IPMPriority     priority;   /* priority of message */
   OSType          msgFamily;  /* message family */
   long            approxSize; /* size of message */
   Boolean         tunnelForm; /* reserved */
   Byte            padByte;    /* pad to even byte boundary */
   NetworkSpec     nextHop;    /* reserved */
   OCECreatorType  msgType;    /* message creator and type */
};

typedef struct MSAMEnumerateOutQReply MSAMEnumerateOutQReply;
```

Field descriptions

seqNum   A sequence number that identifies a specific message in the
                    outgoing queue. It is valid until you delete the message. You
                    pass this value to the MSAMOpen function to identify a message
                    you want to open.

done   A Boolean value that indicates if you have sent—or completed your
                    attempts to send—the message to each of the recipients for which
                    you are responsible. The IPM Manager sets this field to true when
                    you have finished sending or attempting to send the message to all
                    of the recipients for which you are responsible. You tell the IPM
                    Manager which recipients you have processed by calling the
                    MSAMnMarkRecipients function.

priority   A value that indicates the priority with which the message was sent.
                    Possible values are: kIPMNormalPriority, kIPMLowPriority,
                    and kIPMHighPriority.

msgFamily   A value that indicates the message family to which the message
                    belongs. The AOCE-defined message families are kMailFamily,
                    kMailFamilyFile, and kIPMFamilyUnspecified. Developers
                    can define other message families.

approxSize   The size of the message itself, not including some overhead bytes
                    associated with the message when it resides in the outgoing queue.

tunnelForm   Reserved.

nextHop   Reserved.

msgType   A structure that specifies the creator and type of the message. The
                    creator field indicates the creator of the message. The type field
                    identifies the type of message.

MSAMEnumerateInQReply

When a personal MSAM calls the MSAMEnumerate function to enumerate an incoming
queue, the function returns information about the letters in the queue in a list of
MSAMEnumerateInQReply structures, one for each letter.

```
struct MSAMEnumerateInQReply {
    long     seqNum;     /* letter sequence number */
    Boolean  msgDeleted; /* should letter be deleted? */
    Boolean  msgUpdated; /* was message summary updated? */
    Boolean  msgCached;  /* is letter in the incoming queue? */
    Byte     padByte;    /* pad to even byte boundary */
};

typedef struct MSAMEnumerateInQReply MSAMEnumerateInQReply;
```

Field descriptions

seqNum   A sequence number for a specific letter in the incoming queue. It is valid until you delete the letter.

msgDeleted   A Boolean value that indicates whether you should delete the letter. Only the IPM Manager sets and clears this field. If this field is set to true, you should delete the letter.

msgUpdated   A Boolean value that indicates if the IPM Manager has updated the message summary associated with the letter. Only the IPM Manager sets and clears this field. This field is set to true if the IPM Manager has updated the message summary.

msgCached   A Boolean value that indicates if the letter is attached to its message summary. Only the IPM Manager sets and clears this field. This field is set to true if you wrote the letter into the incoming queue.

The Mail Time Structure

The MailTime structure appears in the sendTimeStamp attribute in a letter's header and in the sendTime field of a letter's message summary.

MailTime

The MailTime structure is the standard structure for reporting time in an AOCE system.

```
struct MailTime {
    UTCTime     time;    /* current UTC(GMT) */
    UTCOffset   offset;  /* offset from UTC */
};

typedef struct MailTime MailTime;
```

Field descriptions

time   Current time expressed as universal coordinated time (UTC) in seconds since 00:00 hours, January 1, 1904. (The UTCTime data type is unsigned long.)

offset   Offset from UTC in seconds. The offset is a signed value added to the time value. (The UTCOffset data type is long.)

The Letter Attribute Structures

Letter attributes identify a letter and indicate who wrote it, when it was sent, what its priority for delivery is, who the recipients are, and so forth. Most attributes are stored in the letter header; a few are stored in the message summary.

MailAttributeID

When calling the `MSAMPutAttribute` or `MSAMPutRecipient` function, you use the `MailAttributeID` data type to indicate the letter attribute whose value you are passing to the function. When calling the `MSAMGetRecipients` function, you use it to indicate the recipient type about which you want information.

```
typedef unsigned short MailAttributeID;
```

A variable of type `MailAttributeID` may have any of the following values:

```
enum {
    kMailLetterFlagsBit     = 1,  /* letter flags bit */
    kMailIndicationsBit     = 3,  /* indications bit */
    kMailMsgTypeBit         = 4,  /* letter creator & type bit */
    kMailLetterIDBit        = 5,  /* letter ID bit */
    kMailSendTimeStampBit   = 6,  /* send timestamp bit */
    kMailNestingLevelBit    = 7,  /* nesting level bit */
    kMailMsgFamilyBit       = 8,  /* message family bit */
    kMailReplyIDBit         = 9,  /* reply ID bit */
    kMailConversationIDBit  = 10, /* conversation ID bit */
    kMailSubjectBit         = 11, /* subject bit */
    kMailFromBit            = 12, /* From recipient bit */
    kMailToBit              = 13, /* To recipient bit */
    kMailCcBit              = 14, /* cc recipient bit */
    kMailBccBit             = 15  /* bcc recipient bit */
};
```

MailAttributeBitmap

When calling the `MSAMGetAttributes` function, you use a `MailAttributeBitmap` structure to indicate the letter attributes about which you want information. Each defined bit in the attribute bitmap represents a letter attribute. This structure is also a component part of the `MSAMMsgSummary` structure.

```
struct MailAttributeBitmap {
    unsigned int            /* 32 bits */
        reservedA:16,       /* bits 17 to 32--reserved */
        reservedB:1,        /* bit 16--reserved */
        bcc:1,              /* bit 15--blind carbon copy recipients */
        cc:1,               /* bit 14--carbon copy recipients */
        to:1,               /* bit 13--To recipients */
        from:1,             /* bit 12--sender of letter */
```

```
      subject:1,              /* bit 11--subject of letter */
      conversationID:1,       /* bit 10--ID of conversation thread */
      replyID:1,              /* bit 09--ID of letter being replied to */
      msgFamily:1,            /* bit 08--message family */
      nestingLevel:1,         /* bit 07--nesting level of letter */
      sendTimeStamp:1,        /* bit 06--time letter was sent */
      letterID:1;             /* bit 05--letter's unique ID number */
      msgType:1,              /* bit 04--letter's creator and type */
      indications:1,          /* bit 03--indications */
      reservedC:1,            /* bit 02--reserved */
      letterFlags:1           /* bit 01--letter flags */
};

typedef struct MailAttributeBitmap MailAttributeBitmap;
```

Field descriptions

`bcc`	Secondary recipients whose addresses do not appear on the letter as received by the To and cc recipients and other bcc recipients.
`cc`	Recipients who are being sent a courtesy copy of the letter.
`to`	Primary recipients of the letter.
`from`	The sender of the letter.
`subject`	The subject of the letter.
`conversationID`	The letter ID number of the original letter that began a sequence of replies or forwards that resulted in the current letter.
`replyID`	The letter ID number of the letter to which the current letter is a reply.
`msgFamily`	A value that indicates the message family to which the message belongs.
`nestingLevel`	The nesting level of the letter. A letter that is newly created (that is, not a reply to or forward of an existing letter) has a nesting level of 0. A reply to or forward of a letter whose nesting level is 0 has a nesting level of 1. A reply to or forward of a letter whose nesting level is 1 has a nesting level of 2, and so on. See the section "Letters" beginning on page 2-17 for information on nested letters.
`sendTimeStamp`	The time the letter was sent.
`letterID`	The letter ID number for the letter. This number is generated by the IPM Manager.
`msgType`	The creator and type of the letter. Each letter has a creator and type.
`indications`	Indications of the properties of the letter, such as whether the letter contains a digital signature, whether the originator requested non-delivery reports, and so on. The `MailIndications` structure is described on page 2-102.
`letterFlags`	Flags that indicate the status of the letter, such as whether it has been opened by the user. The `MailLetterFlags` structure is described on page 2-123. Server MSAMs should ignore this attribute.

The following table summarizes letter attributes. In the column headed "O/M", an *M* indicates *mandatory*—that is, this attribute must always be present. An *O* means *optional*— the attribute may or may not be present in a letter. In the column headed "F/V", an *F* indicates *fixed*—that is, this attribute has a fixed size—while a *V* means *variable*—the attribute size is variable.

Constant	Value	Attribute data type	O/M	F/V
kMailLetterFlagsBit	1	MailLetterFlags	M	F
kMailIndicationsBit	3	MailIndications	M	F
kMailMsgTypeBit	4	OCECreatorType	M	F
kMailLetterIDBit	5	MailLetterID	M	F
kMailSendTimeStampBit	6	MailTime	M	F
kMailNestingLevelBit	7	MailNestingLevel	M	F
kMailMsgFamilyBit	8	OSType	M	F
kMailReplyIDBit	9	MailLetterID	O	F
kMailConversationIDBit	10	MailLetterID	O	F
kMailSubjectBit	11	RString	O	V
kMailFromBit	12	OCERecipient	M	V
kMailToBit	13	OCERecipient	M	V
kMailCcBit	14	OCERecipient	O	V
kMailBccBit	15	OCERecipient	O	V

An MSAM should allocate the largest possible buffer for attributes whose size is variable.

Note

All letter attributes except the `letterFlags` attribute are stored in the letter header. Both personal and server MSAMs read or set all letter attributes in the letter header. The `letterFlags` attribute is stored in a letter's message summary. Server MSAMs do not create message summaries and therefore do not set or read a `letterFlags` attribute for letters they handle. The `letterFlags` attribute applies only to letters submitted by a personal MSAM. u

MailIndications

The `MailIndications` structure further defines the letter attribute called `indications`. It is a bit field structure that contains information about several characteristics of the letter, such as what priority level the originator set for the letter, whether it has been sent, what type of reports the originator wants, and so on. An MSAM sets many of these bits for an incoming letter and reads the bits for an outgoing letter.

The following constants define bits in the `MailIndications` structure:

```
enum {
    kMailOriginalInReportBit     = 1,
    kMailNonReceiptReportsBit    = 3,
    kMailReceiptReportsBit       = 4,
    kMailForwardedBit            = 5,
    kMailPriorityBit             = 6,
    kMailIsReportWithOriginalBit = 8,
    kMailIsReportBit             = 9,
    kMailHasContentBit           = 10,
    kMailHasSignatureBit         = 11,
    kMailAuthenticatedBit        = 12,
    kMailSentBit                 = 13
};
```

Note

Constants for the `hasStandardContent`, `hasImageContent`, and `hasNativeContent` bit fields are not defined. u

```
struct MailIndications {
unsigned int
   reservedB:16,
   hasStandardContent:1,/* letter has a content block */
   hasImageContent:1,   /* letter has an image block */
   hasNativeContent:1,  /* letter has a content enclosure */
   sent:1,              /* letter sent, not just composed */
   authenticated:1,     /* letter was created and transported with
                           authentication */
   hasSignature:1,      /* letter was signed with a digital signature */
   hasContent:1,        /* this letter or a nested letter has content */
   isReport:1,          /* not a letter, is really a report */
   isReportWithOriginal:1,/* report contains the original letter */
   priority:2,          /* letter has normal, low, or high priority */
   forwarded:1,         /* letter contains a forwarded letter */
   receiptReports:1,    /* originator requests delivery indications */
   nonReceiptReports:1, /* originator requests non-delivery indications */
   originalInReport:2,  /* originator wants original letter enclosed in
                           reports */
};

typedef struct MailIndications MailIndications;
```

Field descriptions

hasStandardContent

If this bit is set, this letter has a block of type `kMailContentType` that contains data in standard interchange format.

hasImageContent

If this bit is set, this letter has a block of type `kMailImageBodyType` that contains data in standard image format.

hasNativeContent

If this bit is set, this letter contains content in the form of a content enclosure.

sent

If this bit is set, this letter was sent, not just composed. This bit is clear for nested letters and those that exist on disk and have not yet been submitted.

authenticated

If this bit is set, this letter was created by an authenticated user and transported over a secure path using the Apple Secure Data Stream Protocol. In release 1, a letter entering an AOCE system via an MSAM is not authenticated. This bit will always be set to 0 on letters read by a personal MSAM. On letters read by a server MSAM, the bit may be set or clear. In either case, it is for the MSAM's information only.

hasSignature

If this bit is set, the sender signed the letter with a digital signature. The signature applies to the letter as a whole. If a portion of the letter is signed, the bit is not set. See the chapter "Digital Signature Manager" in *Inside Macintosh: AOCE Application Interfaces* for information about digital signatures. The AOCE software sets this bit to 0 for letters submitted by an MSAM. If this bit is set for an outgoing letter, the MSAM can ignore it or add a note to the letter indicating that the letter was originally signed with a digital signature.

hasContent

If this bit is set, this letter, or a letter nested within it, contains content. The content can be a content block, an image block, or a content enclosure. Although this bit doesn't indicate the type of content or the nesting level at which the content exists, it provides useful information to AOCE letter applications that display letter content by indicating if a letter has some type of content at some nesting level.

isReport

If this bit is set, this is an IPM report. Because an IPM report is not a report that an MSAM creates or receives, you never set this bit for a report that you create, nor will it be set on a report that you receive. For more information about reports, see the section "Reports" on page 2-23. IPM reports are discussed in the chapter "Interprogram Messaging Manager" in *Inside Macintosh: AOCE Application Interfaces.*

isReportWithOriginal

If this bit is set, this is an IPM report that contains the original letter
to which the report pertains. Because an IPM report is not a report
that an MSAM creates or receives, you never set this bit for a report
that you create, nor will it be set on a report that you receive. For
more information about reports, see the section "Reports" on
page 2-23. IPM reports are discussed in the chapter "Interprogram
Messaging Manager" in *Inside Macintosh: AOCE Application
Interfaces.*

priority   The priority of the letter, as set by the sender. This 2-bit field can be
set to any of the following values: `kIPMNormalPriority`,
`kIPMLowPriority`, or `kIPMHighPriority`.

```
enum {
    kIPMAnyPriority    = 0,/* not used by MSAM */
    kIPMNormalPriority = 1,
    kIPMLowPriority,
    kIPMHighPriority
};
```

It is up to the recipient to decide how to handle letters of
different priorities.

forwarded   If this bit is set, this letter is a forwarded letter.

receiptReports If this bit is set, the originator of this letter has requested a report
containing delivery indications.

nonReceiptReports

If this bit is set, the originator of this letter has requested a report
containing non-delivery indications.

originalInReport

This 2-bit field can be set to either of the following values:

```
enum {
    kMailNoOriginal        = 0,
    kMailEncloseOnNonReceipt= 3
};
```

If this field is set to `kMailNoOriginal`, the originator of this letter
specified that the original letter not be enclosed in reports. If this
field is set to `kMailEnclosedOnNonReceipt`, the originator of
this letter specified that the original letter be enclosed in reports
containing non-delivery indications. An MSAM ignores this field
and never includes a copy of the original letter in a report it creates.
The AOCE toolbox is responsible for including originals when
appropriate.

The following table indicates who sets the bits in the `MailIndications` structure for an incoming letter. In the column labeled "Responsible for setting," *MSAM* refers to both personal and server MSAMs.

MailIndications bit field	Responsible for setting
`hasStandardContent`	MSAM
`hasImageContent`	MSAM
`hasNativeContent`	MSAM
`sent`	IPM Manager
`authenticated`	IPM Manager
`hasSignature`	IPM Manager
`hasContent`	MSAM
`isReport`	Not applicable
`isReportWithOriginal`	Not applicable
`priority`	MSAM
`forwarded`	MSAM
`receiptReports`	MSAM
`nonReceiptReports`	MSAM
`originalInReport`	MSAM

The Recipient Structures

The structures in this section define the sender or receiver of a message. You use these structures when you get recipient information from a message that you have opened or when you put recipient information into a message that you are creating. The chapter "Interprogram Messaging Manager" in *Inside Macintosh: AOCE Application Interfaces* also describes the `OCERecipient` and `OCEPackedRecipient` structures. The structures are described here from the perspective of an MSAM's use of them.

OCERecipient

The `OCERecipient` structure completely specifies an address. It should contain whatever information is needed to deliver a message to that address.

You use an `OCERecipient` structure to specify a reply address when you call the `MSAMPutMsgHeader` function.

An `OCERecipient` structure is the unpacked form of the `OCEPackedRecipient` structure (described next). The utility routines `OCEPackRecipient` and `OCEUnpackRecipient` allow you to transform the address information from one format to the other. The routines are described in the chapter "Interprogram Messaging Manager" in *Inside Macintosh: AOCE Application Interfaces*.

```
struct OCERecipient {
   RecordID*      entitySpecifier;
   OSType         extensionType;
   unsigned short extensionSize;
   Ptr            extensionValue;
};
```

Field descriptions

entitySpecifier

Pointer to a `RecordID` structure. The record ID contains part of the address. The section "AOCE Addresses" beginning on page 2-23 explains what each field of the `RecordID` structure should contain when it holds either an AOCE address or an external address.

extensionType Identifies the type of messaging system with which this recipient is associated. It determines the format and the meaning of the data pointed to by the `extensionValue` field. You must provide an extension type.

extensionSize The number of bytes in the `extensionValue` field.

extensionValue A pointer to the part of the address that is specific to the messaging system. You should provide the address extension information in an `RString` structure. This allows the information to be displayed properly to the user and allows the user to create new addresses of this type using the type-in addressing feature. (Type-in addressing is a feature of PowerTalk software's human interface.)

Table 2-5 on page 2-30 and Table 2-4 on page 2-29 list the contents of each field in an `OCERecipient` structure for an AOCE address and an external address, respectively.

```
typedef OCERecipient MailRecipient;
```

The `MailRecipient` structure is defined as an `OCERecipient` data type. You use it in exactly the same way as you would an `OCERecipient` structure. You provide a `MailRecipient` structure to specify a recipient of a letter or a report when you call the `MSAMPutRecipient` or `MSAMCreateReport` function, respectively.

OCEPackedRecipient

An `OCEPackedRecipient` structure is the packed form of the `OCERecipient` structure (described in the previous section).

You cannot read the packed address directly. Before you can read it, you must convert it to the unpacked format using the `OCEUnpackRecipient` utility routine. The utility routines `OCESizePackedRecipient`, `OCEGetRecipientType`, and `OCESetRecipientType` allow you to manipulate an `OCEPackedRecipient` structure. They are described in the chapter "Interprogram Messaging Manager" in *Inside Macintosh: AOCE Application Interfaces*.

A structure of type `OCEPackedRecipient` is a minimum-sized structure and should not be allocated on the stack. Instead, use the `NewPtr` or `NewHandle` routine to allocate the structure.

```
struct OCEPackedRecipient {
    unsigned short    dataLength;    /* length of recipient data */
    Byte              data[kOCEPackedRecipientMaxBytes];
};
```

Field descriptions

dataLength   Length of the packed recipient address that immediately follows this field.

data   Packed recipient address.

MailOriginalRecipient

The `MailOriginalRecipient` structure consists of a single field, `index`, that contains an index value for a given recipient. The `MailOriginalRecipient` structure is a model of how address information is stored in a buffer. It is always followed immediately by an `OCEPackedRecipient` structure that contains the address information of that recipient. The `MSAMGetRecipients` function returns recipient information in `MailOriginalRecipient` format when you call the function requesting information about recipients of a particular type (From, To, cc, or bcc).

```
struct MailOriginalRecipient {
    short    index;        /* index for recipient */
             /* followed by OCEPackedRecipient structure */
};

typedef struct MailOriginalRecipient MailOriginalRecipient;
```

Field descriptions

index   An absolute index value associated with the recipient.

MailResolvedRecipient

The `MailResolvedRecipient` structure contains an index value for the recipient, an indication of whether the recipient is a bcc recipient, and a Boolean value that indicates whether you are responsible for delivering the message to this recipient. The `MailResolvedRecipient` structure is a model of how address information is stored in a buffer. The fields of the structure are always followed immediately by an

`OCEPackedRecipient` structure that contains the address information of the recipient. The `MSAMGetRecipients` function returns recipient information in `MailResolvedRecipient` format when you call the function requesting information about resolved recipients.

```
struct MailResolvedRecipient {
    short        index;          /* index for recipient */
    short        recipientFlags;/* recipient information */
    Boolean      responsible;    /* responsible for delivery? */
    Byte         padByte;
                 /* followed by OCEPackedRecipient structure */
};

typedef struct MailResolvedRecipient MailResolvedRecipient;
```

Field descriptions

`index`	An absolute index value associated with the recipient. You need this value when you call the `MSAMPutRecipientReport` function to identify the recipient to whom the report pertains. The index is also useful if you want to match an original recipient with a resolved recipient.
`recipientFlags`	A value that tells you if this recipient is a bcc recipient. Use the mask `kIPMBCCRecMask` to determine if this recipient is a bcc recipient.
`responsible`	A Boolean value that is set to `true` if you are responsible for sending the message to this recipient.

The Segment Types

A content block (type `kMailContentType`) contains the body or main content of a letter in standard interchange format (see the section "Letters" beginning on page 2-17 for more information about interchange format). A content block consists of segments of data in plain text, styled text, picture, sound, or movie format. The `MailSegmentType` data type identifies one of the five standard data segment types. The `MailSegmentMask` data type specifies one or more of these segment types. You read and write content blocks with the `MSAMGetContent` (page 2-150) and `MSAMPutContent` functions (page 2-186).

MailSegmentType

A variable of the `MailSegmentType` data type specifies the format of data in a data segment.

```
typedef unsigned short MailSegmentType;
```

A variable of type `MailSegmentType` can contain one of the following values:

```
enum {        /* values of MailSegmentType */
   kMailInvalidSegmentType   = 0,
   kMailTextSegmentType      = 1,
   kMailPictSegmentType      = 2,
   kMailSoundSegmentType     = 3,
   kMailStyledTextSegmentType = 4,
   kMailMovieSegmentType     = 5
};
```

Constant descriptions

kMailInvalidSegmentType

> This value is included as a convenience. An MSAM can initialize a variable of type `MailSegmentType` to this known value before calling the `MSAMGetContent` function.

kMailTextSegmentType

> The segment contains plain text in one or more character sets. The text data must consist of 1-byte or 2-byte character codes, depending on the character set (Roman, Arabic, Kanji, and so on).

kMailPictSegmentType

> The segment contains picture data in PICT format. For more information about PICT format, see *Inside Macintosh: Imaging With QuickDraw.*

kMailSoundSegmentType

> The segment contains data in Audio Interchange File Format (AIFF). For more information about AIFF format, see *Inside Macintosh: More Macintosh Toolbox.*

kMailStyledTextSegmentType

> The segment contains text and a `StScrpRec` structure containing the style information corresponding to that text. The text data consists of 1-byte or 2-byte character codes, depending on the character set (Roman, Arabic, Kanji, and so on). For more information on the `StScrpRec` structure, the style record, and the style table, see *Inside Macintosh: Text.*

kMailMovieSegmentType

> The segment contains QuickTime movie data in QuickTime movie file format ('MooV'). For more information about the 'MooV' file format, see *Inside Macintosh: QuickTime.*

MailSegmentMask

You use the `MailSegmentMask` data type to indicate the kinds of data segments that you want to read when you call the `MSAMGetContent` function.

```
typedef unsigned short MailSegmentMask;
```

The bits in the segment mask are defined as follows:

```
enum {
    kMailTextSegmentBit,
    kMailPictSegmentBit,
    kMailSoundSegmentBit,
    kMailStyledTextSegmentBit,
    kMailMovieSegmentBit
};
```

You can use a combination of the following values to set bits in the segment mask:

```
enum {           /* values of MailSegmentMask */
    kMailTextSegmentMask       = 1L<<kMailTextSegmentBit,
    kMailPictSegmentMask       = 1L<<kMailPictSegmentBit,
    kMailSoundSegmentMask      = 1L<<kMailSoundSegmentBit,
    kMailStyledTextSegmentMask = 1L<<kMailStyledTextSegmentBit,
    kMailMovieSegmentMask      = 1L<<kMailMovieSegmentBit
};
```

The Enclosure Information Structure

You add an enclosure to a letter by calling the MSAMPutEnclosure function. The function takes a MailEnclosureInfo structure as input. This structure describes the enclosure being added to the letter.

MailEnclosureInfo

You pass a MailEnclosureInfo structure to the MSAMPutEnclosure function when you enclose a file that resides in memory.

```
struct MailEnclosureInfo {
    StringPtr   enclosureName;
                              /* name of the enclosure */
    CInfoPBPtr  catInfo;    /* HFS catalog info about enclosure*/
    StringPtr   comment;    /* comment for Get Info window */
    Ptr         icon        /* icon for enclosure file */
};

typedef struct MailEnclosureInfo MailEnclosureInfo;
```

Field descriptions

enclosureName   A pointer to the name of the file that you want to enclose. Format
                   the filename as a Pascal-style string—that is, add a leading length
                   byte. The name must be 1 to 31 bytes long, excluding the length
                   byte, and must not contain colons (:).

catInfo   A pointer to a fully specified CInfoPBRec structure (defined in
                   Inside Macintosh: Files), which is returned by the PBGetCatInfo
                   function. Set the fields for which you cannot obtain appropriate
                   values to **0**, with the exception of the ioNamePtr and
                   ioFlFndrInfo fields. Ignore the ioNamePtr field because you
                   pass the filename in the enclosureName field. The first 8 bytes of
                   the ioFlFndrInfo field contain values for the file's type and
                   creator. Because the type and creator determine the application
                   associated with the file and the icon that the Finder displays for that
                   file, omitting a value for the ioFlFndrInfo field renders the file
                   unusable. Therefore, you should make every attempt to provide
                   meaningful values for the file's creator and type. If you do not
                   know the application associated with the file, set the creator field
                   to four question marks ('????'). If you do not know the file's type,
                   set the type field to ('????') as well.

comment   A pointer to a Pascal-style string containing the file's comment; it is
                   the information that the Get Info command in the Finder displays
                   for the file. The string cannot be longer than 199 characters,
                   excluding the length byte. The Finder truncates a longer string
                   when it places the file on an HFS volume. If the file has no
                   comment, set the comment field to nil.

icon   A pointer to the file's icon: the standard black-and-white icon (32 by
                   32 bits) consisting of 128 bytes of bitmap followed by 128 bytes of
                   mask. Enclosures in a letter are stored in AppleSingle format.
                   AppleSingle format typically provides a single black-and-white
                   icon so that non-Macintosh file systems can easily read an icon
                   without needing to know how to get at the icon resources stored
                   in AppleSingle format. This field preserves compatibility with
                   AppleSingle format. It is not used by AOCE software. You can set
                   this field to nil.

The Image Block Information Structure

You use the TPfPgDir structure when reading or writing an image block.

TPfPgDir

An image block starts with an image block information structure (the `TPfPgDir` data type defined by the Printing Manager), followed by a series of PICT elements.

```
struct TPfPgDir{
    short iPages;        /* number of pages in image block */
    long  iPgPos[129];   /* array [0..iPfMaxPgs] of offsets */
};
```

Field descriptions

iPages   The number of pages in the image. The image block contains one
                PICT for each page.

iPgPos   An array of offsets from the start of the block to the picture elements
                that follow the `TPfPgDir` structure.

The `iPgPos` array contains offsets to the picture elements that follow the `TPfPgDir` structure. The offset from the start of the image block to the image of page $n + 1$ is `iPgPos`[n] (because page numbers start at 1 and the array elements start at 0). The array contains `iPgPos`[$n + 1$] elements for a document of n pages. The last element is the offset of the end of the last page from the beginning of the block. You can determine the size of a page by subtracting the offset of the current page from the offset of the next page, that is, the size of page n is `iPgPos`[n] – `iPgPos`[$n – 1$].

The High-Level Event Structures

The `MailEPPCMsg`, `SMCA`, `OCESetupLocation`, `MailLocationFlags`, and `MailLocationInfo` structures are used in conjunction with high-level events.

MailEPPCMsg

When you call the `AcceptHighLevelEvent` function after receiving an AOCE high-level event, the function returns a buffer that contains a `MailEPPCMsg` structure.

```
struct MailEPPCMsg {
    short     version;                   /* message version */
    union {
        SMCA *    theSMCA;               /* pointer to SMCA */
        long      sequenceNumber;        /* letter sequence number */
        MailLocationInfo locationInfo;/* location information */
    } u;
};

typedef struct MailEPPCMsg MailEPPCMsg;
```

Field descriptions

version   The version number of the AOCE high-level event. You should
                 verify that this version number matches the value of the
                 kMailEPPCMsgVersion constant in the PowerTalk interface files
                 you used when you built your MSAM.

u.theSMCA   A pointer to an SMCA structure that contains additional information
                 relevant to the event. The IPM Manager uses this field when it
                 sends any of the following events: kMailEPPCCreateSlot,
                 kMailEPPCModifySlot, kMailEPPCDeleteSlot,
                 kMailEPPCMsgOpened, kMailEPPCSendImmediate,
                 kMailEPPCAdmin.

u.sequenceNumber
                 The sequence number of the letter to which the event applies.
                 The IPM Manager uses this field when it sends either the
                 kMailEPPCInQUpdate or kMailEPPCDeleteOutQMsg event.

u.locationInfo   A MailLocationInfo structure. The IPM Manager uses this field
                 when it sends the kMailEPPCLocationChanged event.

SMCA

The shared memory communication area, defined by the SMCA structure, is used to pass
information between the IPM Manager and an MSAM, in addition to the data passed in
the EventRecord structure.

```
struct SMCA {
    unsigned short smcaLength; /* length of entire SMCA
                                    (including the length field) */
    OSErr          result;     /* result code */
    long           userBytes;  /* event-specific data */
    union{
       CreationID  slotCID;    /* creation ID of record
                                    containing slot information */
       long        msgHint;    /* message reference value */
    } u;
};

typedef struct SMCA SMCA;
```

Field descriptions

smcaLength   The total length of the SMCA structure, including the 2 bytes for the
                 smcaLength field itself. The IPM Manager sets this field.

result   You set this field to acknowledge receipt of the event to the IPM
                 Manager or to indicate that you have handled the event. Set it to the
                 noErr result code to acknowledge receipt of the event or to report
                 success. Otherwise, set it to an MSAM-defined error code. See the
                 individual event descriptions for details.

`userBytes`	The interpretation of this field is dependent on the particular event that is being processed. See the individual event descriptions for information on how this field is used for that event.
`u.slotCID`	If the event applies to a particular slot, this field contains the creation ID of the slot's record in the Setup catalog. If the event applies to the MSAM as a whole, this field contains 0. The IPM Manager sets this field. It is irrelevant to server MSAMs.
`u.msgHint`	A reference value associated with a specific letter. The IPM Manager sets this field.

OCESetupLocation

The `OCESetupLocation` data type defines the current system location.

```
typedef char OCESetupLocation;
```

The values 0–8 are valid values for a variable of type `OCESetupLocation`. Values 1–8 refer to an actual location. The value 0 is a special case that indicates the offline or disconnected state. When the current system location is 0, a personal MSAM should not be executing.

The following enumeration defines constants for two of the valid values of type `OCESetupLocation`:

```
enum {
   kOCESetupLocationNone   = 0,  /* disconnect state */
   kOCESetupLocationMax    = 8   /* maximum location value */
};
```

MailLocationFlags

The `MailLocationFlags` data type defines a bit array. Each bit corresponds to a system location. If the bit is set, the slot to which the location flags apply is active at that location. The `MailLocationFlags` data type is used in the `MailLocationInfo` and `MailStandardSlotInfoAttribute` structures.

```
typedef unsigned char MailLocationFlags;
```

A system location is identified by a value ranging from 1 to 8. To test a bit in a variable of type `MailLocationFlags`, the following mask is defined:

```
#define MailLocationMask(locationNumber) (1<<((locationNumber)-1))
```

Note that for the special location value 0, which corresponds to the disconnected or offline state, the mask value is 0. The slot is inactive at all locations when the current system location is 0.

MailLocationInfo

The `MailLocationInfo` structure contains the current system location and a bit array defining the locations at which a given slot is active. The `MailLocationInfo` structure is part of the `MailEPPCMsg` structure. A personal MSAM receives a `MailLocationInfo` structure when it receives a `kMailEPPCLocationChanged` event.

```
struct MailLocationInfo {
    OCESetupLocation    location;    /* the current location */
    MailLocationFlags   active;      /* slot's location flags */
};

typedef struct MailLocationInfo MailLocationInfo;
```

Field descriptions

location   A value that identifies the current system location. It may contain any integer value between 0–8.

active   A bit array that defines whether or not a given slot is active at each system location.

The Server MSAM Administrative Event Structures

The IPM Manager provides a server MSAM with administrative information by means of the `kMailEPPCAdmin` high-level event (page 2-235).

SMSAMAdminCode

The `SMSAMAdminCode` data type defines a set of codes for server MSAM administrative actions.

```
typedef unsigned short SMSAMAdminCode;
```

A variable of type `SMSAMAdminCode` can have any of the following values:

```
enum {
    kSMSAMNotifyFwdrSetupChange= 1,
    kSMSAMNotifyFwdrNameChange = 2,
    kSMSAMNotifyFwdrPwdChange  = 3,
    kSMSAMGetDynamicFwdrParams = 4
};
```

SMSAMAdminEPPCRequest

The `userBytes` field of the `SMCA` structure associated with a `kMailEPPCAdmin` high-level event provides a pointer to an `SMSAMAdminEPPCRequest` structure. The `SMSAMAdminEPPCRequest` structure contains an administrative code followed by data whose type is determined by the code.

```
struct SMSAMAdminEPPCRequest {
   SMSAMAdminCode       adminCode;          /* admin code */
   union {
      SMSAMSetupChange     setupChange;     /* setup change */
      SMSAMNameChange      nameChange;      /* reserved */
      SMSAMPasswordChange  passwordChange;  /* reserved */
      SMSAMDynamicParams   dynamicParams;   /* reserved */
   } u;
};

typedef struct SMSAMAdminEPPCRequest SMSAMAdminEPPCRequest;
```

Field descriptions

adminCode   A value that indicates the type of administrative action requested by the `kMailEPPCAdmin` high-level event. The value in this field determines the type of structure contained in the `u` field. In release 1 of PowerTalk system software, this should always be the `kSMSAMNotifyFwdrSetupChange` code.

u   Contains a structure that varies depending on the value of the `adminCode` field. In release 1 of PowerTalk system software, this should always be an `SMSAMSetupChange` structure.

SMSAMSetupChange

The `SMSAMSetupChange` structure contains connectivity information about a server MSAM.

```
struct SMSAMSetupChange {
   SMSAMSlotChanges  whatChanged;  /* what parameters changed */
   AddrBlock         serverHint;   /* AOCE server address */
};

typedef struct SMSAMSetupChange SMSAMSetupChange;
```

Field descriptions

whatChanged   A value that indicates the connectivity information that has changed.

serverHint   The AppleTalk address of the PowerShare catalog server that the
                    MSAM should use to read its Forwarder record containing the
                    changed connectivity information. Because an AOCE system is a
                    distributed system, the changed data may not have propagated to
                    other servers yet.

SMSAMSlotChanges

The `SMSAMSlotChanges` data type defines a bit array that indicates the kind of
connectivity information that has changed.

```
typedef unsigned long SMSAMSlotChanges;
```

The bits in the `SMSAMSlotChanges` data type are defined as follows:

```
enum {
    kSMSAMFwdrHomeInternetChangedBit,
    kSMSAMFwdrConnectedToChangedBit,
    kSMSAMFwdrForeignRLIsChangedBit,
    kSMSAMFwdrMnMServerChangedBit
};
```

You can use the following values to test the bits in a variable of type
`SMSAMSlotChanges`:

```
enum {            /* values of SMSAMSlotChanges */
   kSMSAMFwdrEverythingChangedMask  = -1,
   kSMSAMFwdrHomeInternetChangedMask= 1L<<kSMSAMFwdrHomeInternetChangedBit,
   kSMSAMFwdrConnectedToChangedMask = 1L<<kSMSAMFwdrConnectedToChangedBit,
   kSMSAMFwdrForeignRLIsChangedMask = 1L<<kSMSAMFwdrForeignRLIsChangedBit,
   kSMSAMFwdrMnMServerChangedMask   = 1L<<kSMSAMFwdrMnMServerChangedBit
};
```

Constant descriptions

`kSMSAMFwdrEverythingChangedMask`
                    In release 1 of the AOCE software, this constant has the same
                    definition as that of the `kSMSAMFwdrForeignRLIsChangedMask`
                    constant.

`kSMSAMFwdrHomeInternetChangedMask`
                    Reserved.

`kSMSAMFwdrConnectedToChangedMask`
                    Reserved.

kSMSAMFwdrForeignRLIsChangedMask
> The record location information that points to a catalog associated with the MSAM's external messaging system has changed. The information changes when the PowerShare system administrator adds or deletes a catalog for a messaging system served by the MSAM.

kSMSAMFwdrMnMServerChangedMask
> Reserved.

The Personal MSAM Setup Structures

The `MailTimer` and `MailTimerKind` data types and the `MailTimers` and `MailStandardSlotInfoAttribute` structures contain the user's send and receive requirements for a given slot and location information for that slot.

MailTimer

A variable of type `MailTimer` specifies a number of seconds. The value is interpreted as a frequency interval or a specific time, depending on which union field is used.

```
union MailTimer {
    long  frequency;    /* how often to connect */
    long  connectTime;  /* time since midnight */
};

typedef union MailTimer MailTimer;
```

Field descriptions

frequency
> A value that tells a personal MSAM how often it should connect to its messaging system to send or retrieve mail. The frequency interval is specified in seconds.

connectTime
> A value that tells a personal MSAM at what time it should connect to its messaging system to send or retrieve mail. The time is specified as the number of seconds since midnight. The midnight used is that of the internal time on the Macintosh as set by the user.

MailTimerKind

A variable of type `MailTimerKind` specifies the type of timer that a user wants to use with a given mail slot.

```
typedef Byte MailTimerKind;
```

A variable of type `MailTimerKind` can have any of the following values:

```
enum {
   kMailTimerOff       = 0,  /* no timer specified */
   kMailTimerTime      = 1,  /* timer relative to midnight */
   kMailTimerFrequency = 2   /* frequency timer*/
};
```

Constant descriptions

kMailTimerOff   Specifies that the user has not requested a timer.

kMailTimerTime   Specifies that a personal MSAM should send or retrieve messages at
                 a particular time.

kMailTimerFrequency
                 Specifies that a personal MSAM should send or retrieve messages at
                 regular intervals.

MailTimers

The `MailTimers` structure indicates how frequently a personal MSAM connects to its
external messaging system. A personal MSAM's setup template sets the fields of the
`MailTimers` structure in response to user actions. The user can express the frequency as
a particular clock time at which the personal MSAM automatically connects every day
(for example, connect at 3:00 A.M. to send and receive letters) or as a periodic occurrence
(for example, connect every two hours). The IPM Manager uses the information in this
structure to determine when it should send a `kMailEPPCSchedule` event to the
personal MSAM.

```
struct MailTimers {
   MailTimerKind  sendTimeKind;     /* timer kind for sending */
   MailTimerKind  receiveTimeKind;  /* timer kind for receiving */
   MailTimer      send;             /* connect time or frequency
                                       for sending letters */
   MailTimer      receive;          /* connect time or frequency
                                       for receiving letters */
};

typedef struct MailTimers MailTimers;
```

Field descriptions

sendTimeKind   A constant that indicates what type of timer the user wants the
                personal MSAM to use for sending messages for a particular slot.
                The setup template sets this field to one of the following values:
                kMailTimerTime, kMailTimerFrequency, or kMailTimerOff.

receiveTimeKind

A constant that indicates what type of timer the user wants the personal MSAM to use for retrieving messages for a particular slot. The setup template sets this field to one of the following values: kMailTimerTime, kMailTimerFrequency, or kMailTimerOff.

send
A value that specifies either the time interval that elapses before the personal MSAM sends messages to its external messaging system or a specific time at which the MSAM sends these messages. The MSAM interprets this field according to the value in the sendTimeKind field. If that value is kMailTimerOff, the MSAM ignores this field.

receive
A value that specifies either the time interval that elapses before the personal MSAM retrieves messages from its external messaging system or a specific time at which the MSAM retrieves these messages. The MSAM interprets this field according to the value in the receiveTimeKind field. If that value is kMailTimerOff, the MSAM ignores this field.

MailStandardSlotInfoAttribute

The personal MSAM's setup template obtains location and timing information from the user to set the active and sendReceiveTimer fields of this structure appropriately. Then it adds the structure to the slot's Combined or Mail Service record in the Setup catalog, where the information is available to the IPM Manager.

```
struct MailStandardSlotInfoAttribute {
    short               version;        /* version of this slot structure */
    MailLocationFlags active;           /* active at location i if
                                            MailLocationMask(i) is set */
    Byte                padByte;
    MailTimers          sendReceiveTimer;
};

typedef struct MailStandardSlotInfoAttribute MailStandardSlotInfoAttribute;
```

Field descriptions

version
The version of the MailStandardSlotInfoAttribute structure. You should set this field to 1. There is no constant defined for it.

active
A bit array that defines whether or not the slot is active at a given location. If the bit is set, the slot is active at the corresponding location. A slot is active if a personal MSAM is able to send and receive messages for the slot.

sendReceiveTimer

The frequency at which the IPM Manager should schedule the personal MSAM to send and receive messages for the user account represented by this slot. (The IPM Manager does this by sending the MSAM a kMailEPPCSchedule event.)

The Personal MSAM Letter Flag Structures

The letter flags provide information about a letter in an incoming queue. Only personal MSAMs use the structures in this section.

MailLetterSystemFlags

The IPM Manager sets the letter system flags.

```
typedef unsigned short   MailLetterSystemFlags;
```

The bit in the system flags bytes that you can test is defined as follows:

```
enum {
   kMailIsLocalBit = 2
   };
```

You can use the following value to test the bit flag in the `MailLetterSystemFlags` data type.

```
enum {
   kMailIsLocalMask         = 1L<<kMailIsLocalBit
};
```

Constant descriptions

`kMailIsLocalMask`
> The letter exists in an incoming queue on the local computer. If the `kMailIsLocalBit` bit is not set, the letter is stored on an external messaging system, and only its message summary is currently available locally.

MailLetterUserFlags

The IPM Manager and a personal MSAM can set letter user flags in response to a user action.

```
typedef unsigned short   MailLetterUserFlags;
```

The bits in the user flags bytes are defined as follows:

```
enum {
   kMailReadBit,
   kMailDontArchiveBit,
   kMailInTrashBit
   };
```

You can use the following values to test the flags in the `MailLetterUserFlags` data type.

```
enum {
    kMailReadMask            = 1L<<kMailReadBit,
    kMailDontArchiveMask     = 1L<<kMailDontArchiveBit,
    kMailInTrashMask         = 1L<<kMailInTrashBit
};
```

Constant descriptions

kMailReadMask   The user has opened this letter. A personal MSAM sets the letter user flags to 0 when it creates the letter's message summary. The IPM Manager sets the `kMailReadBit` bit to 1 when the user opens the letter. A personal MSAM can also modify this bit by calling the `PMSAMPutMsgSummary` function.

kMailDontArchiveMask
                 Reserved.

kMailInTrashMask
                 Reserved.

MailLetterFlags

The `MailLetterFlags` structure contains both system and user letter flags to indicate the status of a letter.

```
struct MailLetterFlags {
    MailLetterSystemFlags    sysFlags;    /* system flags */
    MailLetterUserFlags      userFlags;   /* user flags */
};

typedef struct MailLetterFlags MailLetterFlags;
```

Field descriptions

sysFlags   A set of bit flags managed by the IPM Manager. You can test the `kMailIsLocalBit` bit to determine if a given letter is actually stored on the local computer.

userFlags   A set of bit flags that indicate state changes that are controlled by the user. The only bit flag that is relevant to an MSAM is the `kMailReadBit` bit, which indicates whether the user has opened the letter. You can test this bit with the `kMailReadMask` constant.

MailMaskedLetterFlags

Use the `MailMaskedLetterFlags` structure to set the letter flags attribute in a letter. This structure is used by the `MSAMPutMsgSummary` function.

```
struct MailMaskedLetterFlags {
    MailLetterFlags   flagMask;   /* flags that are to be set */
    MailLetterFlags   flagValues; /* their values */
};

typedef struct MailMaskedLetterFlags MailMaskedLetterFlags;
```

Field descriptions

flagMask   The flags that are to be set.

flagValues   The values of the flags that you want to set.

The Personal MSAM Message Summary Structures

A personal MSAM creates a message summary to store summary information about a letter. The Finder uses message summary information to display incoming letters to the user. The `MSAMMsgSummary` structure defines a message summary. A message summary consists of a few individual fields and two groups of letter attributes. The two groups of letter attributes are defined by the `MailMasterData` and `MailCoreData` structures, described in this section.

MailMasterData

The attributes specified in the `MailMasterData` structure are not critical to the Finder when it displays information about the letter to which the message summary belongs.

```
struct MailMasterData {
   MailAttributeBitmap   attrMask;       /* indicates attributes present in
                                            letter */
   MailLetterID          messageID;      /* ID of this letter *
   MailLetterID          replyID;        /* ID of letter this is a reply to */
   MailLetterID          conversationID;/* ID of letter that started this
                                            conversation */
};

typedef struct MailMasterData MailMasterData;
```

Field descriptions

attrMask   A bit array that indicates letter attributes. You must set the bits that
                 correspond to the attributes that are present in the letter. See the
                 description of the MailAttributeBitmap structure on page 2-100
                 for a description of the bits in the attribute bitmap.

messageID   The letter ID of this letter. The letter ID is a value that uniquely
                 identifies the letter. The letter ID is provided by the IPM Manager.

replyID   The letter ID of the letter to which this letter is a reply. You provide
                 this value if it exists in the letter.

conversationID   The letter ID of the original letter that began a sequence of replies or
                 forwards that resulted in this letter. You provide this value if it
                 exists in the letter.

MailCoreData

The Finder uses the attributes specified in the MailCoreData structure when it displays
information about the letter to which the message summary belongs. You provide values
for the fields of the structure, except where otherwise noted in the field descriptions.

```
/* defines for the addressedToMe field */
#define kAddressedAs_TO 0x1
#define kAddressedAs_CC 0x2
#define kAddressedAs_BCC 0x4

struct MailCoreData {
   MailLetterFlags      letterFlags;   /* letter status flags */
   unsigned long        messageSize    /* size of letter */
   MailIndications      letterIndications;
                                       /* indications for this letter */
   OCECreatorType       messageType;   /* message creator and type of this
                                             letter */
   MailTime             sendTime;      /* time this letter was sent */
   OSType               messageFamily; /* message family */
   unsigned char        reserved;
   unsigned char        addressedToMe; /* user is To, cc, or bcc recipient */
   char                 agentInfo[6];  /* reserved (set to 0) */
   /* these are variable length and even padded */
   RString32            sender;        /* sender of this letter */
   RString32            subject;       /* subject of this letter */
};

typedef struct MailCoreData MailCoreData;
```

Field descriptions

letterFlags   A set of bit flags that indicate the status of the letter, such as
                           whether it has been opened by the user. Set this field to 0. See the
                           description of the `MailLetterFlags` structure on page 2-123 for
                           more information on these bit flags. You can modify the user
                           portion of the letter flags when you call the `PMSAMPutMsgSummary`
                           function.

messageSize   The size of the letter in bytes. You provide this value.

letterIndications
                           Indications of additional properties of the letter, such as whether
                           the letter contains a digital signature, whether or not the originator
                           requested non-delivery indications, and so on. See the description
                           of the `MailIndications` structure on page 2-102. You provide
                           this value.

messageType   The creator and type of the letter. Every letter has a creator and
                           type. You must provide this value.

sendTime   The time the letter was sent. You provide this value.

messageFamily   A value that indicates the message family to which the message
                           belongs. Set this field to `kMailFamily`.

reserved   Reserved.

addressedToMe   Indicates how the letter was sent to the addressee: as a To address, a
                           cc address, or a bcc address; possible values are `kAddressedAs_TO`,
                           `kAddressedAs_CC`, and `kAddressedAs_BCC`. You must set this
                           field appropriately. You can set more than one bit.

agentInfo   Reserved. Set this field to 0.

sender   The sender of the letter. You must provide a value for this field.
                           If your sender information consists of an odd number of bytes,
                           add a pad byte so that it ends on an even byte boundary. The IPM
                           Manager treats this field and the `subject` field that follows as a
                           single common buffer that contains variable-length sender and
                           subject information. See the section "Creating a Letter's Message
                           Summary" beginning on page 2-64 for information on how to
                           correctly assign a value to this field.

subject   The subject of the letter. You must provide this value. If your subject
                           information consists of an odd number of bytes, add a pad byte so
                           that it ends on an even byte boundary. The IPM Manager treats this
                           field and the `sender` field before it as a single common buffer that
                           contains variable-length sender and subject information. You add
                           the subject on the first even-byte boundary following the sender
                           information, which is not necessarily the same as the beginning of
                           this field. See the section "Creating a Letter's Message Summary"
                           beginning on page 2-64 for information on how to correctly assign a
                           value to this field.

MSAMMsgSummary

An `MSAMMsgSummary` structure provides summary information about an incoming letter. You must create one of these structures for each incoming letter. (In addition to the fields defined in the message summary structure, the IPM Manager stores up to `kMailMaxPMSAMMsgSummaryData` bytes of MSAM-specific private data with a message summary.)

```
struct MSAMMsgSummary {
   short                version;      /* version of the MSAMMsgSummary
                                         structure */
   Boolean              msgDeleted;   /* should letter be deleted? */
   Boolean              msgUpdated;   /* was message summary updated? */
   Boolean              msgCached;    /* is letter in the incoming queue? */
   Byte                 padByte;
   MailMasterData       masterData;   /* attributes not essential to
                                         display */
   MailCoreData         coreData;     /* attributes critical to display *
};

typedef struct MSAMMsgSummary MSAMMsgSummary;
```

Field descriptions

`version`	The version of the message summary structure. You must set this field to the constant `kMailMsgSummaryVersion`.
`msgDeleted`	A Boolean value indicating whether you should delete this letter. You do not provide a value for this field. The IPM Manager initially sets this field to `false`. It sets this field to `true` when the user deletes a letter. If this field is `true`, you should delete the letter on your external messaging system and delete the letter's message summary.
`msgUpdated`	A Boolean value indicating whether the IPM Manager updated information in the message summary. You do not provide an initial value for this field. The IPM Manager initially sets this field to `false`. It sets this field to `true` when it updates any of the following fields in the message summary: `msgDeleted`, `msgStoreFlags`, `finderInfo`. You read this field to determine if the message summary has changed. If it has, you should reexamine the message summary and take appropriate action, if any, based on the changed information. After taking the action, you should reset this field to `false`.
`msgCached`	A Boolean value indicating whether the letter associated with the message summary exists in an incoming queue. You do not provide a value for this field. The IPM Manager initially sets this field to `false`. It sets this field to `true` when you write the letter corresponding to this message summary into the incoming queue.

masterData   A `MailMasterData` structure that contains letter attributes not
                    essential to the ability of the Finder to display the letter. See the
                    structure description on page 2-124 for an explanation of the
                    information that you must provide.

coreData   A `MailCoreData` structure that contains the attributes crucial to
                    the Finder's ability to display the letter. See the structure
                    description on page 2-125 for an explanation of the information that
                    you must provide.

The Personal MSAM Error Log Entry Structure

The error log is where a personal MSAM can report errors that require a user's
intervention to correct. The personal MSAM reports errors using the `PMSAMLogError`
function. The function takes a pointer to a `MailErrorLogEntryInfo` structure
as input.

MailErrorLogEntryInfo

You provide a `MailErrorLogEntryInfo` structure to the `PMSAMLogError` function
when you want to report an operational error to the IPM Manager and ultimately to
the user.

```
typedef unsigned short MailLogErrorType;

/* values of MailLogErrorType */
enum {
   kMailELECorrectable  = 0,  /* error correctable by user */
   kMailELEError        = 1,  /* error not correctable by user */
   kMailELEWarning      = 2,  /* warning requiring no user intervention */
   kMailELEInformational= 3   /* informational message */
};

typedef short MailLogErrorCode;

/* predefined values of MailLogErrorCode */
enum {
   kMailMSAMErrorCode   = 0,     /* MSAM-defined error */
   kMailMiscError       = -1,    /* miscellaneous error */
   kMailNoModem         = -2     /* modem required, but missing */
};

struct MailErrorLogEntryInfo {
   short              version;            /* log entry version */
   UTCTime            timeOccurred;       /* time of error */
   Str31              reportingPMSAM;     /* MSAM reporting the error */
   Str31              reportingMSAMSlot;  /* slot having the error */
```

```
   MailLogErrorType   errorType;             /* level of error */
   MailLogErrorCode   errorCode;             /* error code */
   short              errorResource;         /* error string resource index */
   short              actionResource;        /* action string resource index */
   unsigned long      filler;                /* reserved */
   unsigned short     filler2;               /* reserved */
};

typedef struct MailErrorLogEntryInfo MailErrorLogEntryInfo;
```

Field descriptions

version   The version of the error log entry. Set this field to
                kMailErrorLogEntryVersion.

timeOccurred   The time that the error occurred. This is filled in by the
                IPM Manager.

reportingPMSAM

                A string identifying the personal MSAM that is logging the error.
                This is filled in by the IPM Manager.

reportingMSAMSlot

                A string identifying the slot that is experiencing the error, if the
                error is associated with a specific slot. This is filled in by the IPM
                Manager.

errorType   A value that indicates the type of error that you are logging. Set this
                field to one of the following constants: kMailELECorrectable,
                kMailELEError, kMailELEWarning,
                kMailELEInformational.

errorCode   A value that indicates the error you are logging. There are three
                predefined errors; you can define others. If you want to log an error
                that you define, set this field to kMailMSAMErrorCode and set the
                errorResource field to the index into your string list ('STR#')
                resource for the string that describes the error. The constants for the
                predefined errors are kMailMSAMErrorCode, kMailMiscError,
                and kMailNoModem.

errorResource   An index into your list of error messages. An error message
                describes the problem that has occurred. The resource ID of the
                'STR#' resource containing the list of error messages must be
                kMailMSAMErrorStringListID. If you are logging an
                AOCE-defined error, the IPM Manager ignores this field.

actionResource The index into your list of action messages. An action message is
                always associated with an error of type kMailELECorrectable.
                The action message recommends the action that the user should
                take to correct the error. The resource ID of the 'STR#' resource
                containing the list of action messages must be
                kMailMSAMActionStringListID. If you are logging an
                AOCE-defined error, the IPM Manager ignores this field.

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

MSAM Functions

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

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

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

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

Most functions in the MSAM API have the following form:

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

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

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

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

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

Listing 2-16   Calling an MSAM function from assembly language

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

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

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

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

IMPORTANT

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

Initializing an MSAM

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

PMSAMGetMSAMRecord

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

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

`paramBlock` Pointer to a parameter block.

Parameter block

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

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

Field descriptions

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

DESCRIPTION

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

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

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

IMPORTANT

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

SPECIAL CONSIDERATIONS

This function is always executed synchronously.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0506

RESULT CODES

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

SEE ALSO

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

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

PMSAMOpenQueues

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

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

paramBlock   Pointer to a parameter block.

Parameter block

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

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

Field descriptions

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

DESCRIPTION

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

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

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

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

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

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

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

IMPORTANT

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

SPECIAL CONSIDERATIONS

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0500

RESULT CODES

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

SEE ALSO

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

SMSAMSetup

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

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

`paramBlock` Pointer to a parameter block.

Parameter block

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

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

Field descriptions

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

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

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

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

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

DESCRIPTION

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

SPECIAL CONSIDERATIONS

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0523

RESULT CODES

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

SEE ALSO

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

The `SMSAMStartup` function is described next.

SMSAMStartup

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

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

`paramBlock` Pointer to a parameter block.

Parameter block

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

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

Field descriptions

`msamIdentity`   The server MSAM's authentication identity. You obtain this identity from the `AuthBindSpecificIdentity` function.

`queueRef`   A value that identifies the outgoing queue for the server MSAM that you specify.

DESCRIPTION

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

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

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

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0501

RESULT CODES

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

SEE ALSO

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

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

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

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

Enumerating Messages in a Queue

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

MSAMEnumerate

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

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

paramBlock   Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

queueRef   The value that identifies the queue about which you want information. A personal MSAM specifies either the outgoing queue reference or the incoming queue reference that it obtained from the `PMSAMOpenQueues` function, depending on which queue it wants to enumerate. A server MSAM specifies the outgoing queue reference that it obtained from the `SMSAMStartup` function.

startSeqNum   The sequence number of the message in the queue at which you want the `MSAMEnumerate` function to start the enumeration. Set this field to 1 to begin the enumeration with the first message in the queue. When you call the function and there is insufficient space in your buffer to hold information about all of the remaining messages in the queue, the function returns in the `nextSeqNum` field the sequence number of the next message. Use that number in the `startSeqNum` field the next time you call the function.

nextSeqNum   The sequence number of the first message in the queue whose
              information did not fit into your buffer. The function sets this field
              when your buffer is too small to hold all the information you
              requested. To continue the enumeration, call the MSAMEnumerate
              function again and set the startSeqNum field to the current value
              of the nextSeqNum field. The MSAMEnumerate function sets the
              nextSeqNum field to 0 when it has returned information about all
              of the messages in the queue.

buffer   A MailBuffer structure. You set the value of the bufferSize
              field in the MailBuffer structure to the number of bytes in
              your buffer. Because the number of messages in the queue
              varies, use your best estimate to choose the size of the buffer.
              The MSAMEnumerate function retrieves information about the
              messages in the queue that you specify and writes it into your
              buffer, the buffer field. It sets the value of the dataSize field to
              the number of bytes of data it placed in the buffer.

DESCRIPTION

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

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

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

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0503

RESULT CODES

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

SEE ALSO

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

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

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

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

Opening an Outgoing Message

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

MSAMOpen

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

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

paramBlock   Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

queueRef
: The queue reference of the queue that contains the message you want to open. For a personal MSAM, specify the outgoing queue reference you obtained from the PMSAMOpenQueues function. For a server MSAM, specify the queue reference you obtained from the SMSAMStartup function.

seqNum
: The sequence number that identifies the message you want to open. You get this number from the seqNum field in the MSAMEnumerateOutQReply structure returned by the MSAMEnumerate function.

mailMsgRef
: A message reference number that identifies the opened message. The MSAMOpen function returns a reference number for the message that you use in subsequent function calls to read the message.

DESCRIPTION

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

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0508

RESULT CODES

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

SEE ALSO

The MSAMEnumerateOutQReply structure is described on page 2-97.

The PMSAMOpenQueues function is described on page 2-133.

The SMSAMStartup function is described on page 2-136.

The MSAMClose function is described on page 2-167.

The MSAMDelete function is described on page 2-202.

Reading Header Information

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

MSAMGetAttributes

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

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

paramBlock Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

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

requestMask A bit field structure that specifies which attributes in the letter's header you want to read. The attributes whose values you may retrieve with this function are listed below. Set the bit for each attribute that you want to read. Clear the remaining bits.

buffer   A `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. The `MSAMGetAttributes` function writes attribute values into your buffer (the `buffer` field) and sets the value of the `dataSize` field to the number of bytes of data it placed in the buffer.

responseMask   A bit field structure that specifies the attributes for which the
                MSAMGetAttributes function returned values in the buffer. If the
                function did not return an attribute because either a requested
                attribute does not exist in the letter or you did not request the
                attribute, the function sets the corresponding bit in the structure to 0.

more   A Boolean value that indicates whether there are more attribute
                values than can fit in your buffer. If your buffer is too small
                to hold all of the attribute values that you requested, the
                MSAMGetAttributes function sets this field to true; otherwise,
                it sets this field to false. If the value of the more field is true,
                you can call the MSAMGetAttributes function again, setting the
                bits in the request mask for the attributes you did not yet receive.

DESCRIPTION

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

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

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

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

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

SPECIAL CONSIDERATIONS

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

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$050B

RESULT CODES

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

SEE ALSO

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

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

The `MSAMGetRecipients` function is described next.

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

MSAMGetRecipients

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

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

paramBlock   Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

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

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

DESCRIPTION

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

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

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

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

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

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

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

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

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

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

SPECIAL CONSIDERATIONS

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

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$050C

RESULT CODES

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

SEE ALSO

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

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

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

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

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

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

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

MSAMGetMsgHeader

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

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

paramBlock   Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

mailMsgRef   A reference number that identifies the message for which you want
             header information. You obtain the reference number when you call
             the `MSAMOpen` function.

selector   A constant that indicates the type of header information that you
             are requesting. The possible values are defined below. You cannot
             add or combine constant values in the `selector` field.

offset   The byte position, relative to the beginning of the header
             information specified in the `selector` field, from which you want
             the `MSAMGetMsgHeader` function to begin reading. To read from
             the beginning of the header information field, set this field to 0. If
             your buffer is too small to hold all of the data you requested, you
             can call the `MSAMGetMsgHeader` function again and compute a
             new value for the `offset` field using the `dataSize` value that the
             function returns in the `MailBuffer` structure.

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

DESCRIPTION

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

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

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

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0511

RESULT CODES

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

SEE ALSO

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

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

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

The MailBuffer structure is described on page 2-96.

Reading a Message

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

MSAMGetContent

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

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

paramBlock   Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

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

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

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

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

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

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

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

script | A value that indicates the character set (Roman, Arabic, Kanji, etc.) of the text that the function placed in your buffer. The function sets this field only when it returns text data (it sets the `segmentType` field to `kMailTextSegmentType` or `kMailStyledTextSegmentType`).

segmentType | A constant that indicates the type of the current data segment. A segment can contain text, pictures, sound, QuickTime movies, or styled text. The constants that the function may return in this field are described on page 2-109. (If you are reading data from the segment and you need to call the `MSAMGetContent` function more than once to retrieve all of the data from the segment, the function returns a value in this field only the first time you call it for that segment.)

endOfScript | A Boolean value that indicates whether the text in your buffer is the end of a script run. The function sets this flag only when it returns data from a plain text or styled text segment. If there is more text in the current script run, it sets this field to `false`.

endOfSegment | A Boolean value that indicates whether the `MSAMGetContent` function has reached the end of a segment. If you did not request the current segment type in your segment mask, the function always sets this field to `true`. If you requested the current segment type in your segment mask, the function sets this field to `true` if it has returned all of the data in the current segment and to `false` if there is more data in the current segment.

endOfContent | A Boolean value that indicates whether the `MSAMGetContent` function has reached the end of the letter's content block. The `MSAMGetContent` function sets the `endOfContent` field to `true` when it reaches the end of the last segment in the content block; otherwise it sets this field to `false`.

segmentLength | The number of bytes in the current segment. The `MSAMGetContent` function returns a value in this field the first time you call it for a given segment.

segmentID | A segment identifier. This is both an input and an output. Set this field to 0 the first time you call it for a given letter. The function returns a value in this field the first time it reads each segment in a letter. On subsequent calls to the function, you set it to 0 or to a known segment ID. If you set it to 0, the function continues reading sequentially the current segment (or if `endOfSegment` is set to `true`, the next segment). If you set it to a segment ID, the function reads the segment specified by the segment ID.

DESCRIPTION

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

SPECIAL CONSIDERATIONS

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

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

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$050D

RESULT CODES

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

SEE ALSO

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

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

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

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

MSAMGetEnclosure

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

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

paramBlock   Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

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

contentEnclosure

A Boolean value that indicates whether the enclosure is the content enclosure for the letter. A content enclosure contains the content of a letter. It is typically a file in an application's native format. When you call the `MSAMGetEnclosure` function the first time, it sets this field to `true` if the enclosure is a content enclosure or to `false` if it is not. The function also sets the value of this field the first time you call it after the function sets the `endOfFile` flag to `true`. At other times, consider the value of this field invalid.

buffer   A `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. The `MSAMGetEnclosure` function writes the information that you request into your buffer and sets the value of the `dataSize` field to the number of bytes of data it placed in the buffer.

endOfFile   A Boolean value that indicates whether an entire enclosure file has been read. If your buffer is not large enough to hold the entire enclosure file, the `MSAMGetEnclosure` function sets the `endOfFile` field to `false`. You can call the function repeatedly until it sets the `endOfFile` field to `true`, at which point an entire enclosure file has been read. The `MSAMGetEnclosure` function does not put data belonging to more than one enclosure file into your buffer at the same time, even when the end of file is reached on one enclosure file, there are additional enclosure files to read, and your buffer is not full.

When a letter has no enclosures, the function sets this field to `false`. To detect the no-enclosure condition, test only the `endOfEnclosures` field.

endOfEnclosures   A Boolean value that indicates whether the `MSAMGetEnclosure` function has reached the end of all of the enclosures for the letter that you specify. When the `MSAMGetEnclosure` function has retrieved all enclosures for the current nesting level, it sets the `endOfEnclosures` field to `true`.

DESCRIPTION

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

A letter's enclosures can be folders or Macintosh files in AppleSingle stream format. The `MSAMGetEnclosure` function returns all of the files to you; it does not return any folder

information.That is, you do not know how the files might have been organized and stored in the letter.

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

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$050E

RESULT CODES

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

SEE ALSO

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

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

MSAMEnumerateBlocks

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

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

paramBlock   Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

mailMsgRef   A reference number that identifies the message whose message blocks you want to enumerate. You obtain the reference number when you call the `MSAMOpen` function.

startIndex   The sequence number of the block about which you want information. Set this field to 1 to start with the first message block. When you call the function and there is insufficient space in your buffer to hold information about all of the remaining blocks, the function returns in the `nextIndex` field the sequence number of the next block. Use that number in the `startIndex` field the next time you call the function.

buffer   A `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. The `MSAMEnumerateBlocks` function places data in your buffer in the form of a `MailReply` structure. The first 2 bytes in the `MailReply` structure are a count of the number of `MailBlockInfo` structures, followed immediately by the structures. The function sets the value of the `dataSize` field to the number of bytes of data it placed in the buffer.

nextIndex   The sequence number of the first block whose information did not fit into your buffer. The function sets this field when your buffer is too small to hold all the information you requested. If there is no more information to return, the value of the `nextIndex` field is undefined. You must check the value of the `more` field before interpreting the value in the `nextIndex` field. The `nextIndex` field contains meaningful data only when the value of the `more` field is `true`.

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

DESCRIPTION

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

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$050F

RESULT CODES

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

SEE ALSO

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

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

MSAMGetBlock

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

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

paramBlock Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

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

blockType   A structure that specifies the creator and the type of the block that you want to read. You cannot specify a wildcard value for either the creator or block type.

blockIndex   A value that indicates the relative position of the block of type `blockType` that you want to read. To read all blocks of a specific block type, set this field to 1 the first time you call the `MSAMGetBlock` function and increment it by 1 in subsequent calls to the function until you have read all blocks of that type in the message. (Note that the value you supply here is distinct from the index used in the `MSAMEnumerateBlocks` function.)

buffer   A `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. The `MSAMGetBlock` function writes the information that you request into your buffer and sets the value of the `dataSize` field to the number of bytes of data it placed in the buffer.

dataOffset   The byte position relative to the beginning of the block at which you want the `MSAMGetBlock` function to begin reading. Set this field to 0 to read from the beginning of the block.

endOfBlock   A Boolean value that indicates whether the `MSAMGetBlock` function has returned the entire block. If the buffer that you provide is not large enough to contain an entire block, the `MSAMGetBlock` function sets this field to `false`. You can call the function again with an updated value in the `dataOffset` field to retrieve additional data. When the `MSAMGetBlock` function has returned the entire block, it sets the value of the `endOfBlock` field to `true`.

remaining   The number of bytes of data remaining in the block that the `MSAMGetBlock` function has not returned to you. If the `endOfBlock` field is set to `true`, the value of this field is 0.

DESCRIPTION

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

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

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

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

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

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0510

RESULT CODES

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

SEE ALSO

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

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

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

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

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

MSAMOpenNested

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

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

paramBlock   Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

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

nestedRef   A reference number that identifies the nested message opened by the `MSAMOpenNested` function.

DESCRIPTION

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

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

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

SPECIAL CONSIDERATIONS

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0509

RESULT CODES

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

SEE ALSO

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

The MSAMClose function is described on page 2-167.

Marking a Recipient

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

MSAMnMarkRecipients

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

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

paramBlock   Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

mailMsgRef   A message reference number that identifies an open message whose
              recipients you want to mark. You obtain this reference number from
              the `MSAMOpen` function. It is valid if you have not yet closed the
              message by calling the `MSAMClose` function.

buffer   A `MailBuffer` structure. You set the value of the `bufferSize`
              field in the `MailBuffer` structure to the number of bytes in your
              buffer. You place data in your buffer in the form of a `MailReply`
              structure. The first 2 bytes in the buffer contain the number of
              identifying values that follow. Then you store a value that identifies
              each recipient that you want to mark. Each identifying value is
              2 bytes long. The `dataSize` field in the `MailBuffer` structure
              is unused.

DESCRIPTION

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

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

The `MSAMnMarkRecipients` function clears the `responsible` flag for the
recipients you specify. If you call the `MSAMGetRecipients` function after calling
`MSAMnMarkRecipients`, the marked recipients have the `responsible` field of their

corresponding `MailResolvedRecipient` structures set to `false`. After you mark all of the recipients for a message, the `done` field in the `MSAMEnumerateOutQReply` structure is set to `true` for that message when you enumerate the outgoing queue.

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

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

SPECIAL CONSIDERATIONS

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0512

RESULT CODES

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

SEE ALSO

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

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

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

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

The `MSAMMarkRecipients` function is described next.

MSAMMarkRecipients

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

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

paramBlock   Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

queueRef   The value that identifies the outgoing queue that contains the message whose recipients you want to mark.

seqNum   A value that identifies the message whose recipients you want to mark. You obtain this value from the `MSAMEnumerate` function.

buffer   A `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. You place data in your buffer in the form of a `MailReply` structure. The first 2 bytes in the buffer contain the number of identifying values that follow. Then you store a value that identifies each recipient that you want to mark. Each identifying value is 2 bytes long. The identifying value is described on page 2-164. The `dataSize` field in the `MailBuffer` structure is unused.

DESCRIPTION

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

SPECIAL CONSIDERATIONS

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0505

RESULT CODES

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

SEE ALSO

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

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

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

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

Closing a Message

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

MSAMClose

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

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

paramBlock   Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

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

DESCRIPTION

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

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$050A

RESULT CODES

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

SEE ALSO

The MSAMOpen function is described on page 2-140.

Creating, Reading, and Writing Message Summaries

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

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

PMSAMCreateMsgSummary

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

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

paramBlock   Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

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

DESCRIPTION

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

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

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

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

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

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

SPECIAL CONSIDERATIONS

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

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0522

RESULT CODES

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

SEE ALSO

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

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

The `PMSAMGetMsgSummary` function is described next.

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

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

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

PMSAMGetMsgSummary

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

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

paramBlock Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

inQueueRef   The value identifying the incoming queue that holds the message summary you want to read. You obtain the queue reference from the `PMSAMOpenQueues` function.

seqNum   The sequence number that identifies the message summary in the incoming queue. You obtain this value from the `PMSAMCreateMsgSummary` function.

msgSummary   A pointer to a buffer in which the function stores the `MSAMMsgSummary` structure. You provide this buffer. Set this field to `nil` if you do not want to read the message summary.

buffer   A pointer to a `MailBuffer` structure. You set the value of the `bufferSize` field in the `MailBuffer` structure to the number of bytes in your buffer. The `PMSAMGetMsgSummary` function stores your private data associated with the message summary into the buffer and sets the value of the `dataSize` field to the number of bytes of data it actually placed in your buffer. Set this field to `nil` if you do not want to read your private data.

msgSummaryOffset
                The offset from the beginning of your private data area identifying the point at which you want to begin reading. If the `buffer` field is set to `nil`, the function ignores this field.

DESCRIPTION

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

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

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

SPECIAL CONSIDERATIONS

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0526

RESULT CODES

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

SEE ALSO

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

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

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

PMSAMPutMsgSummary

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

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

paramBlock Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

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

DESCRIPTION

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

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

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

SPECIAL CONSIDERATIONS

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0527

RESULT CODES

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

SEE ALSO

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

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

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

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

Creating a Message

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

MSAMCreate

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

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

`paramBlock` Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

`queueRef`   For a personal MSAM, specify the incoming queue reference that you obtained from the `PMSAMOpenQueues` function. The queue reference must belong to the slot to which the message is addressed. For a mail slot, the queue reference identifies the slot's actual incoming queue in which you want to deposit a letter. For a messaging slot, the queue reference identifies the slot itself. For a server MSAM, specify the server MSAM's queue reference that you obtained from the `SMSAMStartup` function.

`asLetter`   A Boolean value that indicates whether the message you are creating is a letter.

`msgType`   An `IPMMsgType` structure. If you are creating a letter, you must set the `format` field of the `IPMMsgType` structure to `kIPMOSFormatType` to indicate that the remainder of the `IPMMsgType` structure consists of an `OCECreatorType` structure.

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

DESCRIPTION

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

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

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

If you are creating a non-letter message, use an application-defined creator and type. You can set the `format` field of the `IPMMsgType` structure to either `kIPMOSFormatType` (which specifies that the message creator and message type information is formatted as type `OCECreatorType`) or `kIPMStringFormatType` (which specifies that the message

creator and message type information is formatted as type `Str32`). Typically, you use type `OCECreatorType`; type `Str32` is included for compatibility with the Program-to-Program Communications (PPC) Toolbox.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0514

RESULT CODES

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

SEE ALSO

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

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

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

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

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

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

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

Writing Header Information

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

MSAMPutAttribute

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

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

paramBlock Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

mailMsgRef   A reference number that identifies the letter to which you want to
                add an attribute. You obtain the reference number when you call the
                MSAMCreate function.

attrID   A value that identifies the type of attribute that you want to add to
                the letter.

buffer   A MailBuffer structure. You set the value of the bufferSize
                field in the MailBuffer structure to the number of bytes in your
                buffer. You store the value of the attribute that you want to add to
                the letter in your buffer. The MSAMPutAttribute function writes
                the information from the buffer to the letter and sets the value of the
                dataSize field to the number of bytes of data it actually wrote.

DESCRIPTION

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

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

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

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

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0518

RESULT CODES

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

SEE ALSO

The MailBuffer structure is described on page 2-96.

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

The MSAMGetAttributes function is described on page 2-142.

The PMSAMCreateMsgSummary function is described on page 2-169.

The MSAMPutRecipient function is described next.

MSAMPutRecipient

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

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

paramBlock   Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

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

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

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

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

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

DESCRIPTION

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

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

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

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

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

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

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

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

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

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

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0519

RESULT CODES

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

SEE ALSO

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

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

The `MSAMPutMsgHeader` function is described next.

MSAMPutMsgHeader

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

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

paramBlock Pointer to a parameter block.

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

Parameter block

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

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

Field descriptions

`mailMsgRef`   A reference number that identifies the non-letter message whose header you want to write. You obtain the reference number when you call the `MSAMCreate` function.

`replyQueue`   A pointer to an `OCERecipient` structure that specifies a reply queue—that is, a return address. You allocate the structure and completely specify it. The receiving application uses this address when it replies to the message. The IPM Manager sends reports to the reply queue address. You are free to specify that replies and reports go to an alternate address, instead of to the sender.

`sender`   A pointer to an `IPMSender` structure that contains the packed record ID or string that identifies the sender of the message.

`deliveryNotification`
                    A bit array that indicates the type of information you want to receive about the delivery of the message. Set the bit values appropriately to request reports with delivery indications (`kIPMDeliveryNotificationMask`), reports with non-delivery indications (`kIPMNonDeliveryNotificationMask`), or no reports (`kIPMNoNotificationMask`).

`priority`   A value that specifies the priority for delivering the message. Set this field to `kIPMHighPriority` to specify high priority. Set this field to `kIPMLowPriority` to specify low priority. Set this field to `kIPMNormalPriority` to specify normal priority.

DESCRIPTION

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

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

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

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

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$051D

RESULT CODES

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

SEE ALSO

The OCERecipient structure is described on page 2-106.

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

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

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

Writing a Message

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

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. s

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`.

s **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. s

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.

s **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. s

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
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

The `MSAMDelete` function is described next.

Letter attributes and the `MailIndications` data type are described on 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 inter-
                change 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

n   the stack pointer and register A6

n   registers A2, A3, and A4

n   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:

n   a pointer to an `SMCA` structure

n   a letter sequence number

n   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.*

Summary of the MSAM Interface

C Summary

Data Types and Constants

```
/* predefined message creator and message type */
#define kMailAppleMailCreator    'apml' /* message creator */
#define kMailLtrMsgType          'lttr' /* message type for letter, report */

/* predefined block creator and block types */
#define kMailAppleMailCreator    'apml' /* block creator */
#define kMailLtrHdrType          'lthd' /* letter header */
#define kMailContentType         'body' /* content of letter */
#define kMailEnclosureListType   'elst' /* list of enclosures */
#define kMailEnclosureDesktopType'edsk' /* Desktop Mgr info for enclosures */
#define kMailEnclosureFileType    'asgl' /* a file enclosure */
#define kMailImageBodyType       'imag' /* image of letter */
#define kMailMSAMType            'gwyi' /* MSAM-specific information */
#define kMailReportType          'rpti' /* report info */
#define kMailTunnelLtrType       'tunl' /* reserved */
#define kMailHopInfoType         'hopi' /* reserved */

/* families used for mail or related msgs */
#define kMailFamily       'mail' /* letter with content block or
                                          content enclosure */
#define kMailFamilyFile    'file' /* letter without content block or
                                          content enclosure */

#define kMailResolvedList  0     /* MailAttributeID value for resolved
                                     recipient list */

/* mask to test location flags of a slot */
#define MailLocationMask(locationNumber) (1<<((locationNumber)-1))

/* bit flags of MailAttributeID type */
enum {
   kMailLetterFlagsBit    = 1,  /* letter flags bit */
   kMailIndicationsBit    = 3,  /* indications bit */
```

```
   kMailMsgTypeBit          = 4,  /* letter creator & type bit */
   kMailLetterIDBit         = 5,  /* letter ID bit */
   kMailSendTimeStampBit    = 6,  /* send timestamp bit */
   kMailNestingLevelBit     = 7,  /* nesting level bit */
   kMailMsgFamilyBit        = 8,  /* message family bit */
   kMailReplyIDBit          = 9,  /* reply ID bit */
   kMailConversationIDBit   = 10, /* conversation ID bit */
   kMailSubjectBit          = 11, /* subject bit */
   kMailFromBit             = 12, /* From recipient bit */
   kMailToBit               = 13, /* To recipient bit */
   kMailCcBit               = 14, /* cc recipient bit */
   kMailBccBit              = 15  /* bcc recipient bit */
};

/* Values of MailAttributeMask data type. The masks are defined for use
 with the MailAttributeBitmap data type. However, because the
 MailAttributeBitmap data type is defined as a bit field structure, and the
 masks operate on variables of type long, you cannot use these masks to set
 or test the value of a bit field in a MailAttributeBitmap structure. The
 masks are included for historical reasons only. */
enum {
   kMailLetterFlagsMask       = 1L<<(kMailLetterFlagsBit-1),
   kMailIndicationsMask       = 1L<<(kMailIndicationsBit-1),
   kMailMsgTypeMask           = 1L<<(kMailMsgTypeBit-1),
   kMailLetterIDMask          = 1L<<(kMailLetterIDBit-1),
   kMailSendTimeStampMask     = 1L<<(kMailSendTimeStampBit-1),
   kMailNestingLevelMask      = 1L<<(kMailNestingLevelBit-1),
   kMailMsgFamilyMask         = 1L<<(kMailMsgFamilyBit-1),
   kMailReplyIDMask           = 1L<<(kMailReplyIDBit-1),
   kMailConversationIDMask    = 1L<<(kMailConversationIDBit-1),
   kMailSubjectMask           = 1L<<(kMailSubjectBit-1),
   kMailFromMask              = 1L<<(kMailFromBit-1),
   kMailToMask                = 1L<<(kMailToBit-1),
   kMailCcMask                = 1L<<(kMailCcBit-1),
   kMailBccMask               = 1L<<(kMailBccBit-1)
};

/* bit flags of MailIndications type */
enum {
   kMailOriginalInReportBit   = 1,
   kMailNonReceiptReportsBit   = 3,
   kMailReceiptReportsBit      = 4,
   kMailForwardedBit           = 5,
   kMailPriorityBit            = 6,
```

```
   kMailIsReportWithOriginalBit  = 8,
   kMailIsReportBit              = 9,
   kMailHasContentBit            = 10,
   kMailHasSignatureBit          = 11,
   kMailAuthenticatedBit         = 12,
   kMailSentBit                  = 13
};

/* Masks for bits of MailIndications type. Because the MailIndications data
 type is defined as a bit field structure, and the masks operate on variables
 of type long, you cannot use the masks to set or test the value of a bit
 field in a MailIndications structure. The masks are included for
 historical reasons only. */
enum {
   kMailSentMask                 = 1L<<(kMailSentBit-1),
   kMailAuthenticatedMask        = 1L<<(kMailAuthenticatedBit-1),
   kMailHasSignatureMask         = 1L<<(kMailHasSignatureBit-1),
   kMailHasContentMask           = 1L<<(kMailHasContentBit-1),
   kMailIsReportMask             = 1L<<(kMailIsReportBit-1),
   kMailIsReportWithOriginalMask = 1L<<(kMailIsReportWithOriginalBit-1),
   kMailPriorityMask             = 3L<<(kMailPriorityBit-1),
   kMailForwardedMask            = 1L<<(kMailForwardedBit-1),
   kMailReceiptReportsMask       = 1L<<(kMailReceiptReportsBit-1),
   kMailNonReceiptReportsMask    = 1L<<(kMailNonReceiptReportsBit-1),
   kMailOriginalInReportMask     = 3L<<(kMailOriginalInReportBit-1)
};

/* bit values of the originalInReport field in MailIndications */
enum {
   kMailNoOriginal        = 0,  /* do not enclose original in report */
   kMailEncloseOnNonReceipt= 3   /* enclose original with non-delivery
                                    indication */
};

/* values of MailSegmentType type*/
enum {
   kMailInvalidSegmentType    = 0,
   kMailTextSegmentType       = 1,
   kMailPictSegmentType       = 2,
   kMailSoundSegmentType      = 3,
   kMailStyledTextSegmentType = 4,
   kMailMovieSegmentType      = 5
};
```

```
enum {
   kMailTextSegmentBit,
   kMailPictSegmentBit,
   kMailSoundSegmentBit,
   kMailStyledTextSegmentBit,
   kMailMovieSegmentBit
};

/* values of MailSegmentMask type */
enum {
   kMailTextSegmentMask       = 1L<<kMailTextSegmentBit,
   kMailPictSegmentMask       = 1L<<kMailPictSegmentBit,
   kMailSoundSegmentMask      = 1L<<kMailSoundSegmentBit,
   kMailStyledTextSegmentMask = 1L<<kMailStyledTextSegmentBit,
   kMailMovieSegmentMask      = 1L<<kMailMovieSegmentBit
};

/* values of MailBlockMode type */
enum {
   kMailFromStart = 1,  /* write data at offset from start of block */
   kMailFromLEOB  = 2,  /* write data at offset from end of block */
   kMailFromMark  = 3   /* write data at offset from the current mark */
};

/* bit values of MailLetterSystemFlags type */
enum {
   kMailIsLocalBit   = 2   /* letter is available locally */
   };

enum {
   kMailIsLocalMask  = 1L<<kMailIsLocalBit
};

/* bit values of MailLetterUserFlags type */
enum {
   kMailReadBit,        /* letter has been opened */
   kMailDontArchiveBit, /* reserved */
   kMailInTrashBit      /* reserved */
   };

enum {
   kMailReadMask         = 1L<<kMailReadBit,
   kMailDontArchiveMask  = 1L<<kMailDontArchiveBit,
   kMailInTrashMask      = 1L<<kMailInTrashBit
};
```

```
#define kMailErrorLogEntryVersion 0x101

/* 'STR#' resource IDs for personal MSAM's error and action messages */
#define kMailMSAMErrorStringListID  128 /* list of error message strings */
#define kMailMSAMActionStringListID 129 /* list of action message strings */

/* values of MailLogErrorType type*/
enum {
   kMailELECorrectable    = 0,  /* error correctable by user */
   kMailELEError          = 1,  /* error not correctable by user */
   kMailELEWarning        = 2,  /* warning requiring no user intervention */
   kMailELEInformational  = 3   /* informational message */
};

/* predefined values of MailLogErrorCode type */
enum {
   kMailMSAMErrorCode= 0,     /* MSAM-defined error */
   kMailMiscError    = -1,    /* miscellaneous error */
   kMailNoModem      = -2     /* modem required, but missing */
};

#define kMailMsgSummaryVersion   1

#define kMailMaxPMSAMMsgSummaryData 128/* maximum bytes for private MSAM
                                          message summary data */

/* defines for the addressedToMe field in MailCoreData */
#define kAddressedAs_TO 0x1
#define kAddressedAs_CC 0x2
#define kAddressedAs_BCC 0x4

enum {
   kMailTimerOff       = 0,  /* no timer specified */
   kMailTimerTime      = 1,  /* timer relative to midnight */
   kMailTimerFrequency = 2   /* frequency timer */
};

/* values of PMSAMStatus type */
enum {
   kPMSAMStatusPending  = 1,  /* for outQueue */
   kPMSAMStatusError    = 2,  /* for inQueue letters */
   kPMSAMStatusSending  = 3,  /* for outQueue */
   kPMSAMStatusCaching  = 4,  /* for inQueue letters */
   kPMSAMStatusSent     = 5   /* for outQueue */
};
```

```
#define kMailEPPCMsgVersion    3

/* values of AOCE high-level event message classes */
enum {
    kMailEPPCCreateSlot         = 'crsl',
    kMailEPPCModifySlot         = 'mdsl',
    kMailEPPCDeleteSlot         = 'dlsl',
    kMailEPPCShutDown           = 'quit',
    kMailEPPCMailboxOpened       = 'mbop',
    kMailEPPCMailboxClosed       = 'mbcl',
    kMailEPPCMsgPending         = 'msgp',
    kMailEPPCSendImmediate       = 'sndi',
    kMailEPPCContinue           = 'cont',
    kMailEPPCSchedule           = 'sked',
    kMailEPPCAdmin              = 'admn',
    kMailEPPCInQUpdate          = 'inqu',
    kMailEPPCMsgOpened          = 'msgo',
    kMailEPPCDeleteOutQMsg       = 'dlom',
    kMailEPPCWakeup             = 'wkup',
    kMailEPPCLocationChanged     = 'locc'
};

/* values of SMSAMAdminCode type */
enum {
    kSMSAMNotifyFwdrSetupChange   = 1,
    kSMSAMNotifyFwdrNameChange    = 2,
    kSMSAMNotifyFwdrPwdChange     = 3,
    kSMSAMGetDynamicFwdrParams    = 4
};

enum {
    kSMSAMFwdrHomeInternetChangedBit,
    kSMSAMFwdrConnectedToChangedBit,
    kSMSAMFwdrForeignRLIsChangedBit,
    kSMSAMFwdrMnMServerChangedBit
};

/* values of SMSAMSlotChanges type */
enum {
    kSMSAMFwdrEverythingChangedMask  = -1,
    kSMSAMFwdrHomeInternetChangedMask= 1L<<kSMSAMFwdrHomeInternetChangedBit,
    kSMSAMFwdrConnectedToChangedMask = 1L<<kSMSAMFwdrConnectedToChangedBit,
    kSMSAMFwdrForeignRLIsChangedMask = 1L<<kSMSAMFwdrForeignRLIsChangedBit,
    kSMSAMFwdrMnMServerChangedMask   = 1L<<kSMSAMFwdrMnMServerChangedBit
};
```

```
enum {
   kOCESetupLocationNone   = 0,   /* disconnect state */
   kOCESetupLocationMax    = 8    /* maximum location value */
};

typedef long MailMsgRef;        /* reference to new/open letter or message */

typedef long MSAMQueueRef;          /* reference to an open MSAM queue */

typedef unsigned short MSAMSlotID;  /* slot identifier */

typedef unsigned short MailSlotID;  /* identifies slots within a mailbox */

typedef long MailboxRef;            /* reference to an active mailbox */

typedef unsigned short MailAttributeID;/* letter attribute identifier */

/* The MailAttributeMask data type defines a set of masks for the
 MailAttributeBitmap data type. However, because the MailAttributeBitmap data
 type is defined as a bit field structure, and the masks operate on variables
 of type long, you cannot use the masks to set or test the value of a bit
 field in a MailAttributeBitmap structure. The MailAttributeMask data type is
 included for historical reasons only. */
typedef unsigned long MailAttributeMask;

typedef IPMMsgID MailLetterID;

typedef unsigned short MailNestingLevel;

typedef OCERecipient MailRecipient;

typedef unsigned short MailSegmentMask;

typedef unsigned short MailSegmentType;

typedef short MailBlockMode;

typedef unsigned short PMSAMStatus;

typedef char OCESetupLocation;              /* current system location */

typedef unsigned char MailLocationFlags;  /* slot location flags */

struct MailBuffer {
   long      bufferSize; /* size of your buffer */
   Ptr       buffer;     /* pointer to your buffer  */
   long      dataSize;   /* amount of data returned in or read out
                            of your buffer */
};

typedef struct MailBuffer MailBuffer;
```

```
struct MailReply {
   unsigned short tupleCount;
   /* tuple[tupleCount] */
};

typedef struct MailReply MailReply;

struct MSAMEnumerateOutQReply {
   long            seqNum;     /* sequence number of message */
   Boolean         done;       /* resolution of message */
   IPMPriority     priority;   /* priority of message */
   OSType          msgFamily   /* message family */
   long            approxSize; /* size of message */
   Boolean         tunnelForm; /* reserved */
   Byte            padByte;    /* for even byte boundary */
   NetworkSpec     nextHop;    /* reserved */
   OCECreatorType msgType;     /* message creator and type */
};

typedef struct MSAMEnumerateOutQReply MSAMEnumerateOutQReply;

struct MSAMEnumerateInQReply {
   long    seqNum;     /* letter sequence number */
   Boolean msgDeleted; /* should letter be deleted? */
   Boolean msgUpdated; /* was message summary updated? */
   Boolean msgCached;  /* is letter in the incoming queue? */
   Byte    padByte;    /* for even byte boundary */
};

typedef struct MSAMEnumerateInQReply MSAMEnumerateInQReply;

struct MailAttributeBitmap {
   unsigned int                /* 32 bits */
      reservedA:16,            /* bits 17 through 32 reserved */
      reservedB:1,             /* bit 16--reserved */
      bcc:1,                   /* bit 15--blind carbon copy recipients */
      cc:1,                    /* bit 14--carbon copy recipients */
      to:1,                    /* bit 13--To recipients */
      from:1,                  /* bit 12--sender of letter */
      subject:1,               /* bit 11--subject of letter */
      conversationID:1,        /* bit 10--ID of conversation thread */
      replyID:1,               /* bit 09--ID of letter being replied to */
      msgFamily:1,             /* bit 08--message family */
      nestingLevel:1,          /* bit 07--nesting level of letter */
      sendTimeStamp:1,         /* bit 06--time letter was sent */
```

```
      letterID:1;               /* bit 05--letter's unique ID number */
      msgType:1,                /* bit 04--letter's creator and type */
      indications:1,            /* bit 03--MailIndications */
      reservedC:1,              /* bit 02--reserved */
      letterFlags:1             /* bit 01--letter flags */
};

typedef struct MailAttributeBitmap MailAttributeBitmap;

struct MailIndications {
unsigned int
   reservedB:16,
   hasStandardContent:1,/* letter has a content block */
   hasImageContent:1,    /* letter an image block */
   hasNativeContent:1,   /* letter has a content enclosure */
   sent:1,               /* letter sent, not just composed */
   authenticated:1,      /* letter was created and transported with
                            authentication */
   hasSignature:1,       /* letter was signed with a digital signature */
   hasContent:1,         /* this letter or a nested letter has content */
   isReport:1,           /* is really a report */
   isReportWithOriginal:1,
                         /* Report contains the original letter */
   priority:2,           /* letter has normal, low, or high priority */
   forwarded:1,          /* letter contains a forwarded letter */
   receiptReports:1,     /* originator requests delivery indications */
   nonReceiptReports:1,  /* originator requests non-delivery indications */
   originalInReport:2    /* originator wants original letter enclosed in
                            reports */
};

typedef struct MailIndications MailIndications;

struct OCERecipient {
   RecordID*       entitySpecifier;
   OSType          extensionType;
   unsigned short  extensionSize;
   Ptr             extensionValue;
};

struct OCEPackedRecipient {
   unsigned short dataLength; /* length of recipient data */
                  /* followed by recipient data of dataLength bytes */
};
```

```
struct MailOriginalRecipient {
   short     index;      /* index for recipient */
             /* followed by OCEPackedRecipient structure */
};

typedef struct MailOriginalRecipient MailOriginalRecipient;

struct MailResolvedRecipient {
   short        index;        /* index for recipient */
   short        recipientFlags;/* recipient information */
   Boolean      responsible;   /* responsible for delivery? */
   Byte         padByte;
                /* followed by OCEPackedRecipient structure */
};

typedef struct MailResolvedRecipient MailResolvedRecipient;

struct MailEnclosureInfo {
   StringPtr   enclosureName; /* name of the enclosure */
   CInfoPBPtr  catInfo;       /* HFS catalog info about enclosure */
   StringPtr   comment;       /* comment for Get-Info window */
   Ptr         icon;          /* icon for enclosure file */
};

typedef struct MailEnclosureInfo MailEnclosureInfo;

typedef unsigned short MailLogErrorType;

typedef short MailLogErrorCode;

struct MailErrorLogEntryInfo {
   short              version;              /* log entry version */
   UTCTime            timeOccurred;         /* time of error */
   Str31              reportingPMSAM;       /* which MSAM? */
   Str31              reportingMSAMSlot;    /* which slot? */
   MailLogErrorType   errorType;            /* level of error */
   MailLogErrorCode   errorCode;            /* error code */
   short              errorResource;        /* error string resource index */
   short              actionResource;       /* action string resource index */
   unsigned long      filler;               /* reserved */
   unsigned short     filler2;              /* reserved */
};

typedef struct MailErrorLogEntryInfo MailErrorLogEntryInfo;
```

```
struct MailMasterData {
   MailAttributeBitmap  attrMask;       /* indicates attributes present in
                                            MSAMMsgSummary */
   MailLetterID         messageID;      /* ID of this letter *
   MailLetterID         replyID;        /* ID of letter this is a reply to */
   MailLetterID         conversationID;/* ID of letter that started this
                                            conversation */
};

typedef struct MailMasterData MailMasterData;

struct MailCoreData {
   MailLetterFlags      letterFlags;    /* letter status flags */
   unsigned long        messageSize     /* size of letter */
   MailIndications      letterIndications;
                                        /* indications for this letter */
   OCECreatorType       messageType;    /* message creator and type of this
                                            letter */
   MailTime             sendTime;       /* time this letter was sent */
   OSType               messageFamily; /* message family */
   unsigned char        reserved;
   unsigned char        addressedToMe; /* user is To, cc, or bcc recipient */
   char                 agentInfo[6];   /* reserved */
   RString32            sender;         /* sender of this letter */
   RString32            subject;        /* subject of this letter */
};

typedef struct MailCoreData MailCoreData;

struct MSAMMsgSummary {
   short                version;        /* version of the MSAMMsgSummary */
   Boolean              msgDeleted;     /* true if letter is to be deleted by
                                            personal MSAM */
   Boolean              msgUpdated;     /* true if MSAMMsgSummary was updated
                                            by IPM Manager */
   Boolean              msgCached;      /* true if letter is in the inQueue */
   Byte                 padByte;
   MailMasterData       masterData;     /* attributes not essential to
                                            display */
   MailCoreData         coreData;       /* attributes critical to display */
   /* followed by the personal MSAM's private data: Byte PMSAMSpecData[]; */
};

typedef struct MSAMMsgSummary MSAMMsgSummary;
```

```
struct MailLocationInfo {
   OCESetupLocation    location;    /* the current location */
   MailLocationFlags   active;      /* slot's location flags */
};

typedef struct MailLocationInfo MailLocationInfo;

struct MailEPPCMsg {
   short    version;                    /* message version */
   union {
      SMCA *    theSMCA;               /* pointer to SMCA */
      long      sequenceNumber;        /* letter sequence number */
      MailLocationInfo locationInfo;/* location information */
   } u;
};

typedef struct MailEPPCMsg MailEPPCMsg;

struct SMCA {
   unsigned short smcaLength; /* length of entire SMCA
                                    (including the length field) */
   OSErr          result;     /* result code */
   long           userBytes;  /* command interpreted user data */
   union{
      CreationID  slotCID;    /* creation ID of record
                                    containing slot information */
      long        msgHint;    /* message reference value */
   } u;
};

typedef struct SMCA SMCA;

typedef unsigned short SMSAMAdminCode;

struct SMSAMAdminEPPCRequest {
   SMSAMAdminCode          adminCode;        /* admin code */
   union {
      SMSAMSetupChange     setupChange;      /* setup change */
      SMSAMNameChange      nameChange;       /* reserved */
      SMSAMPasswordChange  passwordChange;   /* reserved */
      SMSAMDynamicParams   dynamicParams;    /* reserved */
   } u;
};

typedef struct SMSAMAdminEPPCRequest SMSAMAdminEPPCRequest;

typedef unsigned long SMSAMSlotChanges;
```

```
struct SMSAMSetupChange {
   SMSAMSlotChanges  whatChanged;    /* bitmap of changed parameters */
   AddrBlock         serverHint;     /* AOCE server address */
};

typedef struct SMSAMSetupChange SMSAMSetupChange;

struct SMSAMNameChange {        /* reserved data type */
   RString     newName;        /* sever MSAM's new name */
   AddrBlock   serverHint;     /* AOCE server address */
};

typedef struct SMSAMNameChange SMSAMNameChange;

struct SMSAMPasswordChange {  /* reserved data type */
   RString     newPassword;    /* server MSAM's new password */
   AddrBlock   serverHint;     /* AOCE server address */
};

typedef struct SMSAMPasswordChange SMSAMPasswordChange;

struct SMSAMDynamicParams {      /* reserved data type */
   unsigned long  curDiskUsed;    /* disk space used */
   unsigned long  curMemoryUsed; /* memory used */
};

typedef struct SMSAMDynamicParams SMSAMDynamicParams;

struct MailTime {
   UTCTime     time;    /* current UTC (GMT) */
   UTCOffset   offset;  /* offset from UTC */
};

typedef struct MailTime MailTime;

union MailTimer {
   long  frequency;     /* how often to connect */
   long  connectTime;   /* time since midnight */
};

typedef union MailTimer MailTimer;

typedef Byte MailTimerKind;
```

```
struct MailTimers {
   MailTimerKind  sendTimeKind;        /* timer kind for sending */
   MailTimerKind  receiveTimeKind;     /* timer kind for receiving */
   MailTimer      send;                /* connect time or frequency for
                                          sending letters */
   MailTimer      receive;             /* connect time or frequency for
                                          receiving letters */
};

typedef struct MailTimers MailTimers;

struct MailStandardSlotInfoAttribute {
   short             version;          /* MSAM version of the slot */
   MailLocationFlags active;           /* active at location i if
                                          MailLocationMask (i) is set */
   Byte              padByte;
   MailTimers        sendReceiveTimer;
};

typedef struct MailStandardSlotInfoAttribute MailStandardSlotInfoAttribute;

typedef unsigned short  MailLetterSystemFlags;

typedef unsigned short  MailLetterUserFlags;

struct MailLetterFlags {
   MailLetterSystemFlags  sysFlags;   /* system flags */
   MailLetterUserFlags    userFlags;  /* user flags */
};

typedef struct MailLetterFlags MailLetterFlags;

struct MailMaskedLetterFlags {
   MailLetterFlags  flagMask;    /* flags that are to be set */
   MailLetterFlags  flagValues;  /* their values */
};

typedef struct MailMaskedLetterFlags MailMaskedLetterFlags;

struct MailBlockInfo {
   OCECreatorType blockType;
   unsigned long  offset;
   unsigned long  blockLength;
};

typedef struct MailBlockInfo MailBlockInfo;
```

```
# define MailParamBlockHeader
   Ptr            qLink;         /* reserved */ \
   long           reservedH1;    /* reserved */ \
   long           reservedH2;    /* reserved */ \
   ProcPtr        ioCompletion;  /* pointer to completion routine */ \
   OSErr          ioResult;      /* result code */ \
   long           saveA5;        /* pointer to global variables */ \
   short          reqCode;       /* reserved */

struct PMSAMGetMSAMRecordPB {
   MailParamBlockHeader
   CreationID  msamCID;
};

typedef struct PMSAMGetMSAMRecordPB PMSAMGetMSAMRecordPB;

struct PMSAMOpenQueuesPB {
   MailParamBlockHeader
   MSAMQueueRef   inQueueRef;
   MSAMQueueRef   outQueueRef;
   MSAMSlotID     msamSlotID;
   long           filler[2];
};

typedef struct PMSAMOpenQueuesPB PMSAMOpenQueuesPB;

struct PMSAMSetStatusPB {
   MailParamBlockHeader
   MSAMQueueRef   queueRef;
   long           seqNum;
   long           msgHint;
   PMSAMStatus    status;
};

typedef struct PMSAMSetStatusPB PMSAMSetStatusPB;

struct PMSAMLogErrorPB {
   MailParamBlockHeader
   MSAMSlotID             msamSlotID; /* 0 for PMSAM errors */
   MailErrorLogEntryInfo* logEntry;
   long                   filler[2];
};

typedef struct PMSAMLogErrorPB PMSAMLogErrorPB;
```

```
struct PMSAMCreateMsgSummaryPB {
   MailParamBlockHeader
   MSAMQueueRef        inQueueRef;
   long                seqNum;      /* sequence number of new letter */
   MSAMMsgSummary *    msgSummary; /* attributes and mask filled in */
   MailBuffer *        buffer;      /* private MSAM data to add to summary */
};

typedef struct PMSAMCreateMsgSummaryPB PMSAMCreateMsgSummaryPB;

sstruct PMSAMPutMsgSummaryPB {
   MailParamBlockHeader
   MSAMQueueRef            inQueueRef;
   long                    seqNum;
   MailMaskedLetterFlags * letterFlags;
   MailBuffer*             buffer;        /* PMSAM private data */
};

typedef struct PMSAMPutMsgSummaryPB PMSAMPutMsgSummaryPB;

struct PMSAMGetMsgSummaryPB {
   MailParamBlockHeader
   MSAMQueueRef       inQueueRef;
   long               seqNum;
   MSAMMsgSummary *   msgSummary;        /* buffer for message summary */
   MailBuffer *       buffer;            /* buffer for PMSAM private data */
   unsigned short     msgSummaryOffset; /* offset of PMSAM private data
                                           from start of message summary */
};

typedef struct PMSAMGetMsgSummaryPB PMSAMGetMsgSummaryPB;

struct SMSAMSetupPB {
    MailParamBlockHeader
    RecordIDPtr serverMSAM;
    RStringPtr  password;
    OSType      gatewayType;
    RStringPtr  gatewayTypeDescription;
    AddrBlock   catalogServerHint;
 };

typedef struct SMSAMSetupPB SMSAMSetupPB;
```

```
struct SMSAMStartupPB {
   MailParamBlockHeader
   AuthIdentity    msamIdentity;
   MSAMQueueRef    queueRef;
};

typedef struct SMSAMStartupPB SMSAMStartupPB;

struct SMSAMShutdownPB {
   MailParamBlockHeader
   MSAMQueueRef    queueRef;
};

typedef struct SMSAMShutdownPB SMSAMShutdownPB;

struct MSAMEnumeratePB {
   MailParamBlockHeader
   MSAMQueueRef    queueRef;
   long            startSeqNum;
   long            nextSeqNum;
   MailBuffer      buffer;
};

typedef struct MSAMEnumeratePB MSAMEnumeratePB;

struct MSAMDeletePB {
   MailParamBlockHeader
   MSAMQueueRef    queueRef;
   long            seqNum;
   Boolean         msgOnly; /* delete letter and message summary? */
   Byte            padByte;
   OSErr           result;  /* reserved */
};

typedef struct MSAMDeletePB MSAMDeletePB;

struct MSAMOpenPB {
   MailParamBlockHeader
   MSAMQueueRef    queueRef;
   long            seqNum;
   MailMsgRef      mailMsgRef;
};

typedef struct MSAMOpenPB MSAMOpenPB;
```

```
struct MSAMOpenNestedPB {
   MailParamBlockHeader
   MailMsgRef  mailMsgRef;
   MailMsgRef  nestedRef;
};

typedef struct MSAMOpenNestedPB MSAMOpenNestedPB;

struct MSAMClosePB {
   MailParamBlockHeader
   MailMsgRef  mailMsgRef;
};

typedef struct MSAMClosePB MSAMClosePB;

struct MSAMGetMsgHeaderPB {
   MailParamBlockHeader
   MailMsgRef        mailMsgRef;
   IPMHeaderSelector selector;
   unsigned long     offset;
   MailBuffer        buffer;
   unsigned long     remaining;
};

typedef struct MSAMGetMsgHeaderPB MSAMGetMsgHeaderPB;

struct MSAMGetAttributesPB {
   MailParamBlockHeader
   MailMsgRef         mailMsgRef;
   MailAttributeBitmap requestMask;
   MailBuffer         buffer;
   MailAttributeBitmap responseMask;
   Boolean            more;
};

typedef struct MSAMGetAttributesPB MSAMGetAttributesPB;

struct MSAMGetRecipientsPB {
   MailParamBlockHeader
   MailMsgRef       mailMsgRef;
   MailAttributeID  attrID;     /* kMailFromBit thru kMailBccBit */
   unsigned short   startIndex; /* starts at 1 */
   MailBuffer       buffer;
   unsigned short   nextIndex;
   Boolean          more;
};

typedef struct MSAMGetRecipientsPB MSAMGetRecipientsPB;
```

```
struct MSAMGetContentPB {
   MailParamBlockHeader
   MailMsgRef         mailMsgRef;
   MailSegmentMask    segmentMask;
   MailBuffer         buffer;
   StScrpRec *        textScrap;
   ScriptCode         script;
   MailSegmentType    segmentType;
   Boolean            endOfScript;
   Boolean            endOfSegment;
   Boolean            endOfContent;
   long               segmentLength;
   long               segmentID;
};

typedef struct MSAMGetContentPB MSAMGetContentPB;

struct MSAMGetEnclosurePB {
   MailParamBlockHeader
   MailMsgRef  mailMsgRef;
   Boolean     contentEnclosure;
   Byte        padByte;
   MailBuffer  buffer;
   Boolean     endOfFile;
   Boolean     endOfEnclosures;
};

typedef struct MSAMGetEnclosurePB MSAMGetEnclosurePB;

struct MSAMEnumerateBlocksPB {
   MailParamBlockHeader
   MailMsgRef         mailMsgRef;
   unsigned short     startIndex; /* starts at 1 */
   MailBuffer         buffer;
   unsigned short     nextIndex;
   Boolean            more;
};

typedef struct MSAMEnumerateBlocksPB MSAMEnumerateBlocksPB;

struct MSAMGetBlockPB {
   MailParamBlockHeader
   MailMsgRef      mailMsgRef;
   OCECreatorType  blockType;
   unsigned short  blockIndex;
```

```
   MailBuffer       buffer;
   unsigned long  dataOffset;
   Boolean          endOfBlock;
   Byte             padByte;
   unsigned long  remaining;
};

typedef struct MSAMGetBlockPB MSAMGetBlockPB;

struct MSAMMarkRecipientsPB {
   MailParamBlockHeader
   MSAMQueueRef    queueRef;
   long              seqNum;
   MailBuffer      buffer;
};

typedef struct MSAMMarkRecipientsPB MSAMMarkRecipientsPB;

struct MSAMnMarkRecipientsPB {
   MailParamBlockHeader
   MailMsgRef      mailMsgRef;
   MailBuffer      buffer;
};

typedef struct MSAMnMarkRecipientsPB MSAMMarknRecipientsPB;

struct MSAMCreatePB {
   MailParamBlockHeader
   MSAMQueueRef    queueRef;
   Boolean          asLetter;       /* create as letter or message */
   IPMMsgType      msgType;
   long              refCon;         /* for messages only */
   long              seqNum;         /* set if creating message in the inQueue */
   Boolean          tunnelForm;     /* always false */
   Boolean          bccRecipients; /* true if creating letter with bcc
                                        recipients */
   MailMsgRef      newRef;
};

typedef struct MSAMCreatePB MSAMCreatePB;

struct MSAMBeginNestedPB {
   MailParamBlockHeader
   MailMsgRef  mailMsgRef;
   long        refCon;     /* for messages only */
   IPMMsgType  msgType;
};

typedef struct MSAMBeginNestedPB MSAMBeginNestedPB;
```

```
struct MSAMEndNestedPB {
   MailParamBlockHeader
   MailMsgRef  mailMsgRef;
};

typedef struct MSAMEndNestedPB MSAMEndNestedPB;

struct MSAMSubmitPB {
   MailParamBlockHeader
   MailMsgRef     mailMsgRef;     /* message reference number */
   Boolean        submitFlag;     /* submit or delete message? */
   Byte           padByte;
   MailLetterID   msgID;          /* reserved */
};

typedef struct MSAMSubmitPB MSAMSubmitPB;

struct MSAMPutMsgHeaderPB {
   MailParamBlockHeader
   MailMsgRef           mailMsgRef;
   OCERecipient *       replyQueue;
   IPMSender *          sender;
   IPMNotificationType  deliveryNotification;
   IPMPriority          priority;
};

typedef struct MSAMPutMsgHeaderPB MSAMPutMsgHeaderPB;

struct MSAMPutAttributePB {
   MailParamBlockHeader
   MailMsgRef       mailMsgRef;
   MailAttributeID  attrID;
   MailBuffer       buffer;
};

typedef struct MSAMPutAttributePB MSAMPutAttributePB;

struct MSAMPutRecipientPB {
   MailParamBlockHeader
   MailMsgRef       mailMsgRef;
   MailAttributeID  attrID;
   MailRecipient *  recipient;
   Boolean          responsible;   /* for server and message msams only */
};

typedef struct MSAMPutRecipientPB MSAMPutRecipientPB;
```

```
struct MSAMPutContentPB {
   MailParamBlockHeader
   MailMsgRef        mailMsgRef;
   MailSegmentType   segmentType;
   Boolean           append;
   Byte              padByte;
   MailBuffer        buffer;
   StScrpRec *       textScrap;
   Boolean           startNewScript;
   ScriptCode        script;         /* valid when startNewScript is true */
};

typedef struct MSAMPutContentPB MSAMPutContentPB;

struct MSAMPutEnclosurePB {
   MailParamBlockHeader
   MailMsgRef        mailMsgRef;
   Boolean           contentEnclosure;
   Byte              padByte;
   Boolean           hfs;
   Boolean           append;
   MailBuffer        buffer;    /* unused if hfs is true */
   FSSpec            enclosure;
   MailEnclosureInfo addlInfo;
};

typedef struct MSAMPutEnclosurePB MSAMPutEnclosurePB;

struct MSAMPutBlockPB {
   MailParamBlockHeader
   MailMsgRef      mailMsgRef;
   long            refCon;         /* for messages only */
   OCECreatorType  blockType;
   Boolean         append;
   MailBuffer      buffer;
   MailBlockMode   mode;
   unsigned long   offset;
};

typedef struct MSAMPutBlockPB MSAMPutBlockPB;
```

```
struct MSAMCreateReportPB {
   MailParamBlockHeader
   MailMsgRef        mailMsgRef;
   MailLetterID      msgID;   /* letter ID of letter being reported on */
   MailRecipient *   sender;  /* sender of the letter being reported on */
};

typedef struct MSAMCreateReportPB MSAMCreateReportPB;

struct MSAMPutRecipientReportPB {
   MailParamBlockHeader
   MailMsgRef  mailMsgRef;
   short       recipientIndex;   /* recipient index in the original letter */
   OSErr       result;           /* result of sending to the recipient */
};

typedef struct MSAMPutRecipientReportPB MSAMPutRecipientReportPB;

struct MailWakeupPMSAMPB {
   MailParamBlockHeader
   CreationID  pmsamCID;
   MailSlotID  mailSlotID;
};

typedef struct MailWakeupPMSAMPB MailWakeupPMSAMPB;

struct MailCreateMailSlotPB {
   MailParamBlockHeader
   MailboxRef  mailboxRef;
   long        timeout;
   CreationID  pmsamCID;
   SMCA        smca;
};

typedef struct MailCreateMailSlotPB MailCreateMailSlotPB;

struct MailModifyMailSlotPB {
   MailParamBlockHeader
   MailboxRef  mailboxRef;
   long        timeout;
   CreationID  pmsamCID;
   SMCA        smca;
};

typedef struct MailModifyMailSlotPB MailModifyMailSlotPB;
```

```
union MSAMParam {
    struct {MailParamBlockHeader} header;

    PMSAMGetMSAMRecordPB        pmsamGetMSAMRecord;
    PMSAMOpenQueuesPB           pmsamOpenQueues;
    PMSAMSetStatusPB            pmsamSetStatus;
    PMSAMLogErrorPB             pmsamLogError;

    SMSAMSetupPB                smsamSetup;
    SMSAMStartupPB              smsamStartup;
    SMSAMShutdownPB             smsamShutdown;

    MSAMEnumeratePB             msamEnumerate;
    MSAMDeletePB                msamDelete;

    MSAMOpenPB                  msamOpen;
    MSAMOpenNestedPB            msamOpenNested;
    MSAMClosePB                 msamClose;
    MSAMGetMsgHeaderPB          msamGetMsgHeader;
    MSAMGetAttributesPB         msamGetAttributes;
    MSAMGetRecipientsPB         msamGetRecipients;
    MSAMGetContentPB            msamGetContent;
    MSAMGetEnclosurePB          msamGetEnclosure;
    MSAMEnumerateBlocksPB       msamEnumerateBlocks;
    MSAMGetBlockPB              msamGetBlock;
    MSAMMarkRecipientsPB        msamMarkRecipients;
    MSAMnMarkRecipientsPB       msamnMarkRecipients;

    MSAMCreatePB                msamCreate;
    MSAMBeginNestedPB           msamBeginNested;
    MSAMEndNestedPB             msamEndNested;
    MSAMSubmitPB                msamSubmit;
    MSAMPutMsgHeaderPB          msamPutMsgHeader;
    MSAMPutAttributePB          msamPutAttribute;
    MSAMPutRecipientPB          msamPutRecipient;
    MSAMPutContentPB            msamPutContent;
    MSAMPutEnclosurePB          msamPutEnclosure;
    MSAMPutBlockPB              msamPutBlock;

    MSAMCreateReportPB          msamCreateReport;
    MSAMPutRecipientReportPB    msamPutRecipientReport;
```

```
    PMSAMCreateMsgSummaryPB        pmsamCreateMsgSummary;
    PMSAMPutMsgSummaryPB           pmsamPutMsgSummary;
    PMSAMGetMsgSummaryPB           pmsamGetMsgSummary;


    MailWakeupPMSAMPB              wakeupPMSAM;
    MailCreateMailSlotPB           createMailSlot;
    MailModifyMailSlotPB           modifyMailSlot;
};

typedef union MSAMParam MSAMParam;
```

MSAM Functions

Initializing an MSAM

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

pascal OSErr PMSAMOpenQueues(MSAMParam *paramBlock);

pascal OSErr SMSAMSetup(MSAMParam *paramBlock);

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

Enumerating Messages in a Queue

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

Opening an Outgoing Message

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

Reading Header Information

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

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

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

Reading a Message

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

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

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

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

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

Marking a Recipient

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

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

Closing a Message

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

Creating, Reading, and Writing Message Summaries

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

Creating a Message

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

Writing Header Information

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

Writing a Message

```
pascal OSErr MSAMPutContent(MSAMParam *paramBlock, Boolean asyncFlag);
pascal OSErr MSAMPutEnclosure(MSAMParam *paramBlock);
pascal OSErr MSAMPutBlock(MSAMParam *paramBlock, Boolean asyncFlag);
pascal OSErr MSAMBeginNested(MSAMParam *paramBlock, Boolean asyncFlag);
pascal OSErr MSAMEndNested(MSAMParam *paramBlock);
```

Submitting a Message

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

Deleting a Message

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

Generating Log Entries and Reports

```
pascal OSErr PMSAMLogError(MSAMParam *paramBlock);
pascal OSErr MSAMCreateReport(MSAMParam *paramBlock, Boolean asyncFlag);
pascal OSErr MSAMPutRecipientReport(MSAMParam *paramBlock, Boolean asyncFlag);
```

Shutting Down a Server MSAM

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

Setting Message Status

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

Personal MSAM AOCE Template Functions

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

Application-Defined Function

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

Pascal Summary

Data Types and Constants

```
CONST
{ predefined message creator and message type }
kMailAppleMailCreator       = 'apml'; { message creator }
kMailLtrMsgType             = 'lttr'; { message type for letter, report }

{ predefined block creator and block types }
kMailAppleMailCreator       = 'apml'; { block creator }
kMailLtrHdrType             = 'lthd'; { letter header }
kMailContentType            = 'body'; { content of letter }
kMailEnclosureListType      = 'elst'; { list of enclosures }
kMailEnclosureDesktopType   = 'edsk'; { Desktop Mgr info for enclosures }
kMailEnclosureFileType      = 'asgl'; { a file enclosure }
kMailImageBodyType          = 'imag'; { image of letter }
kMailMSAMType               = 'gwyi'; { MSAM-specific information }
kMailReportType             = 'rpti'; { report info }

{ families used for mail or related msgs }
kMailFamily                 = 'mail'; { letter with content block or
                                        content enclosure }
kMailFamilyFile             = 'file'; { letter without content block or
                                        content enclosure }

kMailResolvedList           = 0;      { MailAttributeID value for resolved
                                        recipient list }
```

```
{ bit flags of MailAttributeID type }
kMailLetterFlagsBit      = 1;  { letter flags bit }
kMailIndicationsBit      = 3;  { indications bit }
kMailMsgTypeBit          = 4;  { letter creator & type bit }
kMailLetterIDBit         = 5;  { letter ID bit }
kMailSendTimeStampBit    = 6;  { send timestamp bit }
kMailNestingLevelBit     = 7;  { nesting level bit }
kMailMsgFamilyBit        = 8;  { message family bit }
kMailReplyIDBit          = 9;  { reply ID bit }
kMailConversationIDBit   = 10; { conversation ID bit }
kMailSubjectBit          = 11; { subject bit }
kMailFromBit             = 12; { From recipient bit }
kMailToBit               = 13; { To recipient bit }
kMailCcBit               = 14; { cc recipient bit }
kMailBccBit              = 15; { bcc recipient bit }

{ Values of MailAttributeMask data type. The masks are defined for use
 with the MailAttributeBitmap data type. However, because the
 MailAttributeBitmap data type is defined as a bit field structure, and the
 masks operate on variables of type LONGINT, you cannot use these masks to
 set or test the value of a bit field in a MailAttributeBitmap structure. The
 masks are included for historical reasons only. }
kMailLetterFlagsMask       = $00000001;{1<<(kMailLetterFlagsBit-1)}
kMailIndicationsMask       = $00000004;{1<<(kMailIndicationsBit-1)}
kMailMsgTypeMask           = $00000008;{1<<(kMailMsgTypeBit-1)}
kMailLetterIDMask          = $00000010;{1<<(kMailLetterIDBit-1)}
kMailSendTimeStampMask     = $00000020;{1<<(kMailSendTimeStampBit-1)}
kMailNestingLevelMask      = $00000040;{1<<(kMailNestingLevelBit-1)}
kMailMsgFamilyMask         = $00000080;{1<<(kMailMsgFamilyBit-1)}
kMailReplyIDMask           = $00000100;{1<<(kMailReplyIDBit-1)}
kMailConversationIDMask    = $00000200;{1<<(kMailConversationIDBit-1)}
kMailSubjectMask           = $00000400;{1<<(kMailSubjectBit-1)}
kMailFromMask              = $00000800;{1<<(kMailFromBit-1)}
kMailToMask                = $00001000;{1<<(kMailToBit-1)}
kMailCcMask                = $00002000;{1<<(kMailCcBit-1)}
kMailBccMask               = $00004000;{1<<(kMailBccBit-1)}

{ bit flags of MailIndications type }
kMailOriginalInReportBit   = 1;
kMailNonReceiptReportsBit  = 3;
kMailReceiptReportsBit     = 4;
kMailForwardedBit          = 5;
kMailPriorityBit           = 6;
kMailIsReportWithOriginalBit = 8;
```

```
kMailIsReportBit              = 9;
kMailHasContentBit            = 10;
kMailHasSignatureBit          = 11;
kMailAuthenticatedBit         = 12;
kMailSentBit                  = 13;

{ Masks for bits of MailIndications type. Because the MailIndications data
 type is defined as a bit field structure, and the masks operate on variables
 of type LONGINT, you cannot use the masks to set or test the value of a bit
 field in a MailIndications structure. The masks are included for
 historical reasons only}
kMailSentMask                 = $00001000; {1<<(kMailSentBit-1),
kMailAuthenticatedMask        = $00000800; {1<<(kMailAuthenticatedBit-1),
kMailHasSignatureMask         = $00000400; {1<<(kMailHasSignatureBit-1),
kMailHasContentMask           = $00000200; {1<<(kMailHasContentBit-1),
kMailIsReportMask             = $00000100; {1<<(kMailIsReportBit-1),
kMailIsReportWithOriginalMask = $00000080;
                                     {1<<(kMailIsReportWithOriginalBit-1),
kMailPriorityMask             = $00000060; {3<<(kMailPriorityBit-1),
kMailForwardedMask            = $00000010; {1<<(kMailForwardedBit-1),
kMailReceiptReportsMask       = $00000008; {1<<(kMailReceiptReportsBit-1),
kMailNonReceiptReportsMask    = $00000004; {1<<(kMailNonReceiptReportsBit-1),
kMailOriginalInReportMask     = $00000003; {3<<(kMailOriginalInReportBit-1);

{ Bit values of the originalInReport field in MailIndications }
kMailNoOriginal        = 0;  { do not enclose original in report }
kMailEncloseOnNonReceipt= 3;  { enclose original in non-delivery
                              indication }

{ values of MailSegmentType type}
kMailInvalidSegmentType    = 0;
kMailTextSegmentType       = 1;
kMailPictSegmentType       = 2;
kMailSoundSegmentType      = 3;
kMailStyledTextSegmentType = 4;
kMailMovieSegmentType      = 5;

kMailTextSegmentBit        = 0;
kMailPictSegmentBit        = 1;
kMailSoundSegmentBit       = 2;
kMailStyledTextSegmentBit  = 3;
kMailMovieSegmentBit       = 4;
```

```
{ values of MailSegmentMask type }
kMailTextSegmentMask       = $0001; {1<<kMailTextSegmentBit}
kMailPictSegmentMask       = $0002; {1<<kMailPictSegmentBit}
kMailSoundSegmentMask      = $0004; {1<<kMailSoundSegmentBit}
kMailStyledTextSegmentMask = $0008; {1<<kMailStyledTextSegmentBit}
kMailMovieSegmentMask      = $0010; {1<<kMailMovieSegmentBit}

{ values of MailBlockMode type }
kMailFromStart = 1;  { write data at offset from start of block }
kMailFromLEOB  = 2;  { write data at offset from end of block }
kMailFromMark  = 3;  { write data at offset from the current mark }

{ bit values of MailLetterSystemFlags type }
kMailIsLocalBit  = 2;     { letter is available locally }

kMailIsLocalMask  = $0004; {1<<kMailIsLocalBit}

{ bit values of MailLetterUserFlags type }
kMailReadBit        = 0;     { letter has been opened }
kMailDontArchiveBit = 1;     { reserved }
kMailInTrashBit     = 2;     { reserved }

kMailReadMask           = $0001; {1<<kMailReadBit}
kMailDontArchiveMask    = $0002; {1<<kMailDontArchiveBit}
kMailInTrashMask        = $0004; {1<<kMailInTrashBit}

kMailErrorLogEntryVersion = $101;

{ 'STR#' resource IDs for personal MSAM's error and action messages }
kMailMSAMErrorStringListID    =  128; { list of error message strings }
kMailMSAMActionStringListID   =  129; { list of action message strings }

{ values of MailLogErrorType type}
kMailELECorrectable   = 0;  { error correctable by user }
kMailELEError         = 1;  { error not correctable by user }
kMailELEWarning       = 2;  { warning requiring no user intervention }
kMailELEInformational = 3;  { informational message }

{ values of MailLogErrorCode type }
kMailMSAMErrorCode= 0;     { MSAM-defined error }
kMailMiscError   = -1;     { miscellaneous error }
kMailNoModem     = -2;     { modem required, but missing }

kMailMsgSummaryVersion      =  1;
```

```
kMailMaxPMSAMMsgSummaryData   = 128; { maximum bytes for private MSAM
                                       message summary data }

{ defines for the addressedToMe field in MailCoreData }
kAddressedAs_TO   = 1;
kAddressedAs_CC   = 2;
kAddressedAs_BCC  = 4;

kMailTimerOff        = 0;  { no timer specified }
kMailTimerTime       = 1;  { timer relative to midnight }
kMailTimerFrequency  = 2;  { frequency timer }

{ values of PMSAMStatus type }
   kPMSAMStatusPending  = 1;  { for outQueue }
   kPMSAMStatusError    = 2;  { for inQueue letters }
   kPMSAMStatusSending  = 3;  { for outQueue }
   kPMSAMStatusCaching  = 4;  { for inQueue letters }
   kPMSAMStatusSent     = 5;  { for outQueue }

kMailEPPCMsgVersion     = 3;

{ values of AOCE high-level event message classes }
kMailEPPCCreateSlot        = 'crsl';
kMailEPPCModifySlot        = 'mdsl';
kMailEPPCDeleteSlot        = 'dlsl';
kMailEPPCShutDown          = 'quit';
kMailEPPCMailboxOpened     = 'mbop';
kMailEPPCMailboxClosed     = 'mbcl';
kMailEPPCMsgPending        = 'msgp';
kMailEPPCSendImmediate     = 'sndi';
kMailEPPCContinue          = 'cont';
kMailEPPCSchedule          = 'sked';
kMailEPPCAdmin             = 'admn';
kMailEPPCInQUpdate         = 'inqu';
kMailEPPCMsgOpened         = 'msgo';
kMailEPPCDeleteOutQMsg     = 'dlom';
kMailEPPCWakeup            = 'wkup';
kMailEPPCLocationChanged   = 'locc'

{ values of SMSAMAdminCode type }
kSMSAMNotifyFwdrSetupChange= 1;
kSMSAMNotifyFwdrNameChange = 2;
kSMSAMNotifyFwdrPwdChange  = 3;
kSMSAMGetDynamicFwdrParams = 4;
```

```
kSMSAMFwdrHomeInternetChangedBit = 0;
kSMSAMFwdrConnectedToChangedBit  = 1;
kSMSAMFwdrForeignRLIsChangedBit  = 2;
kSMSAMFwdrMnMServerChangedBit    = 3;

{ values of SMSAMSlotChanges type }
kSMSAMFwdrEverythingChangedMask     = -1,
kSMSAMFwdrHomeInternetChangedMask   = $00000001;
                                    {1<<kSMSAMFwdrHomeInternetChangedBit}
kSMSAMFwdrConnectedToChangedMask    = $00000002;
                                    {1<<kSMSAMFwdrConnectedToChangedBit}
kSMSAMFwdrForeignRLIsChangedMask    = $00000004;
                                    {1<<kSMSAMFwdrForeignRLIsChangedBit}
kSMSAMFwdrMnMServerChangedMask      = $00000008;
                                    {1<<kSMSAMFwdrMnMServerChangedBit}

kOCESetupLocationNone  = 0;  { disconnect state }
kOCESetupLocationMax   = 8;  { maximum location value }

TYPE

MailMsgRef         = LONGINT;      { reference to new/open letter or message }

MSAMQueueRef       = LONGINT;      { reference to an open MSAM queue }

MSAMSlotID         = INTEGER;      { slot identifier }

MailSlotID         = INTEGER;      { identifies slots within a mailbox }

MailboxRef         = LONGINT;      { reference to an active mailbox }

MailAttributeID    = INTEGER;      { letter attribute identifier }

{ The MailAttributeMask data type defines a set of masks for the
 MailAttributeBitmap data type. However, because the MailAttributeBitmap data
 type is defined as a bit field structure, and the masks operate on variables
 of type LONGINT, you cannot use the masks to set or test the value of a bit
 field in a MailAttributeBitmap structure. The MailAttributeMask data type is
 included for historical reasons only. }
MailAttributeMask = LONGINT;

MailLetterID       = IPMMsgID;

MailNestingLevel   = INTEGER;

MailRecipient      = OCERecipient;
```

```
MailSegmentMask   = INTEGER;

MailSegmentType   = INTEGER;

MailBlockMode     = INTEGER;

PMSAMStatus       = INTEGER;

OCESetupLocation  =  Byte; { current system location }

MailLocationFlags =  Byte; { slot location flags }

MailBuffer = RECORD
   bufferSize: LONGINT; { size of your buffer }
   buffer:     Ptr;     { pointer to your buffer }
   dataSize:   LONGINT; { amount of data returned in or read out
                          of your buffer }

   END;

MailReply = RECORD
   tupleCount: INTEGER;
   { tuple[1..tupleCount] }
   END;

MSAMEnumerateOutQReply = PACKED RECORD
   seqNum:     LONGINT;          { sequence number of message }
   done:       BOOLEAN;          { resolution of message }
   priority:   IPMPriority;   { priority of message }
   msgFamily:  OSType          { message family }
   approxSize: LONGINT;          { size of message }
   tunnelForm: BOOLEAN;          { reserved }
   padByte:    Byte;             { for even byte boundary }
   nextHop:    NetworkSpec;   { reserved }
   msgType:    OCECreatorType;{ message creator and type }
   END;

MSAMEnumerateInQReply = RECORD
   seqNum:     LONGINT;    { letter sequence number }
   msgDeleted: BOOLEAN;    { should letter be deleted? }
   msgUpdated: BOOLEAN;    { was message summary updated? }
   msgCached:  BOOLEAN;    { is letter in the incoming queue? }
   {padByte:   Byte;}
   END;
```

```
MailAttributeBitmap = PACKED RECORD
   reservedA:      0..65535;{ reserved }
   reservedB:      0..1;    { reserved }
   bcc:            0..1;    { blind carbon copy recipients }
   cc:             0..1;    { carbon copy recipients }
   toRecipient:    0..1;    { to recipients }
   from:           0..1;    { sender of letter }
   subject:        0..1;    { subject of letter }
   conversationID:0..1;     { ID of conversation thread }
   replyID:        0..1;    { ID of letter being replied to }
   msgFamily:      0..1;    { message family }
   nestingLevel:   0..1;    { nesting level of letter }
   sendTimeStamp:  0..1;    { time letter was sent }
   letterID:       0..1;    { letter's unique ID number }
   msgType:        0..1;    { letter's creator and type }
   indications:    0..1;    { MailIndications }
   reservedC:      0..1;    { reserved }
   letterFlags:    0..1;    { letter flags }
   END;

MailIndications = PACKED RECORD
   reservedB:              0..65535;{ reserved }
   hasStandardContent:     0..1; { letter has a content block }
   hasImageContent:        0..1; { letter has an image block }
   hasNativeContent:       0..1; { letter has a content enclosure }
   sent:                   0..1; { letter was sent, not just composed }
   authenticated:          0..1; { letter was created and transported with
                                      authentication }
   hasSignature:           0..1; { letter was signed with digital signature }
   hasContent:             0..1; { this letter or nested letter has content }
   isReport:               0..1; { letter is a report }
   isReportWithOriginal:   0..1; { report contains the original letter }
   priority:               0..3; { letter has normal, low, or high priority }
   forwarded:              0..1; { letter contains a forwarded letter }
   receiptReports:         0..1; { originator requests delivery indications }
   nonReceiptReports:      0..1; { originator requests non-delivery
                                      indications }
   originalInReport:       0..3; { originator wants original letter
                                     enclosed in reports }
   END;
```

```
OCERecipient = RECORD
   entitySpecifier:  ^RecordID;
   extensionType:    OSType;
   extensionSize:    INTEGER;
   extensionValue:   Ptr;
   END;


OCEPackedRecipient = RECORD
   dataLength: INTEGER; { length of recipient data }
   data:       PACKED ARRAY[1..kPackedDSSpecMaxBytes] OF Byte;
   END;


MailOriginalRecipient = RECORD
   index: INTEGER;   { index for recipient }
   { Followed by OCEPackedRecipient }
   END;


MailResolvedRecipient = PACKED RECORD
   index:            INTEGER; { index for recipient }
   recipientFlags:   INTEGER; { recipient information }
   responsible:      BOOLEAN; { responsible for delivery? }
   padByte:          Byte;
        { followed by OCEPackedRecipient }
   END;


MailEnclosureInfo = RECORD
   enclosureName: StringPtr;  { name of the enclosure }
   catInfo:       CInfoPBPtr; { HFS catalog info about enclosure }
   comment:       StringPtr;  { comment for Get-Info window }
   icon:          Ptr;        { icon for enclosure file }
   END;


MailLogErrorType = INTEGER;


MailLogErrorCode = INTEGER;


MailErrorLogEntryInfo = RECORD
   version:            INTEGER;          { log entry version }
   timeOccurred:       UTCTime;          { time of error }
   reportingPMSAM:     Str31;            { which MSAM? }
   reportingMSAMSlot:  Str31;            { which slot? }
   errorType:          MailLogErrorType; { level of error }
   errorCode:          MailLogErrorCode; { error code }
```

```
    errorResource:      INTEGER;              { error string resource index }
    actionResource:     INTEGER;              { action string resource index }
    filler:             LONGINT;              { reserved }
    filler2:            INTEGER;              { reserved }
    END;

MailMasterData = RECORD
    attrMask:           MailAttributeBitmap;  { indicates attributes present in
                                                MSAMMsgSummary }
    messageID:          MailLetterID;         { ID of this letter }
    replyID:            MailLetterID;         { ID of letter this is a reply to }
    conversationID:     MailLetterID;         { ID of letter that started this
                                                conversation}

    END;

MailCoreData = RECORD
    letterFlags:        MailLetterFlags;      { letter status flags }
    messageSize:        LONGINT;              { size of letter }
    letterIndications:  MailIndications;      { indications for this letter }
    messageType:        OCECreatorType;       { message creator and type of
                                                 this letter }
    sendTime:           MailTime;             { time this letter was sent }
    messageFamily:      OSType;               { message family }
    reserved            unsigned char;
    addressedToMe       unsigned char;
    agentInfo:          ARRAY[1..2] OF Byte; { reserved }
    { sender and subject are variable length and even padded }
    sender:             RString32;            { sender of this letter }
    subject:            RString32;            { subject of this letter }
    END;

MSAMMsgSummary = RECORD
    version:    INTEGER;     { version of the MSAMMsgSummary }
    msgDeleted: BOOLEAN;     { true if letter is to be deleted by MSAM }
    msgUpdated: BOOLEAN;     { true if MSAMMsgSummary was updated by IPM
                               Manager }
    msgCached:  BOOLEAN;     { true if letter is in the incoming queue }
    {padByte:    Byte;}
    masterData: MailMasterData;   { attributes not essential to display }
    coreData:   MailCoreData;     { attributes critical to display }
    { followed by the personal MSAM's private data }
    END;
```

```
MailLocationInfo = RECORD
   location:   OCESetupLocation;    { the current location }
   active:     MailLocationFlags;   { slot's location flags }
   END;

MailEPPCMsg = RECORD
   version:    INTEGER; { message version }
   CASE INTEGER OF
      1. (smca:            ^SMCA);              { pointer to SMCA }
      2. (sequenceNumber:  LONGINT);            { letter sequence number }
      3. (locationInfo:    MailLocationInfo);{ location information }
   END;

SMCA = RECORD
   smcaLength: INTEGER;             { length of entire SMCA, including size of
                                      smcaLength field }
   result:     OSErr;              { result code }
   userBytes:  LONGINT;            { command interpreted user data }
   CASE INTEGER OF
      1: (slotCID: CreationID);   { creation ID of record
                                    containing slot information }
      2: (msgHint: LONGINT);      { message reference value }
   END;

SMSAMAdminCode = INTEGER;

SMSAMAdminEPPCRequest = RECORD
   adminCode: SMSAMAdminCode;                       { admin code }
   CASE INTEGER OF
      1: (setupChange:     SMSAMSetupChange);       { setup change }
      2: (nameChange:      SMSAMNameChange);        { reserved }
      3: (passwordChange:  SMSAMPasswordChange);    { reserved }
      4: (dynamicParams:   SMSAMDynamicParams);     { reserved }
   END;

SMSAMSlotChanges = LONGINT;

SMSAMSetupChange = RECORD
   whatChanged:    SMSAMSlotChanges; { bitmap of changed parameters }
   serverHint:     AddrBlock;        { AOCE server address }
   END;
```

```
SMSAMNameChange = RECORD              { reserved data type }
   newName:    RString;              { server MSAM's new name }
   serverHint: AddrBlock;            { AOCE server address }
   END;

SMSAMPasswordChange = RECORD      { reserved data type }
   newPassword:   RString;       { server MSAM's new password }
   serverHint:    AddrBlock;     { AOCE server address }
   END;

SMSAMDynamicParams = RECORD       { reserved data type }
   curDiskUsed:    LONGINT;       { disk space used }
   curMemoryUsed: LONGINT;        { memory used }
   END;

MailTime = RECORD
   time:    UTCTime;              { current UTC(GMT) }
   offset: UTCOffset;             { offset from UTC }
   END;

MailTimer = RECORD
   CASE INTEGER OF
      1: (frequency: LONGINT);    { how often to connect }
      2: (connectTime: LONGINT); { time since midnight }
   END;

MailTimerKind = Byte;

MailTimers = PACKED RECORD
   sendTimeKind:    MailTimerKind;    { timer kind for sending }
   receiveTimeKind: MailTimerKind;    { timer kind for receiving }
   send:            MailTimer;        { connect time or frequency
                                        for sending letters }
   receive:         MailTimer;        { connect time or frequency
                                        for sending letters }
   END;

MailStandardSlotInfoAttribute = PACKED RECORD
   version:           INTEGER;             { MSAM version of the slot }
   active:            MailLocationFlags;  { active at location i if
                                             MailLocation Mask(i) is set }
   padByte:           Byte;
   sendReceiveTimer: MailTimers;
   END;
```

```
MailLetterSystemFlags= INTEGER;

MailLetterUserFlags  = INTEGER;

MailLetterFlags = RECORD
   sysFlags:  MailLetterSystemFlags;  { system flags }
   userFlags: MailLetterUserFlags;    { user flags }
   END;

MailMaskedLetterFlags = RECORD
   flagMask:   MailLetterFlags;  { flags that are to be set }
   flagValues: MailLetterFlags;  { their values }
   END;

MailBlockInfo = RECORD
   blockType:    OCECreatorType;
   offset:       LONGINT;
   blockLength:  LONGINT;
   END;

MailParamBlockHeader = RECORD
   qLink:         Ptr;       { next queue entry }
   reservedH1:    LONGINT;   { reserved }
   reservedH2:    LONGINT;   { reserved }
   ioCompletion:  ProcPtr;   { pointer to completion routine }
   ioResult:      OSErr;     { result code }
   saveA5:        LONGINT;   { pointer to global variables }
   reqCode:       INTEGER;   { reserved }
   END;

PMSAMGetMSAMRecordPB = RECORD
   qLink:         Ptr;       { next queue entry }
   reservedH1:    LONGINT;   { reserved }
   reservedH2:    LONGINT;   { reserved }
   ioCompletion:  ProcPtr;   { pointer to completion routine }
   ioResult:      OSErr;     { result code }
   saveA5:        LONGINT;   { pointer to global variables }
   reqCode:       INTEGER;   { reserved }
   msamCID:       CreationID;
   END;

PMSAMOpenQueuesPB = RECORD
   qLink:         Ptr;       { next queue entry }
   reservedH1:    LONGINT;   { reserved }
   reservedH2:    LONGINT;   { reserved }
```

```
    ioCompletion:  ProcPtr;     { pointer to completion routine }
    ioResult:      OSErr;       { result code }
    saveA5:        LONGINT;     { pointer to global variables }
    reqCode:       INTEGER;     { reserved }
    inQueueRef:    MSAMQueueRef;
    outQueueRef:   MSAMQueueRef;
    msamSlotID:    MSAMSlotID;
    filler:        ARRAY[1..2] OF LONGINT;
    END;

PMSAMSetStatusPB = RECORD
    qLink:         Ptr;         { next queue entry }
    reservedH1:    LONGINT;     { reserved }
    reservedH2:    LONGINT;     { reserved }
    ioCompletion:  ProcPtr;     { pointer to completion routine }
    ioResult:      OSErr;       { result code }
    saveA5:        LONGINT;     { pointer to global variables }
    reqCode:       INTEGER;     { reserved }
    queueRef:      MSAMQueueRef;
    seqNum:        LONGINT;
    msgHint:       LONGINT;
    status:        PMSAMStatus;
    END;

PMSAMLogErrorPB = RECORD
    qLink:         Ptr;         { next queue entry }
    reservedH1:    LONGINT;     { reserved }
    reservedH2:    LONGINT;     { reserved }
    ioCompletion:  ProcPtr;     { pointer to completion routine }
    ioResult:      OSErr;       { result code }
    saveA5:        LONGINT;     { pointer to global variables }
    reqCode:       INTEGER;     { reserved }
    msamSlotID:    MSAMSlotID;
    logEntry:      ^MailErrorLogEntryInfo;
    filler:        ARRAY[1..2] OF LONGINT;
    END;

PMSAMCreateMsgSummaryPB = RECORD
    qLink:         Ptr;         { next queue entry }
    reservedH1:    LONGINT;     { reserved }
    reservedH2:    LONGINT;     { reserved }
    ioCompletion:  ProcPtr;     { pointer to completion routine }
    ioResult:      OSErr;       { result code }
    saveA5:        LONGINT;     { pointer to global variables }
```

```
   reqCode:        INTEGER;    { reserved }
   inQueueRef:     MSAMQueueRef;
   seqNum:         LONGINT;      { seq of the new letter }
   msgSummary:     ^MSAMMsgSummary;
   buffer:         ^MailBuffer;{ PMSAM specific data }
   END;

PMSAMPutMsgSummaryPB = RECORD
   qLink:          Ptr;          { next queue entry }
   reservedH1:     LONGINT;      { reserved }
   reservedH2:     LONGINT;      { reserved }
   ioCompletion:   ProcPtr;      { pointer to completion routine }
   ioResult:       OSErr;        { result code }
   saveA5:         LONGINT;      { pointer to global variables }
   reqCode:        INTEGER;      { reserved }
   inQueueRef:     MSAMQueueRef;
   seqNum:         LONGINT;
   letterFlags:    ^MailMaskedLetterFlags;
   buffer:         ^MailBuffer;  { PMSAM private data }
   END;

PMSAMGetMsgSummaryPB = RECORD
   qLink:          Ptr;          { next queue entry }
   reservedH1:     LONGINT;      { reserved }
   reservedH2:     LONGINT;      { reserved }
   ioCompletion:   ProcPtr;      { pointer to completion routine }
   ioResult:       OSErr;        { result code }
   saveA5:         LONGINT;      { pointer to global variables }
   reqCode:        INTEGER;      { reserved }
   inQueueRef:     MSAMQueueRef;
   seqNum:         LONGINT;
   msgSummary:     ^MSAMMsgSummary;
   buffer:         ^MailBuffer;{ PMSAM private data }
   msgSummaryOffset: INTEGER;    { offset of PMSAM private data  }
                                 { from start of MsgSummary }
   END;

SMSAMSetupPB = RECORD
   qLink:          Ptr;          { next queue entry }
   reservedH1:     LONGINT;      { reserved }
   reservedH2:     LONGINT;      { reserved }
   ioCompletion:   ProcPtr;      { pointer to completion routine }
   ioResult:       OSErr;        { result code }
   saveA5:         LONGINT;      { pointer to global variables }
```

```
    reqCode:             INTEGER;     { reserved }
    serverMSAM          RecordIDPtr;
    password          RStringPtr;
    gatewayType       OSType;
    gatewayTypeDescription  RStringPtr;
    catalogServerHint       AddrBlock;
    END;

SMSAMStartupPB = RECORD
    qLink:          Ptr;          { next queue entry }
    reservedH1:     LONGINT;      { reserved }
    reservedH2:     LONGINT;      { reserved }
    ioCompletion:   ProcPtr;      { pointer to completion routine }
    ioResult:       OSErr;        { result code }
    saveA5:         LONGINT;      { pointer to global variables }
    reqCode:        INTEGER;      { reserved }
    msamIdentity:   AuthIdentity;
    queueRef:       MSAMQueueRef;
    END;

SMSAMShutdownPB = RECORD
    qLink:          Ptr;          { next queue entry }
    reservedH1:     LONGINT;      { reserved }
    reservedH2:     LONGINT;      { reserved }
    ioCompletion:   ProcPtr;      { pointer to completion routine }
    ioResult:       OSErr;        { result code }
    saveA5:         LONGINT;      { pointer to global variables }
    reqCode:        INTEGER;      { reserved }
    queueRef:       MSAMQueueRef;
    END;

MSAMEnumeratePB = RECORD
    qLink:          Ptr;          { next queue entry }
    reservedH1:     LONGINT;      { reserved }
    reservedH2:     LONGINT;      { reserved }
    ioCompletion:   ProcPtr;      { pointer to completion routine }
    ioResult:       OSErr;        { result code }
    saveA5:         LONGINT;      { pointer to global variables }
    reqCode:        INTEGER;      { reserved }
    queueRef:       MSAMQueueRef;
    startSeqNum:    LONGINT;
    nextSeqNum:     LONGINT;
    buffer:         MailBuffer;
    END;
```

```
MSAMDeletePB = PACKED RECORD
    qLink:          Ptr;            { next queue entry }
    reservedH1:     LONGINT;        { reserved }
    reservedH2:     LONGINT;        { reserved }
    ioCompletion:   ProcPtr;        { pointer to completion routine }
    ioResult:       OSErr;          { result code }
    saveA5:         LONGINT;        { pointer to global variables }
    reqCode:        INTEGER;        { reserved }
    queueRef:       MSAMQueueRef;
    seqNum:         LONGINT;
    msgOnly:        BOOLEAN;        { only valid for PMSAM & inQueue }
    padByte:        Byte;
    result:         OSErr;          { reserved }
    END;

MSAMOpenPB = RECORD
    qLink:          Ptr;            { next queue entry }
    reservedH1:     LONGINT;        { reserved }
    reservedH2:     LONGINT;        { reserved }
    ioCompletion:   ProcPtr;        { pointer to completion routine }
    ioResult:       OSErr;          { result code }
    saveA5:         LONGINT;        { pointer to global variables }
    reqCode:        INTEGER;        { reserved }
    queueRef:       MSAMQueueRef;
    seqNum:         LONGINT;
    mailMsgRef:     MailMsgRef;
    END;

MSAMOpenNestedPB = RECORD
    qLink:          Ptr;            { next queue entry }
    reservedH1:     LONGINT;        { reserved }
    reservedH2:     LONGINT;        { reserved }
    ioCompletion:   ProcPtr;        { pointer to completion routine }
    ioResult:       OSErr;          { result code }
    saveA5:         LONGINT;        { pointer to global variables }
    reqCode:        INTEGER;        { reserved }
    mailMsgRef:     MailMsgRef;
    nestedRef:      MailMsgRef;
    END;

MSAMClosePB = RECORD
    qLink:          Ptr;            { next queue entry }
    reservedH1:     LONGINT;        { reserved }
    reservedH2:     LONGINT;        { reserved }
```

```
    ioCompletion:   ProcPtr;      { pointer to completion routine }
    ioResult:       OSErr;        { result code }
    saveA5:         LONGINT;      { pointer to global variables }
    reqCode:        INTEGER;      { reserved }
    mailMsgRef:     MailMsgRef;
    END;

MSAMGetMsgHeaderPB = RECORD
    qLink:          Ptr;          { next queue entry }
    reservedH1:     LONGINT;      { reserved }
    reservedH2:     LONGINT;      { reserved }
    ioCompletion:   ProcPtr;      { pointer to completion routine }
    ioResult:       OSErr;        { result code }
    saveA5:         LONGINT;      { pointer to global variables }
    reqCode:        INTEGER;      { reserved }
    mailMsgRef:     MailMsgRef;
    selector:       IPMHeaderSelector;
    offset:         LONGINT;
    buffer:         MailBuffer;
    remaining:      LONGINT;
    END;

MSAMGetAttributesPB = RECORD
    qLink:          Ptr;          { next queue entry }
    reservedH1:     LONGINT;      { reserved }
    reservedH2:     LONGINT;      { reserved }
    ioCompletion:   ProcPtr;      { pointer to completion routine }
    ioResult:       OSErr;        { result code }
    saveA5:         LONGINT;      { pointer to global variables }
    reqCode:        INTEGER;      { reserved }
    mailMsgRef:     MailMsgRef;
    requestMask:    MailAttributeBitmap;
    buffer:         MailBuffer;
    responseMask:   MailAttributeBitmap;
    more:           BOOLEAN;
    END;

MSAMGetRecipientsPB = RECORD
    qLink:          Ptr;          { next queue entry }
    reservedH1:     LONGINT;      { reserved }
    reservedH2:     LONGINT;      { reserved }
    ioCompletion:   ProcPtr;      { pointer to completion routine }
    ioResult:       OSErr;        { result code }
    saveA5:         LONGINT;      { pointer to global variables }
```

```
    reqCode:         INTEGER;      { reserved }
    mailMsgRef:      MailMsgRef;
    attrID:          MailAttributeID;
    startIndex:      INTEGER;
    buffer:          MailBuffer;
    nextIndex:       INTEGER;
    more:            BOOLEAN;
    END;

MSAMGetContentPB = RECORD
    qLink:           Ptr;          { next queue entry }
    reservedH1:      LONGINT;      { reserved }
    reservedH2:      LONGINT;      { reserved }
    ioCompletion:    ProcPtr;      { pointer to completion routine }
    ioResult:        OSErr;        { result code }
    saveA5:          LONGINT;      { pointer to global variables }
    reqCode:         INTEGER;      { reserved }
    mailMsgRef:      MailMsgRef;
    segmentMask:     MailSegmentMask;
    buffer:          MailBuffer;
    textScrap:       ^StScrpRec;
    script:          ScriptCode;
    segmentType:     MailSegmentType;
    endOfScript:     BOOLEAN;
    endOfSegment:    BOOLEAN;
    endOfContent:    BOOLEAN;
    segmentLength:   LONGINT;
    segmentID:       LONGINT;
    END;

MSAMGetEnclosurePB = PACKED RECORD
    qLink:             Ptr;          { next queue entry }
    reservedH1:        LONGINT;      { reserved }
    reservedH2:        LONGINT;      { reserved }
    ioCompletion:      ProcPtr;      { pointer to completion routine }
    ioResult:          OSErr;        { result code }
    saveA5:            LONGINT;      { pointer to global variables }
    reqCode:           INTEGER;      { reserved }
    mailMsgRef:        MailMsgRef;
    contentEnclosure:  BOOLEAN;
    padByte:           Byte;
    buffer:            MailBuffer;
    endOfFile:         BOOLEAN;
    endOfEnclosures:   BOOLEAN;
    END;
```

```
MSAMEnumerateBlocksPB = RECORD
   qLink:         Ptr;         { next queue entry }
   reservedH1:    LONGINT;     { reserved }
   reservedH2:    LONGINT;     { reserved }
   ioCompletion:  ProcPtr;     { pointer to completion routine }
   ioResult:      OSErr;       { result code }
   saveA5:        LONGINT;     { pointer to global variables }
   reqCode:       INTEGER;     { reserved }
   mailMsgRef:    MailMsgRef;
   startIndex:    INTEGER;     { starts at 1 }
   buffer:        MailBuffer;
   nextIndex:     INTEGER;
   more:          BOOLEAN;
   END;

MSAMGetBlockPB = PACKED RECORD
   qLink:         Ptr;         { next queue entry }
   reservedH1:    LONGINT;     { reserved }
   reservedH2:    LONGINT;     { reserved }
   ioCompletion:  ProcPtr;     { pointer to completion routine }
   ioResult:      OSErr;       { result code }
   saveA5:        LONGINT;     { pointer to global variables }
   reqCode:       INTEGER;     { reserved }
   mailMsgRef:    MailMsgRef;
   blockType:     OCECreatorType;
   blockIndex:    INTEGER;
   buffer:        MailBuffer;
   dataOffset:    LONGINT;
   endOfBlock:    BOOLEAN;
   padByte:       Byte;
   remaining:     LONGINT;
   END;

MSAMMarkRecipientsPB = RECORD
   qLink:         Ptr;         { next queue entry }
   reservedH1:    LONGINT;     { reserved }
   reservedH2:    LONGINT;     { reserved }
   ioCompletion:  ProcPtr;     { pointer to completion routine }
   ioResult:      OSErr;       { result code }
   saveA5:        LONGINT;     { pointer to global variables }
   reqCode:       INTEGER;     { reserved }
   queueRef:      MSAMQueueRef;
   seqNum:        LONGINT;
   buffer:        MailBuffer;
   END;
```

```
MSAMnMarkRecipientsPB = RECORD
   qLink:         Ptr;          { next queue entry }
   reservedH1:    LONGINT;      { reserved }
   reservedH2:    LONGINT;      { reserved }
   ioCompletion:  ProcPtr;      { pointer to completion routine }
   ioResult:      OSErr;        { result code }
   saveA5:        LONGINT;      { pointer to global variables }
   reqCode:       INTEGER;      { reserved }
   mailMsgRef:    MailMsgRef;
   buffer:        MailBuffer;
   END;

MSAMCreatePB = RECORD
   qLink:         Ptr;          { next queue entry }
   reservedH1:    LONGINT;      { reserved }
   reservedH2:    LONGINT;      { reserved }
   ioCompletion:  ProcPtr;      { pointer to completion routine }
   ioResult:      OSErr;        { result code }
   saveA5:        LONGINT;      { pointer to global variables }
   reqCode:       INTEGER;      { reserved }
   queueRef:      MSAMQueueRef;
   asLetter:      BOOLEAN;      { create as letter or message? }
   msgType:       IPMMsgType;
   refCon:        LONGINT;      { for non-letter messages only }
   seqNum:        LONGINT;
   tunnelForm:    BOOLEAN;      { always false }
   bccRecipients: BOOLEAN;      { true if creating letter with bcc recipients }
   newRef:        MailMsgRef;
   END;

MSAMBeginNestedPB = RECORD
   qLink:         Ptr;          { next queue entry }
   reservedH1:    LONGINT;      { reserved }
   reservedH2:    LONGINT;      { reserved }
   ioCompletion:  ProcPtr;      { pointer to completion routine }
   ioResult:      OSErr;        { result code }
   saveA5:        LONGINT;      { pointer to global variables }
   reqCode:       INTEGER;      { reserved }
   mailMsgRef:    MailMsgRef;
   refCon:        LONGINT;      { for messages only }
   msgType:       IPMMsgType;
   END;
```

```
MSAMEndNestedPB = RECORD
   qLink:         Ptr;          { next queue entry }
   reservedH1:    LONGINT;      { reserved }
   reservedH2:    LONGINT;      { reserved }
   ioCompletion:  ProcPtr;      { pointer to completion routine }
   ioResult:      OSErr;        { result code }
   saveA5:        LONGINT;      { pointer to global variables }
   reqCode:       INTEGER;      { reserved }
   mailMsgRef:    MailMsgRef;
   END;

MSAMSubmitPB = PACKED RECORD
   qLink:         Ptr;          { next queue entry }
   reservedH1:    LONGINT;      { reserved }
   reservedH2:    LONGINT;      { reserved }
   ioCompletion:  ProcPtr;      { pointer to completion routine }
   ioResult:      OSErr;        { result code }
   saveA5:        LONGINT;      { pointer to global variables }
   reqCode:       INTEGER;      { reserved }
   mailMsgRef:    MailMsgRef;   { message reference number }
   submitFlag:    BOOLEAN;      { submit or delete message? }
   padByte:       Byte;
   msgID:         MailLetterID; { reserved }
   END;

MSAMPutMsgHeaderPB = PACKED RECORD
   qLink:         Ptr;          { next queue entry }
   reservedH1:    LONGINT;      { reserved }
   reservedH2:    LONGINT;      { reserved }
   ioCompletion:  ProcPtr;      { pointer to completion routine }
   ioResult:      OSErr;        { result code }
   saveA5:        LONGINT;      { pointer to global variables }
   reqCode:       INTEGER;      { reserved }
   mailMsgRef:    MailMsgRef;
   replyQueue:    ^OCERecipient;
   sender:        ^IPMSender;
   deliveryNotification: IPMNotificationType;
   priority:      IPMPriority;
   END;
```

```
MSAMPutAttributePB = RECORD
   qLink:         Ptr;           { next queue entry }
   reservedH1:    LONGINT;       { reserved }
   reservedH2:    LONGINT;       { reserved }
   ioCompletion:  ProcPtr;       { pointer to completion routine }
   ioResult:      OSErr;         { result code }
   saveA5:        LONGINT;       { pointer to global variables }
   reqCode:       INTEGER;       { reserved }
   mailMsgRef:    MailMsgRef;
   attrID:        MailAttributeID;
   buffer:        MailBuffer;
   END;

MSAMPutRecipientPB = RECORD
   qLink:         Ptr;           { next queue entry }
   reservedH1:    LONGINT;       { reserved }
   reservedH2:    LONGINT;       { reserved }
   ioCompletion:  ProcPtr;       { pointer to completion routine }
   ioResult:      OSErr;         { result code }
   saveA5:        LONGINT;       { pointer to global variables }
   reqCode:       INTEGER;       { reserved }
   mailMsgRef:    MailMsgRef;
   attrID:        MailAttributeID;
   recipient:     ^MailRecipient;
   responsible:   BOOLEAN;       { for server and message msams only }
   END;

MSAMPutContentPB = PACKED RECORD
   qLink:         Ptr;           { next queue entry }
   reservedH1:    LONGINT;       { reserved }
   reservedH2:    LONGINT;       { reserved }
   ioCompletion:  ProcPtr;       { pointer to completion routine }
   ioResult:      OSErr;         { result code }
   saveA5:        LONGINT;       { pointer to global variables }
   reqCode:       INTEGER;       { reserved }
   mailMsgRef:    MailMsgRef;
   segmentType:   MailSegmentType;
   append:        BOOLEAN;
   padByte:       Byte;
   buffer:        MailBuffer;
   textScrap:     ^StScrpRec;
   startNewScript:BOOLEAN;
   script:        ScriptCode;
   END;
```

```
MSAMPutEnclosurePB = RECORD
   qLink:            Ptr;          { next queue entry }
   reservedH1:       LONGINT;      { reserved }
   reservedH2:       LONGINT;      { reserved }
   ioCompletion:     ProcPtr;      { pointer to completion routine }
   ioResult:         OSErr;        { result code }
   saveA5:           LONGINT;      { pointer to global variables }
   reqCode:          INTEGER;      { reserved }
   mailMsgRef:       MailMsgRef;
   contentEnclosure: BOOLEAN;
   padByte:          BOOLEAN;
   hfs:              BOOLEAN;      { true = in file system, false = in memory }
   append:           BOOLEAN;
   buffer:           MailBuffer;
   enclosure:        FSSpec;
   addlInfo:         MailEnclosureInfo;
   END;

MSAMPutBlockPB = RECORD
   qLink:            Ptr;          { next queue entry }
   reservedH1:       LONGINT;      { reserved }
   reservedH2:       LONGINT;      { reserved }
   ioCompletion:     ProcPtr;      { pointer to completion routine }
   ioResult:         OSErr;        { result code }
   saveA5:           LONGINT;      { pointer to global variables }
   reqCode:          INTEGER;      { reserved }
   mailMsgRef:       MailMsgRef;
   refCon:           LONGINT;      { for messages only }
   blockType:        OCECreatorType;
   append:           BOOLEAN;
   buffer:           MailBuffer;
   mode:             MailBlockMode;
   offset:           LONGINT;
   END;

MSAMCreateReportPB = RECORD
   qLink:            Ptr;          { next queue entry }
   reservedH1:       LONGINT;      { reserved }
   reservedH2:       LONGINT;      { reserved }
   ioCompletion:     ProcPtr;      { pointer to completion routine }
   ioResult:         OSErr;        { result code }
   saveA5:           LONGINT;      { pointer to global variables }
   reqCode:          INTEGER;      { reserved }
   queueRef:         MSAMQueueRef; { to distinguish personal and server MSAMs }
```

```
    mailMsgRef:      MailMsgRef;
    msgID:           MailLetterID;  { of letter being reported upon }
    sender:          ^MailRecipient;{ sender of the letter you're reporting on }
    END;

MSAMPutRecipientReportPB = RECORD
    qLink:           Ptr;          { next queue entry }
    reservedH1:      LONGINT;      { reserved }
    reservedH2:      LONGINT;      { reserved }
    ioCompletion:    ProcPtr;      { pointer to completion routine }
    ioResult:        OSErr;        { result code }
    saveA5:          LONGINT;      { pointer to global variables }
    reqCode:         INTEGER;      { reserved }
    mailMsgRef:      MailMsgRef;
    recipientIndex:  INTEGER;      { recipient index in the original letter }
    result:          OSErr;        { result of sending the recipient }
    END;

MailWakeupPMSAMPB = RECORD
    qLink:           Ptr;          { next queue entry }
    reservedH1:      LONGINT;      { reserved }
    reservedH2:      LONGINT;      { reserved }
    ioCompletion:    ProcPtr;      { pointer to completion routine }
    ioResult:        OSErr;        { result code }
    saveA5:          LONGINT;      { pointer to global variables }
    reqCode:         INTEGER;      { reserved }
    pmsamCID:        CreationID;
    mailSlotID:      MailSlotID;
    END;

MailCreateMailSlotPB = RECORD
    qLink:           Ptr;          { next queue entry }
    reservedH1:      LONGINT;      { reserved }
    reservedH2:      LONGINT;      { reserved }
    ioCompletion:    ProcPtr;      { pointer to completion routine }
    ioResult:        OSErr;        { result code }
    saveA5:          LONGINT;      { pointer to global variables }
    reqCode:         INTEGER;      { reserved }
    mailboxRef:      MailboxRef;
    timeout:         LONGINT;
    pmsamCID:        CreationID;
    smca:            SMCA;
    END;
```

```
MailModifyMailSlotPB = RECORD
   qLink:         Ptr;        { next queue entry }
   reservedH1:    LONGINT;    { reserved }
   reservedH2:    LONGINT;    { reserved }
   ioCompletion:  ProcPtr;    { pointer to completion routine }
   ioResult:      OSErr;      { result code }
   saveA5:        LONGINT;    { pointer to global variables }
   reqCode:       INTEGER;    { reserved }
   mailboxRef:    MailboxRef;
   timeout:       LONGINT;
   pmsamCID:      CreationID;
   smca:          SMCA;
   END;

MSAMParam = RECORD
   CASE INTEGER OF
      1: (header: MailParamBlockHeader);
      2: (pmsamGetMSAMRecord: PMSAMGetMSAMRecordPB);
      3: (pmsamOpenQueues: PMSAMOpenQueuesPB);
      4: (pmsamSetStatus: PMSAMSetStatusPB);
      5: (pmsamLogError: PMSAMLogErrorPB);
      6: (smsamSetup: SMSAMSetupPB);
      7: (smsamStartup: SMSAMStartupPB);
      8: (smsamShutdown: SMSAMShutdownPB);
      9: (msamEnumerate: MSAMEnumeratePB);
      10: (msamDelete: MSAMDeletePB);
      11: (msamOpen: MSAMOpenPB);
      12: (msamOpenNested: MSAMOpenNestedPB);
      13: (msamClose: MSAMClosePB);
      14: (msamGetMsgHeader: MSAMGetMsgHeaderPB);
      15: (msamGetAttributes: MSAMGetAttributesPB);
      16: (msamGetRecipients: MSAMGetRecipientsPB);
      17: (msamGetContent: MSAMGetContentPB);
      18: (msamGetEnclosure: MSAMGetEnclosurePB);
      19: (msamEnumerateBlocks: MSAMEnumerateBlocksPB);
      20: (msamGetBlock: MSAMGetBlockPB);
      21: (msamMarkRecipients: MSAMMarkRecipientsPB);
      22: (msamnMarkRecipients: MSAMnMarkRecipientsPB);
      23: (msamCreate: MSAMCreatePB);
      24: (msamBeginNested: MSAMBeginNestedPB);
      25: (msamEndNested: MSAMEndNestedPB);
      26: (msamSubmit: MSAMSubmitPB);
      27: (msamPutMsgHeader: MSAMPutMsgHeaderPB);
      28: (msamPutAttribute: MSAMPutAttributePB);
```

```
    29: (msamPutRecipient: MSAMPutRecipientPB);
    30: (msamPutContent: MSAMPutContentPB);
    31: (msamPutEnclosure: MSAMPutEnclosurePB);
    32: (msamPutBlock: MSAMPutBlockPB);
    33: (msamCreateReport: MSAMCreateReportPB);
    34: (msamPutRecipientReport: MSAMPutRecipientReportPB);
    35: (pmsamCreateMsgSummary: PMSAMCreateMsgSummaryPB);
    36: (pmsamPutMsgSummary: PMSAMPutMsgSummaryPB);
    37: (pmsamGetMsgSummary: PMSAMGetMsgSummaryPB);
    38: (wakeupPMSAM: MailWakeupPMSAMPB);
    39: (createMailSlot: MailCreateMailSlotPB);
    40: (modifyMailSlot: MailModifyMailSlotPB);
  END;
```

MSAM Functions

Initializing an MSAM

```
FUNCTION PMSAMGetMSAMRecord(VAR paramBlock: MSAMParam): OSErr;

FUNCTION PMSAMOpenQueues(VAR paramBlock: MSAMParam): OSErr;

FUNCTION SMSAMSetup(VAR paramBlock: MSAMParam): OSErr;

FUNCTION SMSAMStartup(VAR paramBlock: MSAMParam): OSErr;
```

Enumerating Messages in a Queue

```
FUNCTION MSAMEnumerate(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN): OSErr;
```

Opening an Outgoing Message

```
FUNCTION MSAMOpen(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN): OSErr;
```

Reading Header Information

```
FUNCTION MSAMGetAttributes(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN):
                              OSErr;

FUNCTION MSAMGetRecipients(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN):
                              OSErr;

FUNCTION MSAMGetMsgHeader(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN):
                              OSErr;
```

Reading a Message

```
FUNCTION MSAMGetContent(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN):
                            OSErr;

FUNCTION MSAMGetEnclosure(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN):
                            OSErr;

FUNCTION MSAMEnumerateBlocks(VAR paramBlock: MSAMParam; asyncFlag:
                            BOOLEAN): OSErr;

FUNCTION MSAMGetBlock(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN): OSErr;

FUNCTION MSAMOpenNested(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN):
                            OSErr;
```

Marking a Recipient

```
FUNCTION MSAMnMarkRecipients(VAR paramBlock: MSAMParam; asyncFlag:
                            BOOLEAN): OSErr;

FUNCTION MSAMMarkRecipients(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN):
                            OSErr;
```

Closing a Message

```
FUNCTION MSAMClose(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN): OSErr;
```

Creating, Reading, and Writing Message Summaries

```
FUNCTION PMSAMCreateMsgSummary(VAR paramBlock: MSAMParam; asyncFlag:
                            BOOLEAN): OSErr;

FUNCTION PMSAMGetMsgSummary(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN):
                            OSErr;

FUNCTION PMSAMPutMsgSummary(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN):
                            OSErr;
```

Creating a Message

```
FUNCTION MSAMCreate(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN): OSErr;
```

Writing Header Information

```
FUNCTION MSAMPutAttribute(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN):
                            OSErr;

FUNCTION MSAMPutRecipient(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN):
                            OSErr;

FUNCTION MSAMPutMsgHeader(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN):
                            OSErr;
```

Writing a Message

```
FUNCTION MSAMPutContent(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN):
                                 OSErr;

FUNCTION MSAMPutEnclosure(VAR paramBlock: MSAMParam): OSErr;

FUNCTION MSAMPutBlock(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN): OSErr;

FUNCTION MSAMBeginNested(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN):
                                 OSErr;

FUNCTION MSAMEndNested(VAR paramBlock: MSAMParam): OSErr;
```

Submitting a Message

```
FUNCTION MSAMSubmit(VAR paramBlock: MSAMParam): OSErr;
```

Deleting a Message

```
FUNCTION MSAMDelete(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN): OSErr;
```

Generating Log Entries and Reports

```
FUNCTION PMSAMLogError(VAR paramBlock: MSAMParam): OSErr;

FUNCTION MSAMCreateReport(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN):
                                 OSErr;

FUNCTION MSAMPutRecipientReport(VAR paramBlock: MSAMParam; asyncFlag:
                                 BOOLEAN): OSErr;
```

Shutting Down a Server MSAM

```
FUNCTION SMSAMShutdown(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN): OSErr;
```

Setting Message Status

```
FUNCTION PMSAMSetStatus(VAR paramBlock: MSAMParam; asyncFlag: BOOLEAN):
                                 OSErr;
```

Personal MSAM AOCE Template Functions

```
FUNCTION MailCreateMailSlot(VAR paramBlock: MSAMParam): OSErr;

FUNCTION MailModifyMailSlot(VAR paramBlock: MSAMParam): OSErr;

FUNCTION MailWakeupPMSAM(VAR paramBlock: MSAMParam): OSErr;
```

Application-Defined Routine

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

Assembly-Language Summary

Trap Macros

Trap Macros Requiring Routine Selectors

`_oceTBDispatch`

Selector	Routine
$0500	PMSAMOpenQueues
$0501	SMSAMStartup
$0502	SMSAMShutdown
$0503	MSAMEnumerate
$0504	MSAMDelete
$0505	MSAMMarkRecipients
$0506	PMSAMGetMSAMRecord
$0507	MailWakeupPMSAM
$0508	MSAMOpen
$0509	MSAMOpenNested
$050A	MSAMClose
$050B	MSAMGetAttributes
$050C	MSAMGetRecipients
$050D	MSAMGetContent
$050E	MSAMGetEnclosure
$050F	MSAMEnumerateBlocks
$0510	MSAMGetBlock
$0511	MSAMGetMsgHeader
$0512	MSAMnMarkRecipients
$0514	MSAMCreate
$0515	MSAMBeginNested
$0516	MSAMEndNested
$0517	MSAMSubmit
$0518	MSAMPutAttribute
$0519	MSAMPutRecipient
$051A	MSAMPutContent
$051B	MSAMPutEnclosure
$051C	MSAMPutBlock
$051D	MSAMPutMsgHeader
$051F	MSAMCreateReport

Selector	Routine
$0520	MSAMPutRecipientReport
$0521	PMSAMLogError
$0522	PMSAMCreateMsgSummary
$0523	SMSAMSetup
$0525	PMSAMPutMsgSummary
$0526	PMSAMGetMsgSummary
$0527	PMSAMSetStatus
$052B	MailCreateMailSlot
$052C	MailModifyMailSlot

Result Codes

noErr	0	No error
corErr	–3	PowerShare mail server not running
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
kOCEToolboxNotOpen	–1500	Collaboration toolbox is shutting down
kOCEInvalidRef	–1502	Invalid message reference number
kOCEBufferTooSmall	–1503	Buffer is too small
kOCEVersionErr	–1504	Wrong version of nested message
kOCEInternalErr	–1506	Serious internal error
kOCEAlreadyExists	–1510	Duplicate recipient type
kIPMMsgTypeReserved	–1511	Message creator and/or type specified not allowed
kOCEInvalidRecipient	–1514	Bad recipient
kOCERefIsClosing	–1516	IPM Manager is shutting down the personal MSAM, or server MSAM's mail server is shutting down
kOCEWriteAccessDenied	–1541	Identity lacks write access privileges
kOCETargetDirectoryInaccessible	–1613	Target catalog is not currently available
kOCENoSuchDNode	–1615	Can't find specified dNode
kOCENoDupAllowed	–1641	Duplicate record name and type
kMailInvalidOrder	–15040	Content already closed
kMailInvalidSeqNum	–15041	Invalid message sequence number
kMailHdrAttrMissing	–15043	Required attribute not added to message
kMailBadEnclLengthErr	–15044	Invalid data length
kMailInvalidRequest	–15045	Reference number invalid with this request
kMailInvalidPostItVersion	–15046	Message summary is wrong version
kMailNotASlotInQ	–15047	Invalid value for a slot's incoming queue
kMailIgnoredErr	–15053	MSAM ignored high-level event
kMailLengthErr	–15054	Error occurred in sending the event
kMailTooManyErr	–15055	IPM Manager or MSAM too busy to handle event
kMailNoMSAMErr	–15056	No such MSAM

`kMailSlotSuspended`	–15058	**Slot is suspended**
`kMailMSAMSuspended`	–15059	**MSAM is suspended**
`kMailBadSlotInfo`	–15060	**Invalid slot information**
`kMailMalformedContent`	–15061	**Content data malformed**
`kMailNoSuchSlot`	–15062	**No such slot**
`kMailBadMSAM`	–15066	**MSAM unusable for unspecified reason**
`kMailBadState`	–15068	**Invalid status setting**
`kIPMInvalidMsgType`	–15091	**Only `kIPMOSFormatType` allowed when creating a letter**
`kIPMBlkNotFound`	–15107	**No such block**

Catalog Service Access Modules

Contents

This chapter describes how to write a **catalog service access module (CSAM),** a device driver that gives PowerTalk users access to external catalogs. Read this chapter if you want to integrate an external catalog into an AOCE system. You do not need to read this chapter if you simply want to use the Standard Catalog Package or the Catalog Manager to obtain catalog services.

To write a CSAM, you must already be familiar with the Catalog Manager application program interface (API). It is essential that you read the chapters "AOCE Utilities" and "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* before reading this chapter. This chapter assumes that you understand the Catalog Manager's functions and data types.

Because a CSAM is implemented as a Macintosh device driver, you also need to be familiar with the Device Manager. For information about the Device Manager and writing a device driver, see *Inside Macintosh: Devices.*

To allow the user to add and remove your CSAM and its catalogs from an AOCE system, you need to provide an AOCE setup template. The chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces* describes how to write an AOCE template. The chapter "Service Access Module Setup" in this book describes the setup template specifically, including how the setup template adds and removes a CSAM and its catalogs from the Setup catalog.

This chapter provides a brief introduction to CSAMs. Then it describes

n   the components of a CSAM

n   a CSAM's driver resource, including the Open and Close driver subroutines

n   a CSAM's catalog service and parse functions, which respond to requests from clients of the Catalog Manager

n   the method of indicating the features a catalog can support

n   the impact of various catalog features on the user's experience with a catalog

Introduction to Catalog Service Access Modules

The Catalog Manager provides a consistent interface for applications that use AOCE catalog services, regardless of whether the catalog is external to or part of AOCE software. Apple PowerShare catalogs and personal catalogs are part of AOCE software. Any other type of catalog is referred to as an **external catalog.** An external catalog is made available within an AOCE system by means of a CSAM, which supports the Catalog Manager API.

A CSAM provides these basic functions:

n   accepting Catalog Manager requests

n   translating the requests into a form that its external catalog understands

n   processing the requests, including activities such as obtaining information from the external catalog and adding information to the external catalog

n translating information for the Catalog Manager client into AOCE data formats such as records and attributes

n returning the information to the Catalog Manager

AOCE data formats are described in detail in the chapter "AOCE Utilities." The Catalog Manager API is described in the chapter "Catalog Manager." Both chapters are in *Inside Macintosh: AOCE Application Interfaces.*

A CSAM is not invoked directly by an application but indirectly through the Catalog Manager. The CSAM hides any underlying differences in how data is accessed and stored in its external catalog. For example, suppose an application wants to add a record to a catalog. The application calls the `DirAddRecord` function. If the target catalog is an external catalog, the Catalog Manager passes the request to the CSAM that supports that catalog. The CSAM then adds the record to its catalog and provides the creation ID of the new record. Thus, a Catalog Manager client can interact with all catalogs in the same way and can use standard AOCE data types to manipulate data. Figure 3-1 shows the relationship of an application, the Catalog Manager, a CSAM, and an external catalog. Although the figure shows a single external catalog, a CSAM can actually support any number of catalogs. The application and the Catalog Manager communicate through the Catalog Manager API. The Catalog Manager and the CSAM communicate through the CSAM's catalog service and parse functions, which are introduced in the next section.

Figure 3-1 Relationship of an application, the Catalog Manager, and a CSAM
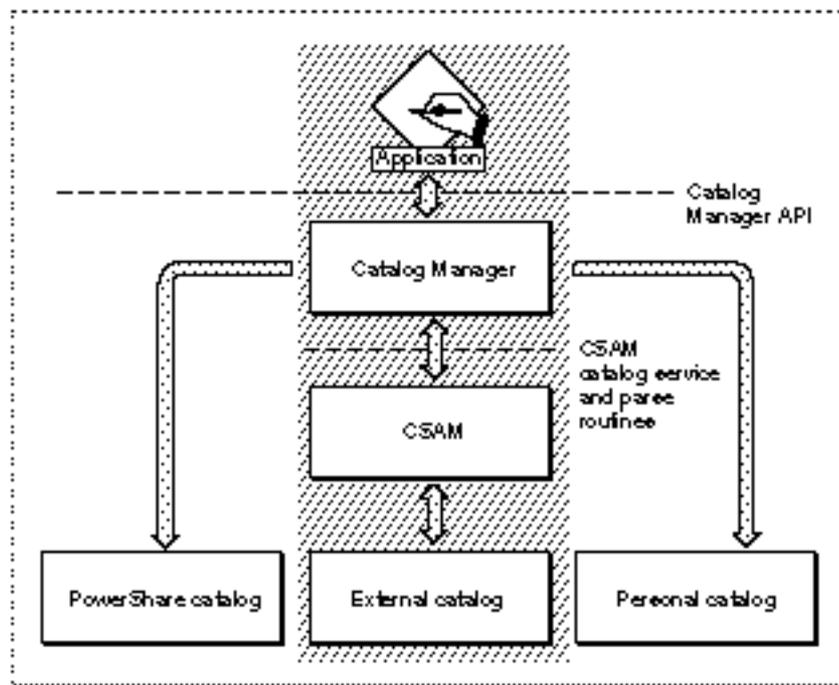
Every CSAM should support Catalog Manager requests to

n examine the contents of a dNode by real-time browsing, a search mechanism, or both

n enumerate the attribute types within a record

n look up attribute values

n detect changes within a dNode or a record

n get access controls for a dNode, record, or attribute type

A CSAM resides on a user's Macintosh computer and provides personal access to an external catalog. The catalog itself can exist anywhere—on the user's Macintosh, on a network server, or at a remote site accessed by a modem connection.

You can package a CSAM as a stand-alone driver file or as part of an AOCE messaging service access module. A **messaging service access module (MSAM)** translates and transfers messages between an AOCE messaging system and another messaging system. If you choose the stand-alone option, you provide a file of type `'dsam'` that contains the resources described in the section "Writing a Driver Resource for a CSAM" beginning on page 3-7. The file must also contain the resources that constitute your setup template. If you package your CSAM with a messaging service access module, include your CSAM, its setup template, and the MSAM in a file of type `'csam'` (for "combined SAM"). MSAMs are documented in the chapter "Messaging Service Access Modules" in this book. The setup template resources are described in the chapter "Service Access Module Setup" in this book.

Note
For historical reasons, the string `dsam` (or `DSAM`) rather than `csam` (or `CSAM`) is often part of a function name, field name, or data type name referring to a CSAM. u

Components of a CSAM

A CSAM consists of two main components: a driver resource that includes at least your driver's Open and Close subroutines, and the collection of functions that implement Catalog Manager functions. In addition, you must provide an AOCE setup template that allows the user to add, remove, and configure the CSAM and its catalogs. It can be helpful to think of the template as the third component of a CSAM product.

The setup template consists of a set of associated resources that reside in the resource fork of the CSAM file. A template code resource calls the Catalog Manager functions that add, remove, and configure the CSAM and its catalogs. The setup template is described in the chapter "Service Access Module Setup" in this book. Figure 3-2 shows the calling relationships between an application, a setup template, the Catalog Manager, the Device Manager, and a CSAM.

Figure 3-2   Calling relationships
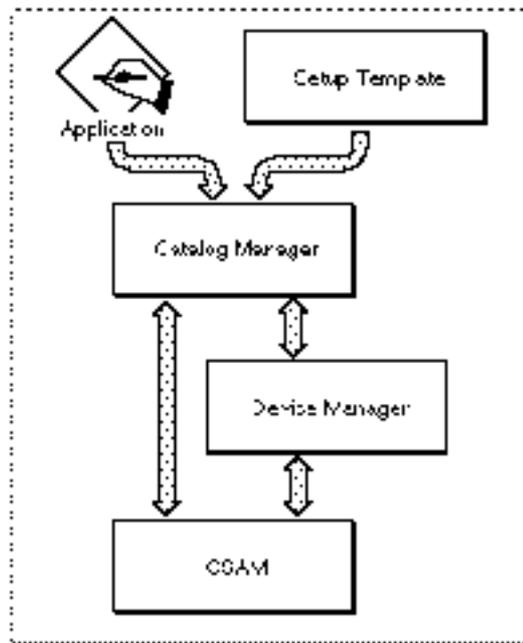
Requests for catalog services tend to be real-time in nature. Because Macintosh device drivers lend themselves to implementing real-time responses, you implement a CSAM as a Macintosh device driver.

A CSAM has two interfaces to Macintosh system software—one through the Device Manager and the other through the Catalog Manager. For the Device Manager interface, you must provide Open and Close driver subroutines. The Catalog Manager calls the Device Manager to open and close your driver. The Device Manager, in turn, calls your driver's Open and Close subroutines. You may provide the Prime, Status, and Control driver subroutines in accordance with the needs of your driver, but the Catalog Manager does not call these subroutines to communicate with your driver. The Open and Close driver subroutines are described in the section "Writing a Driver Resource for a CSAM" beginning on page 3-7. The Prime, Status, and Control driver subroutines are described in *Inside Macintosh: Devices.*

For the Catalog Manager interface, you provide a catalog service function and a parse function. When an application calls a Catalog Manager function, the Catalog Manager calls the CSAM's catalog service or parse function and passes it the application's catalog service request. A **catalog service function** accepts requests for catalog services from the Catalog Manager and calls CSAM-defined routines to implement those services. A **parse function** accepts requests to parse data about the CSAM's catalogs and their contents and calls CSAM-defined routines to implement those services.

Figure 3-3 illustrates who calls your driver subroutines and your catalog service and parse functions.

Figure 3-3   Who calls the CSAM driver subroutines and the catalog service and
parse functions
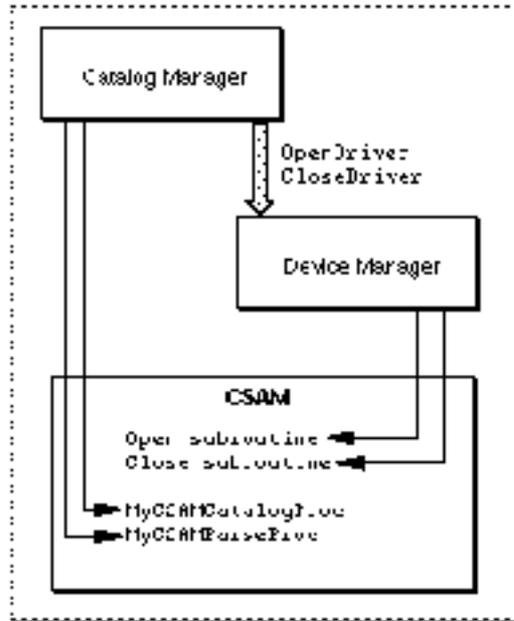
The sections that follow describe the CSAM's driver resource and the CSAM catalog service and parse functions.

Writing a Driver Resource for a CSAM

This section provides information about the required resources that constitute your CSAM's device driver.

The driver resource that you must provide in your CSAM, like all resources, has a type, a resource ID, a resource name, and resource attributes. The resource type is `'DRVR'`. You may set your `'DRVR'` resource ID to any valid value. The Catalog Manager properly installs your driver. The `'DRVR'` resource name must be the same as the name of your driver. This point is illustrated later in this section.

For your driver to work properly with the Catalog Manager, you must configure your `'DRVR'` resource as follows:

n Set the `resSysHeap` resource attribute to guarantee that your driver is loaded into the system heap.

n Set the `resLocked` resource attribute so that your driver is always available and nonrelocatable in memory.

You may set other attributes needed for your CSAM. See *Inside Macintosh: Devices* for more detailed information about the `'DRVR'` resource. See the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox* for information on resource attributes.

A resource of type `'DRVR'` contains header information and the driver's subroutines. The header information specifies certain settings for the driver and the offsets of the Open, Close, Status, Prime, and Control subroutines. The book *Inside Macintosh: Devices* provides information on setting up the header information. The header is followed by the driver subroutines themselves. Listing 3-1 illustrates the header of a sample CSAM's driver resource in Rez format.

Listing 3-1 A sample CSAM's driver resource header

```
#define DriverID            0x0b // unused, placeholder value

resource 'DRVR' (DriverID, ".SampleCSAM", sysheap, locked)
{
   /* driver flags */
   needLock, dontNeedTime, dontNeedGoodbye, noStatusEnable,
   ctlEnable, noWriteEnable, noReadEnable,

   0,              /* driver delay in ticks */
   0,              /* desk accessory mask */
   0,              /* desk accessory menu */
   ".SampleCSAM", /* driver name */
   /* the driver code follows the header fields */
};
```

Your Open subroutine handles initialization functions. It must do the following:

n Allocate and initialize any memory required. You need to allocate memory now because you cannot do so when the Catalog Manager calls your catalog service or parse function with an asynchronous request. Your CSAM must allocate its memory in the system heap and store the handle to the memory in the `dCtlStorage` field of the device control entry (`DCtlEntry`) structure.

n Call the `DirInstantiateDSAM` function to provide the Catalog Manager with pointers to your catalog service and parse functions. You can also provide a pointer to your private data, which the Catalog Manager passes back to you when it calls your catalog service and parse functions.

n Do any other preparation required to make the CSAM ready to receive and process service requests.

Your Open subroutine is always called synchronously.

In your Close subroutine, you should release any memory that you allocated in your Open subroutine. The Close subroutine is always called synchronously.

Depending on the needs of your driver, your Status, Prime, and Control subroutines may perform some work or simply return if called.

Note

The Device Manager interface requires you to use some assembly language. You can write your driver subroutines in a high-level language if you provide a dispatching mechanism, written in assembly language, between the Device Manager and the subroutines. See *Inside Macintosh: Devices* for instructions on writing subroutines in a high-level language and for detailed descriptions of all of the driver subroutines. u

When writing a device driver, you ordinarily write software that installs the driver in the Device Manager's unit table and opens the driver. For a CSAM, you do not need to provide software to install and open your driver directly. Instead, an AOCE setup template that you provide calls the `DirAddDSAM` function. This causes the Catalog Manager to install and open your driver. (Setup templates are discussed in the chapter "Service Access Module Setup" in this book.)

In addition to the `'DRVR'` resource, you must also provide a resource of type `'STR '` containing a single string that is both the name of your driver and the name of your `'DRVR'` resource. This string resource must have the resource name `DashName`. If you use another name for the string resource, the Catalog Manager will not be able to install your driver. Listing 3-2 illustrates the string resource. The name contained in this string resource must be the same as the name of the `'DRVR'` resource.

Listing 3-2   A CSAM's driver name string resource

```
/* The Driver's name must be in the resource named DashName." */
 resource 'STR ' (128, "DashName", purgeable ) {
    ".SampleCSAM"
};
```

Listing 3-1, Listing 3-2, and Figure 3-4 illustrate the following example. A file named My CSAM File contains a CSAM. The filename can be any string and is editable by a user. The file contains a `'STR '` resource named `DashName` that contains the string `.SampleCSAM`. The file also contains a `'DRVR'` resource whose resource name is `.SampleCSAM`. The driver itself is also named `.SampleCSAM`. The content of the string resource, the name of the `'DRVR'` resource, and the name of the driver are all the same.

Note that a driver name should always start with a period, followed by printable uppercase or lowercase characters, not to exceed a total of 31 characters.

Figure 3-4   Relationship of `'DRVR'` and `'STR '` resources
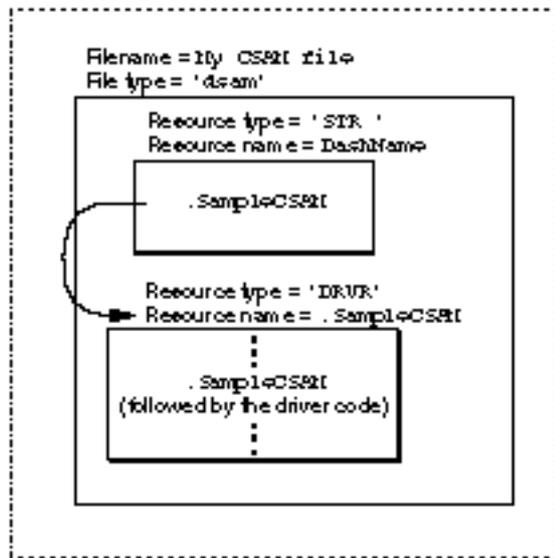
Responding to the Catalog Manager

When an application makes a request for catalog services and specifies an external catalog for which your CSAM is responsible, the Catalog Manager calls your CSAM's catalog service or parse function. The catalog service and parse functions are essentially dispatching functions that receive all Catalog Manager requests. They in turn call other functions that you provide to service the request.

A CSAM does not need to support every function in the Catalog Manager API. The Catalog Manager itself handles calls to the `DirGetDirectoryInfo`, `DirGetExtendedDirectoriesInfo`, `DirEnumerateDirectoriesGet`, and `DirEnumerateDirectoriesParse` functions and, therefore, does not pass these requests to a CSAM. Other Catalog Manager functions that are not passed to a CSAM include

n `DirAddADAPDirectory`

n `DirNetSearchADAPDirectoriesGet`

n `DirNetSearchADAPDirectoriesParse`

n `DirFindADAPDirectoryByNetSearch`

n `DirCreatePersonalDirectory`

n `DirOpenPersonalDirectory`

n `DirClosePersonalDirectory`

n `DirMakePersonalDirectoryRLI`

n `DirGetOCESetupRefNum`

You must provide a dispatch function. You can provide both a catalog service function and a parse function for this purpose. However, because Catalog Manager request codes for catalog service and parse requests do not overlap, you can process all Catalog Manager requests through a single dispatch function. To do this, specify the same address for your catalog service function and your parse function when you call the `DirInstantiateDSAM` function.

The Catalog Service Function

The Catalog Manager calls your catalog service function when an application calls a Catalog Manager function (other than one of the parse functions) and specifies a catalog that you support. Your catalog service function must determine the type of request that the application is making and then service that request.

The catalog service function has the following declaration:

```
pascal OSErr MyDSAMDirProc (Ptr dsamData,
                            DirParamBlockPtr paramBlock,
                            Boolean async);
```

The `dsamData` parameter contains the private value that you provided to the `DirInstantiateDSAM` function in the `dsamData` field of that function's parameter block. You define this value for your own use. Typically, it is a pointer to your private data area. The `paramBlock` parameter contains a pointer to the `DirParamBlock` parameter block that the application provided to the Catalog Manager when the application made the service request. The `async` parameter is a Boolean value that specifies if the request must be processed synchronously or asynchronously. If this parameter is `true`, you must process the request asynchronously; otherwise, you process the request synchronously.

You determine the type of request by examining the `reqCode` field of the `DirParamBlock` parameter block. Requests for catalog services map one-to-one to functions in the Catalog Manager API. The method by which you service the request (that is, implement the Catalog Manager function) is up to you. See the section "Data Types and Constants" beginning on page 3-42 for a complete list of request codes for Catalog Manager requests. See the function descriptions in the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* for information on the type of service each function performs, the behavior of the function, and the information it returns.

When an application calls a Catalog Manager function synchronously, the Catalog Manager passes the request to your CSAM within the calling application's context. Therefore, the CSAM can allocate, move, or purge memory and can call any function. The CSAM must process a synchronous request immediately. (See *Inside Macintosh: Processes* for a discussion of application context.)

When an application calls a Catalog Manager function asynchronously, the Catalog Manager passes the request to your CSAM at interrupt time. You cannot allocate, move, or purge memory at interrupt time, nor can you call a function that allocates, moves, or purges memory. If you can service the asynchronous request immediately—that is, if you

can service the request without performing tasks that are likely to consume a relatively large amount of time, such as an I/O operation—do so. Otherwise, your catalog service function should place the request in a private queue that it maintains and return control to the Catalog Manager with a result code of `noErr`. The Catalog Manager will already have set the `ioResult` field of the `DirParamBlock` parameter block to 1 before passing the asynchronous request to your catalog service function. As your function receives time to execute, service the request.

The CSAM can defer processing an asynchronous request until it is convenient to complete the request. It can install a VBL task, a Time Manager task, a Deferred Task Manager task, or a Notification Manager task to ensure that it receives system time at some point in the future. See *Inside Macintosh: Processes* for more information on these topics.

Note

When you have insufficient memory to service an asynchronous request, you should return an error. However, before returning, you can attempt to acquire additional memory for future requests. Set the `dNeedTime` flag in the `dCtlflags` field in your driver's `DCtlEntry` structure. Later, after a process calls the `SystemTask` or `WaitNextEvent` function, the Device Manager calls your Control subroutine with the `accRun` control code. At this time, you can safely allocate memory.

Do not queue an asynchronous request for which you have insufficient memory in the hope that you can acquire the memory later and successfully complete the request. This may result in a system freeze condition. u

Your catalog service function returns both a function result and a value in the `ioResult` field of the `DirParamBlock` parameter block to indicate the outcome of its handling of the request. For each type of service request (function) that you process, you should return only those result codes that are defined by the Catalog Manager for the function. The description of each Catalog Manager function provides the result codes that a given function can return.

If your function was called synchronously, set the `ioResult` field and return the appropriate function result code when you finish servicing the request.

If your function was called asynchronously, do the following when you finish servicing the request: Set the `ioResult` field to the appropriate result code. If the application provided a completion routine (the value of the `ioCompletion` field of the `DirParamBlock` parameter block is not `nil`), restore the application's A5 register by setting register A5 to the value of the `saveA5` field of the `DirParamBlock` parameter block and call the application's completion routine; otherwise, return. When the completion routine returns control to your catalog service function, you may service another pending request or return.

Listing 3-3 is an example of a simple catalog service function, the `DoMyDSAMDirProc` function, that determines the type of request and then calls another function to service the request. `DoMyDSAMDirProc` passes the called function a pointer to the CSAM's global data area, `myGlobalInfoPtr`. This is the value the CSAM originally gave to the

Catalog Manager when it called the `DirInstantiateDSAM` function. The Catalog Manager passes the value back to the catalog service function to use in servicing the request. In this example, the functions that service a particular catalog service request, such as the `DoProcessDirGetDNodeMetaInfoReqt` function, set the `ioResult` field of the `DirParamBlock` parameter block. Before returning, the `DoMyDSAMDirProc` function calls the `DoProcessCallCompletion` function, which calls the completion routine if the calling application specified one. See Listing 3-6 on page 3-28 for an example of calling an application's completion routine.

Listing 3-3   A catalog service function

```
pascal OSErr DoMyDSAMDirProc(
    register Ptr              myGlobalInfoPtr,
    register DirParamBlockPtr myParamBlock,
    Boolean                   async)
{
    switch (myParamBlock->header.reqCode) {   /* determine type of request */
    case kDirGetDirectoryIcon:
        DoProcessDirGetDirIconRequest(myGlobalInfoPtr, myParamBlock, async);
        break;
    case kDirGetDNodeMetaInfo:
        DoProcessDirGetDNodeMetaInfoReqt(myGlobalInfoPtr, myParamBlock, async);
        break;
    case kDirGetRecordMetaInfo:
        DoProcessDirGetRecrdMetaInfoReqt(myGlobalInfoPtr, myParamBlock, async);
        break;
    /* process other catalog service requests */
    }
    return (DoProcessCallCompletion(myParamBlock->header.ioResult, async));
}
```

The Catalog Manager defers calling your catalog service function until a time, sometimes called *deferred-task time*, when your function will work properly if the Macintosh is using virtual memory. See *Inside Macintosh: Memory* for information about memory management issues, including virtual memory.

The Parse Function

The Catalog Manager calls your parse function each time an application makes a parse request for a catalog that you support. A parse request corresponds to one of the Catalog Manager's parse functions, such as `DirLookupParse`, `DirEnumerateParse`, and so forth. Your parse function must determine the type of parse request that the application is making and then service that request.

The parse function has the following declaration:

```
pascal OSErr MyDSAMDirParseProc (Ptr dsamData,
                                 DirParamBlockPtr paramBlock,
                                 Boolean async);
```

The information in the section "The Catalog Service Function" beginning on page 3-11 also applies to the parse function. That information is not repeated here.

When you service a Catalog Manager parse request, you return information to the application by two methods. The first method, common to all Catalog Manager requests, consists of storing information in the appropriate fields of the `DirParamBlock` parameter block. The second, unique to parse requests, consists of passing data in predefined units to an application's callback routine.

It might be helpful to review here how Catalog Manager parse functions work. Each Catalog Manager parse function is paired with an associated get function. The `DirEnumerateDirectoriesGet/DirEnumerateDirectoriesParse` and `DirLookupGet/DirLookupParse` functions are examples of the get/parse function pairs in the Catalog Manager API. An application calls a Catalog Manager get function to obtain information about catalogs, records, attribute types, and so forth. If the target catalog is a catalog that you support, the Catalog Manager calls your CSAM's catalog service function to service the request. You place the requested data into a buffer provided by the application. You can use any format you wish for the data in this buffer; the data is therefore unreadable by the application. To retrieve the data from the buffer in a format that it understands, the application calls the corresponding Catalog Manager parse function, providing a pointer to a callback routine. The Catalog Manager, in turn, calls your CSAM's parse function. Your parse function passes data to the application by repeatedly calling the application's callback routine, each time passing it a defined chunk of data. The chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* provides descriptions of the application callback routines associated with different Catalog Manager parse functions and the type of data you need to return with each.

Note

Not all Catalog Manager get/parse function pairs work in exactly the same way. For example, most support starting or continuing an enumeration from a specified starting point, but some do not. Be sure to read the Catalog Manager function descriptions carefully to make sure your CSAM properly implements the Catalog Manager functions. u

You determine which Catalog Manager function the application has called by examining the `reqCode` field of the `DirParamBlock` parameter block. Then you process the request, just as you would when servicing a catalog service request. In addition, you call the application's callback routine as part of processing every parse request. You must set the A5 register to the value of the `saveA5` field of the `DirParamBlock` parameter block before calling the callback routine. You typically restore your own A5 register when you regain control.

Listing 3-4 illustrates how you call an application's callback routine. The
DoEnumerateParse function is called by another CSAM function in the course
of servicing the parse request that results from an application calling the
DirEnumerateParse function. The DoEnumerateParse function gets a pointer to
the DirEnumerateParse function's parameter block and a pointer to the buffer
the CSAM previously filled in response to the DirEnumerateGet function. The
application's callback routine expects to get a DirEnumSpec structure that provides
information about one record, alias, pseudonym, or child dNode in a given dNode.
Inside its main processing loop, the DoEnumerateParse function performs the
following tasks:

n It initializes the dataLength fields inside the DirEnumSpec structure to the
  maximum size RString that the CSAM supports.

n It calls its DoFillEnumSpec function to extract data about one record, alias,
  pseudonym, or child dNode from the buffer the CSAM previously filled and
  stores the data in a DirEnumSpec structure. If this function does not return the
  noErr result code, DoEnumerateParse exits the loop immediately, knowing it
  has extracted all the data from the buffer or it has encountered an error.

n It sets register A5 to the application's register A5 so the callback routine can access the
  application's global variables and saves its own register A5 value.

n It calls the application's callback routine, passing it the value of the clientData field
  from the DirEnumerateParse parameter block and the enumeration specification
  just constructed.

n It restores register A5 to its own register A5 value.

The DoEnumerateParse function continues to execute the loop until it runs out of data
to parse or encounters an error, or until the application's callback routine returns true.

Listing 3-4   Calling an application's callback routine

```
OSErr DoEnumerateParse (DirParamBlockPtr myParamBlock, Ptr buffer)
{
    DirEnumSpec      enumSpec;
    RString64        name, type;
    long             oldA5, saveSeq;
    Boolean          done = false;
    OSErr            myErr = 0;

    enumSpec.u.recordIdentifier.recordName = (RString*)&name;
    enumSpec.u.recordIdentifier.recordType = (RString*)&type;
    enumSpec.indexRatio = 0;

    while(!done) {
        name.dataLength = kRString64Size;
        type.dataLength = kRString64Size;
```

```
    /* extract data from the buffer and fill enumSpec appropriately */
    myErr = DoFillEnumSpec(buffer, &enumSpec);
    if (myErr != noErr) /* if no more data in the buffer, exit the loop */
        break;

    /* save my A5 register and call application's callback routine */
    oldA5 = SetA5(myParamBlock->enumerateParsePB.saveA5);
    done = (*myParamBlock->enumerateParsePB.eachEnumSpec)
                (myParamBlock->enumerateParsePB.clientData, &enumSpec);

    /* restore my A5 register */
    (void) SetA5(oldA5);
    }
return myErr;
}
```

To avoid problems when virtual memory is in use, you must call an application's callback routine at deferred-task time. See the chapters "Virtual Memory Manager" in *Inside Macintosh: Memory* and "Deferred Task Manager" in *Inside Macintosh: Processes* for more information on the handling of virtual memory and deferred tasks.

Determining the Version of the Catalog Manager

To determine the version of the Catalog Manager that is available, call the `Gestalt` function with the selector `gestaltOCEToolboxVersion`. The function returns the version number of the Collaboration toolbox in the low-order word of the `response` parameter. For example, a value of 0x0101 indicates version 1.0.1. If the Collaboration toolbox is not present and available, the `Gestalt` function returns 0 for the version number. You can use the constant `gestaltOCETB` for AOCE Collaboration toolbox version 1.0.

Indicating the Features You Support

A catalog may not support all of the features of the Catalog Manager API. Therefore, you must identify to the Catalog Manager the features supported by each catalog to which your CSAM provides access. The Catalog Manager API defines the data type `DirGestalt` that consists of bits that specify the features supported by a given catalog.

This section defines those bits, sometimes referred to as *feature flags* or *capability flags*. The support or lack thereof for certain features affects the human interface of some components of PowerTalk. The impact of various feature settings on the human interface is discussed in "Human Interface Considerations" beginning on page 3-22.

The features represented by the bits can be grouped into six general categories (the corresponding bits are listed for each category):

n **supplying identifying information**
  n `kSupportsDNodeNumberBit`
  n `kSupportsRecordCreationIDBit`
  n `kSupportsAttributeCreationIDBit`
  n `kSupportsPartialPathNamesBit`

n **pattern-matching for record names in an enumeration**
  n `kSupportsMatchAllBit`
  n `kSupportsBeginsWithBit`
  n `kSupportsExactMatchBit`
  n `kSupportsEndsWithBit`
  n `kSupportsContainsBit`

n **ordering the results of an enumeration**
  n `kSupportsOrderedEnumerationBit`
  n `kCanSupportNameOrderBit`
  n `kCanSupportTypeOrderBit`
  n `kSupportSortBackwardsBit`
  n `kSupportIndexRatioBit`

n **enumerating from a specified starting point**
  n `kSupportsEnumerationContinueBit`
  n `kSupportsLookupContinueBit`
  n `kSupportsEnumerateAttributeTypeContinueBit`
  n `kSupportsEnumeratePseudonymContinueBit`

n **other capabilities**
  n `kSupportsFindRecordBit`
  n `kSupportsAliasesBit`
  n `kSupportsPseudonymsBit`

n **reserved features**
  n `kSupportsAuthenticationBit`
  n `kSupportsProxiesBit`

The bits in a variable of type `DirGestalt` are defined as follows:

```
enum {
   kSupportsDNodeNumberBit               = 0,
   kSupportsRecordCreationIDBit          = 1,
   kSupportsAttributeCreationIDBit       = 2,
   kSupportsMatchAllBit                  = 3,
   kSupportsBeginsWithBit                = 4,
   kSupportsExactMatchBit                = 5,
```

```
    kSupportsEndsWithBit                          = 6,
    kSupportsContainsBit                          = 7,
    kSupportsOrderedEnumerationBit                = 8,
    kCanSupportNameOrderBit                       = 9,
    kCanSupportTypeOrderBit                       = 10,
    kSupportSortBackwardsBit                      = 11,
    kSupportIndexRatioBit                         = 12,
    kSupportsEnumerationContinueBit               = 13,
    kSupportsLookupContinueBit                    = 14,
    kSupportsEnumerateAttributeTypeContinueBit    = 15,
    kSupportsEnumeratePseudonymContinueBit        = 16,
    kSupportsAliasesBit                           = 17,
    kSupportsPseudonymsBit                        = 18,
    kSupportsPartialPathNamesBit                  = 19,
    kSupportsAuthenticationBit                    = 20,
    kSupportsProxiesBit                           = 21,
    kSupportsFindRecordBit                        = 22
};
```

Bit descriptions

`kSupportsDNodeNumberBit`

Set this bit if the catalog can identify a dNode by a dNode number. All catalogs must be able to identify a dNode by its pathname.

`kSupportsRecordCreationIDBit`

Set this bit if a catalog can identify a record by a record creation ID. If a catalog cannot identify a record by a record creation ID, you must set any record creation IDs that you return to 0. All catalogs must support identification of records by record name and record type. If a catalog does not additionally support record creation IDs, the record name and record type must be unique for each record. Note that to assure the proper behavior of aliases, a record creation ID must persist through system shutdown and startup.

`kSupportsAttributeCreationIDBit`

Set this bit if a catalog can identify an attribute value by specifying its attribute creation ID and attribute type. All catalogs must be able to identify an attribute value by specifying the attribute value and attribute type.

`kSupportsMatchAllBit`

Set this bit if the catalog supports browsing of record names and record types; that is, when an application calls the `DirEnumerateGet` or `DirFindRecordGet` function, the catalog can service a request to return information about all the records in a dNode or catalog.

kSupportsBeginsWithBit

Set this bit if the catalog supports a search for record names and record types beginning with a certain string; that is, when an application calls the `DirEnumerateGet` or `DirFindRecordGet` function, the catalog can service a request to provide information about all records whose record name or record type begins with the string provided by the application.

kSupportsExactMatchBit

Set this bit if the catalog supports a search for a record based on an exact match with the record name or record type; that is, when an application calls the `DirEnumerateGet` or `DirFindRecordGet` function, the catalog can service a request to provide information about the record whose record name or record type is provided by the application.

kSupportsEndsWithBit

Set this bit if the catalog supports a search for record names and record types ending with a certain string; that is, when an application calls the `DirEnumerateGet` or `DirFindRecordGet` function, the catalog can service a request to provide information about all records whose record name or record type ends with the string provided by the application.

kSupportsContainsBit

Set this bit if the catalog supports a search for record names and record types that contain a certain string; that is, when an application calls the `DirEnumerateGet` or `DirFindRecordGet` function, the catalog can service a request to provide information about all records whose record name or record type contains the string provided by the application.

kSupportsOrderedEnumerationBit

Set this bit if the catalog provides requested information in some sorted order when an application calls the `DirEnumerateGet` function. The catalog may provide the information in an unspecified sorted order. If it returns the information sorted by name or by type, set one or both of the two following bits.

kCanSupportNameOrderBit

Set this bit if the catalog supports the sorting by name option in the `DirEnumerateGet` function. If you set this bit, you must also set the `kSupportsOrderedEnumerationBit` bit.

kCanSupportTypeOrderBit

Set this bit if the catalog supports the sorting by type option in the `DirEnumerateGet` function. If you set this bit, you must also set the `kSupportsOrderedEnumerationBit` bit.

kSupportSortBackwardsBit

Set this bit if the catalog supports the backward sort direction option in the `DirEnumerateGet` function; that is, the catalog can provide entries preceding a certain point and sort those entries in reverse order.

kSupportIndexRatioBit

Set this bit if the catalog supports the index ratio feature in the
DirEnumerateGet function; that is, the catalog can provide the
approximate position of a record among all records in a dNode
as a percentile.

kSupportsEnumerationContinueBit

Set this bit if the catalog supports the continue feature in the
DirEnumerateGet function.

kSupportsLookupContinueBit

Set this bit if the catalog supports the continue feature in the
DirLookupGet function.

kSupportsEnumerateAttributeTypeContinueBit

Set this bit if the catalog supports the continue feature in the
DirEnumerateAttributeTypesGet function.

kSupportsEnumeratePseudonymContinueBit

Set this bit if the catalog supports the continue feature in the
DirEnumeratePseudonymGet function.

kSupportsAliasesBit

Set this bit if the catalog supports adding an alias with
the DirAddAlias function, deleting an alias with the
DirDeleteRecord function, and enumerating aliases with
the DirEnumerateGet function.

kSupportsPseudonymsBit

Set this bit if the catalog supports the DirAddPseudonym,
DirDeletePseudonym, and DirEnumeratePseudonymGet
functions, and if it supports enumerating pseudonyms with
the DirEnumerateGet function.

kSupportsPartialPathNamesBit

Set this bit if a catalog can specify a catalog node by using the
dNode number of an intermediate dNode and a partial pathname
starting from the intermediate dNode to the target dNode.

kSupportsAuthenticationBit

Reserved. Do not set this bit.

kSupportsProxiesBit

Reserved. Do not set this bit.

kSupportsFindRecordBit

Set this bit if the catalog supports the DirFindRecordGet and
DirFindRecordParse functions, that is, it can provide informa-
tion about records in the entire catalog, rather than in a given
dNode. The DirFindRecordGet function requests information
about records in an entire catalog; the DirEnumerateGet function
requests information about records in a particular dNode.

These bits are also described from the application's perspective in the chapter "Catalog
Manager" in *Inside Macintosh: AOCE Application Interfaces*.

You can use the following mask values to set the bits in a variable of type `DirGestalt`.

```
enum {
    kSupportsDNodeNumberMask        = 1L<<kSupportsDNodeNumberBit,
    kSupportsRecordCreationIDMask   = 1L<<kSupportsRecordCreationIDBit,
    kSupportsAttributeCreationIDMask = 1L<<kSupportsAttributeCreationIDBit,
    kSupportsMatchAllMask           = 1L<<kSupportsMatchAllBit,
    kSupportsBeginsWithMask         = 1L<<kSupportsBeginsWithBit,
    kSupportsExactMatchMask         = 1L<<kSupportsExactMatchBit,
    kSupportsEndsWithMask           = 1L<<kSupportsEndsWithBit,
    kSupportsContainsMask           = 1L<<kSupportsContainsBit,
    kSupportsOrderedEnumerationMask = 1L<<kSupportsOrderedEnumerationBit,
    kCanSupportNameOrderMask        = 1L<<kCanSupportNameOrderBit,
    kCanSupportTypeOrderMask        = 1L<<kCanSupportTypeOrderBit,
    kSupportSortBackwardsMask       = 1L<<kSupportSortBackwardsBit,
    kSupportIndexRatioMask          = 1L<<kSupportIndexRatioBit,
    kSupportsEnumerationContinueMask = 1L<<kSupportsEnumerationContinueBit,
    kSupportsLookupContinueMask     = 1L<<kSupportsLookupContinueBit,
    kSupportsEnumerateAttributeTypeContinueMask =
                        1L<<kSupportsEnumerateAttributeTypeContinueBit,
    kSupportsEnumeratePseudonymContinueMask =
                        1L<<kSupportsEnumeratePseudonymContinueBit,
    kSupportsAliasesMask            = 1L<<kSupportsAliasesBit,
    kSupportsPseudonymsMask         = 1L<<kSupportsPseudonymsBit,
    kSupportsPartialPathNamesMask   = 1L<<kSupportsPartialPathNamesBit,
    kSupportsAuthenticationMask     = 1L<<kSupportsAuthenticationBit,
    kSupportsProxiesMask            = 1L<<kSupportsProxiesBit,
    kSupportsFindRecordMask         = 1L<<kSupportsFindRecordBit
};
```

You can define the features that a catalog supports by adding the values of the appropriate masks and storing the resulting value in the CSAM file, where it is available to both your CSAM and your setup template. Listing 3-5 provides an example of specifying the features that a given catalog supports.

Listing 3-5   Setting the feature flags for a catalog

```
#define kPDirFeatures(           \
        kSupportsRecordCreationIDMask\
        + kSupportsAttributeCreationIDMask \
        + kSupportsMatchAllMask \
        + kSupportsBeginsWithMask\
        + kSupportsExactMatchMask \
        + kSupportsOrderedEnumerationMask \
```

```
+ kCanSupportNameOrderMask \
+ kSupportSortBackwardsMask \
+ kSupportsEnumerationContinueMask \
+ kSupportsLookupContinueMask \
)
```

Once you define the features for a given catalog, your setup template passes that information to the Catalog Manager when it calls the `DirAddDSAMDirectory` function to add that catalog to the Setup catalog. The Catalog Manager, in turn, provides the feature flags for a given catalog to an application when the application calls the `DirGetDirectoryInfo` function for a given catalog.

Human Interface Considerations

Although a CSAM itself has no human interface, the features that its catalogs can support affect the human interface provided for those catalogs by certain components of PowerTalk system software. The following components of PowerTalk system software make information in a catalog available to the user:

- the Catalogs Extension (CE)
- the Catalog-Browsing panel in the mailer
- the Find panel in the mailer
- the Find in Catalog command in the Apple menu

The mailer is described in the chapter "Standard Mail Package" in *Inside Macintosh: AOCE Application Interfaces*. For a description of how these elements appear to the user, see the book *PowerTalk User's Guide*.

You need to understand how the settings of certain feature flags affect the user's ability to make use of the information in a catalog using the PowerTalk human interface components. This section notes the capabilities a catalog must support to provide a particular service to the user through the PowerTalk components and the implications of not supporting those capabilities. Here are some service guidelines:

- For catalogs that may contain multiple records with the same name and type, support record creation IDs.
- For catalogs that may contain more than one attribute value of a given attribute type, support attribute creation IDs.
- For a browsable catalog, support "match all" and "exact match" capabilities.
- For proper searching of a catalog, support the "exact match" and "begins with" capabilities and either the "match all" or "find record" capability.
- For efficient handling of large catalogs, support "ordered enumeration," "sort backward," and "enumeration continue" capabilities.
- For best scrolling with large catalogs, support index ratios.
- For efficient attribute lookups, support the "lookup continue" capability.

This information is based on release 1 of the PowerTalk components and is subject to change in future releases.

Supporting Records Having the Same Name and Type

If a catalog allows multiple records to have the same name and type, then it must support record creation IDs. Allowing more than one record with the same name and type without support for record creation IDs creates problems with the CE's user interface. For instance, if a user opens such a catalog, the CE displays the records having the same name and type. If the user then opens one of the records, it is indeterminate which record's attributes are shown to the user. Likewise, if the user makes an alias to such a record, it is not guaranteed that the alias will resolve to the correct record.

Supporting Multiple Attribute Values of the Same Type

If a record in a catalog can contain more than one attribute value of a given attribute type, then you need to support attribute creation IDs for that catalog. The CE requires an attribute creation ID. In the absence of attribute creation IDs, the only way to distinguish among attribute values of the same type is by specifying the attribute value itself. Since attribute values may be as large as 64 KB, this is not efficient, and the attribute creation ID is required for performance reasons. For instance, imagine a record that contains many attributes whose type is Lyric and whose value is the lyric of a popular song. If a user wants to view all of the lyrics, you might run out of buffer space while responding to the `DirLookupGet` function. When the CE calls `DirLookupGet` again to continue the enumeration, it needs a practical way to indicate from which point to continue the enumeration.

If your catalog is unable to support a genuine attribute creation ID that permanently and uniquely identifies an attribute value, then it must support for each attribute value a unique identifier that persists from the time the CSAM is opened at system startup until system shutdown. This unique identifier is called a **pseudo-persistent attribute creation ID.** The pseudo-persistent attribute creation ID for a given attribute value is not, by definition, consistent between one session and the next.

Because the CE requires an attribute creation ID when a catalog may contain more than one attribute value of a given attribute type, you must set the `kSupportsAttributeCreationIDBit` bit, regardless of whether the type of attribute creation ID your catalog supports is genuine or pseudo-persistent.

It is desirable that you not reuse a value for a pseudo-persistent attribute creation ID once a session has ended. One way of achieving this is to generate values that incorporate a number derived from the date and time of the session with an incrementing number. This guarantees uniqueness both within and between sessions.

Note
If a catalog's records contain only one attribute value per attribute type, the CE does not require you to support attribute creation IDs. u

Supporting Browsing and Finding

If the user can view all of the records in a catalog through the CE or the Catalog-Browsing panel in the mailer, the catalog is browsable. If the user cannot view a catalog's contents, the catalog is nonbrowsable.

A catalog is browsable when both the `kSupportsMatchAllBit` and `kSupportsExactMatchBit` bits are set. A catalog with the "match all" capability supports user browsing by servicing requests to return information on all the records in a dNode or catalog. A browsable catalog must also support an "exact match" capability because, while browsing, a user may make an alias for any object. The "exact match" capability is needed to resolve an alias.

Finding or searching a catalog differs from browsing in that the user specifies, in whole or in part, a particular record name as the target of interest. The Find panel in the mailer and the Find in Catalog command in the Apple menu do not search a catalog unless the following bits are set:

n either the `kSupportsMatchAllBit` or the `kSupportsFindRecordBit` bit

n the `kSupportsExactMatchBit` bit

n the `kSupportsBeginsWithBit` bit

n the `kSupportsEnumerationContinue` bit

Supporting Large Catalogs

The CE and the Catalog-Browsing panel in the mailer attempt to achieve efficiencies in memory requirements and response time when dealing with large catalogs containing many records. This behavior is called **large-catalog mode.**

The CE and the Catalog-Browsing panel in the mailer can operate in large-catalog mode only if the catalog supports the following capabilities (the relevant bit that must be set is in parentheses):

n catalog can provide records in some sorted order
   (`kSupportsOrderedEnumerationBit`)

n catalog can provide, in reverse sorted order, the records preceding a specific point
   (`kSupportSortBackwardsBit`)

n catalog can continue an enumeration from a specified starting point
   (`kSupportsEnumerationContinueBit`)

If your CSAM provides access to a large catalog that does not provide records in some sorted and reverse sorted order and that cannot continue an enumeration from a specified starting point, you should make the catalog nonbrowsable. This avoids subjecting the user to heavy performance penalties and large memory requirements when working with that catalog. For example, when the CE is not operating in large-catalog mode, it attempts to enumerate all of the records in a given dNode of a catalog, bring the records into memory, and then sort them in the user's system script before displaying any records to the user. If the `DirEnumerateGet` function returns the `kOCEMoreData` result code, the CE calls the function again with a bigger buffer. It starts

the enumeration from the first record since the catalog does not support continuing the enumeration from the last record read. The CE continues to re-enumerate with a bigger buffer until the catalog dNode is completely enumerated or the Macintosh runs out of memory. It could take an unacceptable amount of memory and an unacceptably long time to open a catalog window for a large catalog that does not support large-catalog mode. (When the CE is operating in large-catalog mode, it enumerates either 60 records or three times the number of the records visible in the catalog window, whichever is greater.)

A user can still search for specific records in a large catalog that does not support large-catalog mode, although he or she is unable to view all of the records. The AppleLink address list is an example of a searchable but nonbrowsable catalog.

With large catalogs (those setting the `kSupportsOrderedEnumerationBit`, `kSupportSortBackwardsBit`, and `kSupportsEnumerationContinueBit` bits), the CE and the Catalog-Browsing panel use three different methods of managing scroll bars in a catalog window or panel:

n ratio-approximation

n letter-approximation

n three-position-thumb

The choice of method depends on the capabilities of the catalog being displayed and the script used in the catalog. If the catalog can provide the approximate position of a record within a catalog as a percentile value (an index ratio), it sets the `kSupportIndexRatioBit` bit. When this bit is set, the CE and the Catalog-Browsing panel always use the ratio-approximation method. The ratio-approximation method results in scroll bars that best indicate the true position of a record in a sorted catalog.

If a catalog cannot supply an index ratio, the scrolling method depends on whether the catalog can provide records sorted by record name (`kCanSupportNameOrderBit`) and whether the script used by the Macintosh system software matches the script used by the catalog.

If the catalog can return records in name order and the same script is used by both the catalog and the system software, the letter-approximation method is used. The letter-approximation method uses a table that maps each letter or range of letters in a given script to a number. After determining where the first visible record fits in the complete range of letters, the thumb is set accordingly.

If the scripts differ, the CE and the Catalog-Browsing panel have no idea where the record belongs within the range of letters in the catalog script. Therefore, they use the three-position-thumb method. They also use this method if a catalog cannot provide records sorted by record name. The three-position-thumb method is the least desirable method. It provides a scroll bar having only three positions–at the top of the scroll bar, at the bottom, and in the middle. These positions correspond to the first record in a catalog, the last record, and any other record. Thus, it gives no real information about the majority of records contained in a catalog. It is used as a last resort.

Table 3-1 summarizes the factors that determine the scrolling method.

Table 3-1   Determining the scrolling method for a catalog

Supports index ratio	Supports name order	Scripts	Scrolling method
Yes	Not applicable	Not applicable	Ratio approximation
No	Yes	Match	Letter-approximation
No	Yes	Do not match	Three-position-thumb
No	No	Not applicable	Three-position-thumb

Supporting Attribute Lookups

When the user is looking up attribute values through the CE, the efficiency of the operation depends a great deal on whether the catalog supports the continuation of the attribute lookup (indicated by the `kSupportsLookupContinueBit` bit). If a catalog does not support this feature and the `DirLookupGet` function returns the `kOCEMoreData` result code, the CE calls the function again with a bigger buffer instead of continuing the lookup from the last attribute. The CE continues to do this until all attribute values are completely enumerated or the Macintosh runs out of memory.

Providing Access Controls

You may want to provide access controls to safeguard the content of the catalogs that you support. If a catalog that you support already has its own system of controlling access, you can translate AOCE access controls into those of the external catalog, and vice versa. If a catalog has no access controls, you can implement them in your CSAM. You may provide access controls at the dNode, record, and attribute-type level to limit who may browse the contents of a dNode, record, or attribute type; who may modify the contents; and so forth. See the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* for a complete description of access controls.

To implement access controls, you must know who is making a particular service request. The `identity` field in the `DirParamBlock` parameter block indicates who is making the service request. It may contain the local identity, a specific identity, or 0.

The *local identity* is a reference value that identifies the principal user of the Macintosh computer on which your CSAM is installed. If your CSAM implements access controls, you should obtain the local identity by calling the `AuthGetLocalIdentity` function. When you receive requests for catalog service, compare the value in the `identity` field

in the `DirParamBlock` parameter block with the local identity. If the local identity is making the request, you can then determine if the access privileges of the local identity are sufficient to perform the requested operation.

If the `identity` field contains neither the local identity nor 0, it contains a specific identity. A *specific identity* is a reference value that identifies a user, other than the principal user, who has a PowerShare account. Your CSAM should take whatever action is appropriate, depending on how you choose to handle specific identities. One option, for example, is to treat a specific identity as a guest.

If the `identity` field contains 0, it indicates that a guest made the catalog service request. A guest is anyone other than the principal user and alternate users with PowerShare accounts. If the target catalog supports guest access, you can then determine if the access privileges for a guest are sufficient to perform the requested operation.

See the chapter "Authentication Manager" in *Inside Macintosh: AOCE Application Interfaces* for descriptions of the local identity, specific identities, and the `AuthGetLocalIdentity` function.

Handling Application Completion Routines

An application may provide a pointer to a completion routine when it makes an asynchronous Catalog Manager service request. The completion routine takes a single parameter—a pointer to the parameter block associated with the request.

Your CSAM must call the completion routine that an application provides. You need to

n   push the pointer to the parameter block onto the stack (in case the completion routine was written in C or Pascal)

n   store the pointer to the parameter block in register A0 (in case the completion routine was written in assembly language)

n   store the result code for the function you just serviced in register D0 (in case the completion routine was written in assembly language)

n   put the result code for the function in the `ioResult` field of the parameter block

After taking these steps, you set the A5 register to the value of the `saveA5` field of the `DirParamBlock` parameter block and call the completion routine.

You must call a completion routine at deferred-task time to avoid problems when virtual memory may be in use. See the chapters "Virtual Memory Manager" in *Inside Macintosh: Memory* and "Deferred Task Manager" in *Inside Macintosh: Processes* for more information on the handling of virtual memory and deferred tasks.

Listing 3-6 illustrates how you can call an application's completion routine.

Listing 3-6   Calling an application's completion routine

```
DirParamHeader record   0  ;  struct DirParamBlock {
qLink          ds.l     1  ;      Ptr       qLink;
reserved_H1    ds.l     1  ;      long      reserved_H1;
reserved_H2    ds.l     1  ;      long      reserved_H2;
ioCompletion   ds.l     1  ;      ProcPtr   ioCompletion;
ioResult       ds.w     1  ;      OSErr     ioResult;
saveA5         ds.l     1  ;      long      saveA5;
reqCode        ds.w     1  ;      short     reqCode;
               endr        ;  };

CallCompletion proc     export
               with     DirParamHeader
               move.l   4(sp),a0            ;A0 -> parameter block
               move.w   ioResult(a0),d0     ;D0 == ioResult
               move.l   ioCompletion(a0),d1 ;get application completion
               beq.s    @1                  ;exit if none
               move.l   a5,-(sp)            ;save my A5
               move.l   saveA5(a0),a5       ;restore application A5
               link     a6,#0               ;establish new stack frame
               move.l   a0,-(sp)            ;push param block on stack
               move.l   d1,a1               ;put completion routine in A1
               tst.w    d0                  ;set condition codes
               jsr      (a1)                ;call appl completion routine
               unlk     a6                  ;clean out the stack
               move.l   (sp)+,a5            ;restore my A5
@1             rts                          ;exit from CallCompletion
               endwith
               endp
               end
```

Catalog Service Access Module Reference

This section describes the Catalog Manager functions that a CSAM or its setup template calls and the functions that a CSAM provides. The structures and data types used by these functions are described in the chapters "AOCE Utilities" and "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces.* The Catalog Manager functions that your CSAM supports are described in the chapter "Catalog Manager."

CSAM Functions

This section describes the Catalog Manager functions that you use to initialize a CSAM and to add and remove a CSAM and the external catalogs that it supports.

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

To call a Catalog Manager function from assembly language, push the address of the `DirParamBlock` parameter block and the `async` flag onto the stack using the Pascal calling convention, and place the selector value for the `_oceTBDispatch` trap macro in register D0. Each function description includes the selector value for that function. The function returns its result code in the `ioResult` field of the parameter block. (The `DirParamBlock` parameter block is described in the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces*.)

A CSAM must support asynchronous requests. See the sections "The Catalog Service Function" on page 3-11 and "The Parse Function" on page 3-13 for information on how to support an asynchronous request.

Initializing a CSAM

A CSAM must call the `DirInstantiateDSAM` function before it can receive catalog service requests.

DirInstantiateDSAM

The `DirInstantiateDSAM` function provides the Catalog Manager with the addresses of a CSAM's catalog service and parse functions.

```
pascal OSErr DirInstantiateDSAM (DirParamBlockPtr paramBlock);
```

paramBlock **Pointer to a parameter block.**

Parameter block

ioResult	OSErr	Result code
dsamName	RStringPtr	CSAM name
dsamKind	OCEDirectoryKind	CSAM kind
dsamData	Ptr	CSAM private data
dsamDirProc	ProcPtr	CSAM's catalog service function
dsamDirParseProc	ProcPtr	CSAM's parse function
dsamAuthProc	ProcPtr	Reserved; set to nil

Field descriptions

`ioResult`	The result of the function.
`dsamName`	A pointer to the name of the CSAM. You define the name of your CSAM. Use the same name that your setup template provides to the `DirAddDSAM` function.
`dsamKind`	You define this field to identify your CSAM further. Typically, you provide the signature of your CSAM. Use the same value that your setup template provides to the `DirAddDSAM` function.
`dsamData`	A pointer to data that is private to the CSAM. You provide this pointer. The Catalog Manager passes this pointer to the CSAM's catalog service or parse function when an application calls a Catalog Manager function and specifies a catalog that you support.
`dsamDirProc`	A pointer to the CSAM's catalog service function. The Catalog Manager calls the CSAM's catalog service function to process all application requests for catalog services except parse requests. You must provide this value.
`dsamDirParseProc`	
	A pointer to the CSAM's parse function. The Catalog Manager calls the CSAM's parse function to process an application's parse request. You must provide this value. You can pass the same pointer as you provided in the `dsamDirProc` field if you process all Catalog Manager requests through a single function.
`dsamAuthProc`	Reserved. Set this field to `nil`.

DESCRIPTION

Your CSAM's Open subroutine must call the `DirInstantiateDSAM` function to provide the Catalog Manager with the addresses of the CSAM's catalog service and parse functions. Until you do this, no application can use the services of the CSAM. Note that the addresses (or entry points) can be identical if you simply dispatch the incoming requests to other functions within your CSAM.

The `DirInstantiateDSAM` function is the only function in the Catalog Manager API that is called exclusively by a CSAM.

If the values that you provide in the `dsamName` and `dsamKind` fields do not match those provided by your setup template to the `DirAddDSAM` function, then the `DirInstantiateDSAM` function returns the `kOCEDSAMInstallErr` result code. If this occurs, the Catalog Manager never sends the CSAM any requests.

SPECIAL CONSIDERATIONS

This function is always executed synchronously.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0127

RESULT CODES

noErr	0	No error
kOCELocalAuthenticationFail		
	−1561	User hasn't entered Key Chain password
kOCEDSAMInstallErr	−1628	Mismatch on CSAM name and kind
kOCEOCESetupRequired	−1633	Local identity is not set up
kOCEDSAMRecordNotFound	−1634	CSAM record not in Setup catalog

SEE ALSO

The `DirAddDSAM` function, described next, causes the Catalog Manager to install and open a CSAM.

A CSAM's catalog service and parse functions are described in the section "Application-Defined Functions" beginning on page 3-37.

Application signatures are described in the chapter "Finder Interface" in *Inside Macintosh: Macintosh Toolbox Essentials*.

Adding a CSAM and Its Catalogs

The Catalog Manager provides the `DirAddDSAM` and `DirAddDSAMDirectory` functions so that your setup template can add a CSAM and the catalogs it supports to a user's Setup catalog.

DirAddDSAM

The `DirAddDSAM` function opens a CSAM that you specify and adds a record representing the new CSAM to the Setup catalog.

```
pascal OSErr DirAddDSAM (DirParamBlockPtr paramBlock);
```

paramBlock Pointer to a parameter block.

Parameter block

ioResult	OSErr	Result code
dsamRecordCID	CreationID	Creation ID of CSAM record
dsamName	RStringPtr	CSAM name
dsamKind	OCEDirectoryKind	CSAM kind
fsSpec	FSSpecPtr	CSAM file specification

Field descriptions

ioResult   The result of the function.

dsamRecordCID   The creation ID of the record that the function adds to the Setup catalog. This record represents the CSAM. You pass the CSAM record's creation ID to the `DirAddDSAMDirectory` function when you want to add a catalog that the CSAM supports.

dsamName   A pointer to the name of the CSAM. You define the name of your
                            CSAM. Use the same name that your CSAM provides to the
                            DirInstantiateDSAM function.
dsamKind   You define this field to further identify your CSAM. Typically, you
                            provide the signature of your CSAM. Use the same value that your
                            CSAM provides to the DirInstantiateDSAM function.
fsSpec   A pointer to the file system specification structure that identifies the
                            file containing the CSAM.

DESCRIPTION

Your setup template calls the DirAddDSAM function to install a CSAM and make it
available to the user. You call this function before calling the DirAddDSAMDirectory
function.

The function installs the CSAM in the Device Manager's unit table and opens the
driver. The function creates a record for the CSAM. The CSAM record name is the string
that you provide in the dsamName field; its record type is aoce DSAM*xxxx*, where *xxxx*
is the value you provide in the dsamKind field. The function then adds the new
CSAM record to the Setup catalog and returns the record's creation ID.

The dsamName and dsamKind fields are provided to identify your CSAM. For example,
the name of an AppleLink CSAM might be AppleLink CSAM whereas its kind might be
ALNK. The combination of name and kind must be unique among CSAMs installed on
the computer.

If the CSAM is already installed, the function provides you with the creation ID of the
CSAM record and returns the kOCEDSAMRecordExists result code.

SPECIAL CONSIDERATIONS

If your CSAM is a component of a personal MSAM, your setup template calls the
DirAddDSAM function as part of the combined access module initialization procedure,
described in the chapter "Service Access Module Setup" in this book.

This function is always executed synchronously.

There is no registry to guarantee that your CSAM name and kind are unique. To ensure
uniqueness, set your CSAM name to your company name or product name and set your
CSAM kind to your CSAM's signature that is registered with Macintosh Developer
Technical Services.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$011D

RESULT CODES

noErr	**0**	No error
kOCEParamErr	**–50**	Invalid parameter
kOCELocalAuthenticationFail		
	–1561	User hasn't entered Key Chain password
kOCEDSAMInstallErr	**–1628**	CSAM could not be installed
kOCEDSAMRecordExists	**–1636**	CSAM record is already in Setup catalog

SEE ALSO

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

The `DirAddDSAMDirectory` function is described next.

To remove a CSAM that you added, use the `DirRemoveDSAM` function (page 3-35).

For more information about the Setup catalog and the CSAM record, see the chapter "Service Access Module Setup" in this book.

Application signatures are described in the chapter "Finder Interface" in *Inside Macintosh: Macintosh Toolbox Essentials.*

DirAddDSAMDirectory

The `DirAddDSAMDirectory` function adds a record for an external catalog to the Setup catalog.

```
pascal OSErr DirAddDSAMDirectory (DirParamBlockPtr paramBlock,
                                   Boolean async);
```

paramBlock   Pointer to a parameter block.

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

Parameter block

ioCompletion	ProcPtr	Your completion routine
ioResult	OSErr	Result code
clientData	long	You define this field
dsamRecordCID	CreationID	Creation ID of CSAM record
directoryName	DirectoryNamePtr	Name of the catalog
discriminator	DirDiscriminator	Discriminator value
features	DirGestalt	Feature flags
directoryRecordCID		
	CreationID	Creation ID of Catalog record

Field descriptions

ioCompletion	A pointer to a completion routine that you can provide. If you call this function asynchronously, it calls your completion routine when it completes execution. Set this field to nil if you don't provide a completion routine. The function ignores this field if you call it synchronously.
ioResult	The result of the function. When you execute the function asynchronously, the function sets this field to 1 as soon as the function has been queued for execution. When the function completes execution, it sets this field to the actual function result code.
clientData	Reserved for your use. If you call the DirAddDSAMDirectory function asynchronously, you can use this field to pass a private value to your completion routine.
dsamRecordCID	The creation ID of the record representing the CSAM associated with the catalog you want to add. You can obtain the CSAM's record creation ID from the DirAddDSAM function.
directoryName	A pointer to the name of the catalog that you want to add.
discriminator	A value that distinguishes between two or more catalogs with the same name. You define this value for the catalog you want to add.
features	The set of feature flags for the catalog you want to add. The flags are described in the section "Indicating the Features You Support" beginning on page 3-16.
directoryRecordCID	
	The creation ID of the record for the catalog that you want to add. You obtain the creation ID by using the CallBackDET macro to call the kDETcmdGetDSSpec callback routine. This provides you with the Catalog record's complete record ID, from which you can extract the creation ID.

DESCRIPTION

Your setup template calls the DirAddDSAMDirectory function to add to the Setup catalog a Catalog record for an external catalog that you specify. Once the function successfully completes execution, the external catalog is accessible to the user.

When you add a record for an external catalog, the catalog becomes visible to the DirEnumerateDirectoriesGet function. The catalog remains visible and available for use with other Catalog Manager functions until its Catalog record is explicitly removed from the Setup catalog by the DirRemoveDirectory function.

(AOCE software creates the Catalog record whose creation ID you provide in the directoryRecordCID field. It does this when the user adds a catalog to his or her available catalog services.)

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0133

RESULT CODES

noErr	**0**	No error
kOCEAlreadyExists	**−1510**	Catalog with same name and kind already exists
kOCELocalAuthenticationFail		
	−1561	User hasn't entered Key Chain password
kOCEDSAMInstallErr	**−1628**	CSAM doesn't exist
kOCEDSAMNotInstantiated	**−1635**	CSAM is not instantiated

SEE ALSO

The `DirRemoveDirectory` function is described on page 3-37.

The `DirEnumerateDirectoriesGet` function is described in the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces.*

The `DirAddDSAM` function is described on page 3-31.

For more information about the Setup catalog and the Catalog record, see the chapter "Service Access Module Setup" in this book.

Catalog feature flags are described in the section "Indicating the Features You Support" beginning on page 3-16.

The `CallBackDET` macro and the `kDETcmdGetDSSpec` callback routine are described in the chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces.*

Removing a CSAM and Its Catalogs

The Catalog Manager provides the `DirRemoveDSAM` and `DirRemoveDirectory` functions. Your template uses these functions to remove records for CSAMs and external catalogs from the Setup catalog.

DirRemoveDSAM

The `DirRemoveDSAM` function removes a record for a specific CSAM from the Setup catalog.

```
pascal OSErr DirRemoveDSAM (DirParamBlockPtr paramBlock);
```

`paramBlock`   Pointer to a parameter block.

Parameter block

ioResult	OSErr	**Result code**
dsamRecordCID	CreationID	**Creation ID of CSAM record**

Field descriptions

`ioResult`	The result of the function.
`dsamRecordCID`	The creation ID of the CSAM record in the Setup catalog for the CSAM that you want to remove. This creation ID is stored in the `kParentDSAMAttrTypeNum` attribute type in the template's record.

DESCRIPTION

Your setup template calls the `DirRemoveDSAM` function to remove a CSAM record from the Setup catalog. The function also closes the CSAM driver and removes from the Setup catalog all Catalog records for catalogs supported by the CSAM.

You can obtain the creation ID of the CSAM record by using the `CallBackDET` macro to call the `kDETcmdGetDSSpec` callback routine. Specify `kDETSelf` as the target to retrieve the `DSSpec` structure that identifies your template record. Then pass that `DSSpec` structure to the `DirLookupGet` function to read the `kParentDSAMAttrTypeNum` attribute type.

Once a CSAM's record is removed from the Setup catalog, the catalogs it serves are unavailable.

Ordinarily, you do not call this function. It is included to provide setup templates with flexibility in handling the CSAM record. For instance, if a user deletes all of the catalogs a CSAM supports, its setup template may remove the CSAM.

SPECIAL CONSIDERATIONS

This function is always executed synchronously.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0120

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEDSAMInstallErr`	–1628	CSAM doesn't exist

SEE ALSO

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

For more information about the Setup catalog, see the chapter "Service Access Module Setup" in this book.

You can add a CSAM to the Setup catalog by calling the `DirAddDSAM` function (page 3-31).

For information about the `CallBackDET` macro and the `kDETcmdGetDSSpec` callback routine, see the chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces*.

The `DirLookupGet` function is described in the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces*.

DirRemoveDirectory

The `DirRemoveDirectory` function removes from the Setup catalog a record that represents a catalog.

Because the function is not limited to removing external catalogs, it is described in the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces*.

Application-Defined Functions

You provide the catalog service and parse functions described in this section. You pass their addresses to the Catalog Manager when you call the `DirInstantiateDSAM` function. The Catalog Manager calls your functions when an application requests a service from an external catalog that you support. It is through these functions that you supply catalog services.

MyDSAMDirProc

The `MyDSAMDirProc` function accepts and processes Catalog Manager requests for catalog services. You must provide this function as part of your CSAM.

```
pascal OSErr MyDSAMDirProc (Ptr dsamData,
                            DirParamBlockPtr paramBlock,
                            Boolean async);
```

dsamData   A pointer to the CSAM's private data. This is the value that you previously passed to the `DirInstantiateDSAM` function in the `dsamData` field of its `DirParamBlock` parameter block.

paramBlock A pointer to the parameter block that the application passed to the Catalog Manager when the application called a Catalog Manager function.

async   A Boolean value that specifies if the request must be processed synchronously or asynchronously. If this field is set to `true`, you must process the request asynchronously.

DESCRIPTION

The Catalog Manager calls your catalog service function when an application requests a service, other than parse, from a catalog supported by your CSAM. You determine the type of request by examining the `reqCode` field in the `DirParamBlock` parameter block. Each possible value of the `reqCode` field corresponds to a Catalog Manager function. You then process the request and return the necessary information in the fields of the `paramBlock` parameter block.

RESULT CODES

Each type of service request that you may receive corresponds to a single Catalog Manager function. For each type of service request that you process, you should return only those result codes that are defined by the Catalog Manager for the corresponding function. See the description of each Catalog Manager function for the list of result codes you can return for that function.

SEE ALSO

The section "The Catalog Service Function" on page 3-11 provides general information on the actions that your `MyDSAMDirProc` function should take while servicing a request for catalog services. You decide how to implement a given Catalog Manager function for the catalog that you support.

The `DirInstantiateDSAM` function is described on page 3-29.

The chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* contains descriptions of each Catalog Manager function.

The request codes that may appear in the `reqCode` field in the `DirParamBlock` parameter block are listed in "Data Types and Constants" beginning on page 3-42.

MyDSAMDirParseProc

The `MyDSAMDirParseProc` function accepts and processes Catalog Manager parse requests. You may provide this function as part of your CSAM.

```
pascal OSErr MyDSAMDirParseProc (Ptr dsamData,
                                 DirParamBlockPtr paramBlock,
                                 Boolean async);
```

dsamData   A pointer to the CSAM's private data. This is the value that you previously passed to the `DirInstantiateDSAM` function in the `dsamData` field of its `DirParamBlock` parameter block.

paramBlock   A pointer to the parameter block that the application provided to the
              Catalog Manager when the application made a parse request.

async   A Boolean value that specifies if the request must be processed synchro-
              nously or asynchronously. If this field is set to `true`, you must process the
              request asynchronously.

DESCRIPTION

The Catalog Manager calls your parse function when an application makes a parse
request and specifies a catalog that your CSAM supports. You determine the specific
type of parse request by examining the `reqCode` field in the `DirParamBlock`
parameter block. Each possible value of the `reqCode` field corresponds to a Catalog
Manager function. You then process the request by returning the necessary information
in the fields of the parameter block and calling the application's callback routine.

SPECIAL CONSIDERATIONS

You can choose to dispatch all service requests through a single function. In that case,
you don't provide a separate and distinct parse function. Instead, you pass the same
address to the `DirInstantiateDSAM` function in both the `dsamDirProc` and
`dsamDirParseProc` fields.

RESULT CODES

Each type of parse request that you may receive corresponds to a single Catalog
Manager function. For each type of parse request that you process, you should return
only those result codes that are defined by the Catalog Manager for the corresponding
function. See the description of each Catalog Manager function for the list of result codes
you can return for that function.

SEE ALSO

The sections "The Catalog Service Function" on page 3-11 and "The Parse Function" on
page 3-13 provide general information on the actions that your `MyDSAMDirParseProc`
function should take while servicing a parse request. You decide how to implement a
given Catalog Manager parse function for the catalog that you support.

The chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* contains
descriptions of each Catalog Manager function.

The request codes that may appear in the `reqCode` field in the `DirParamBlock`
parameter block are listed in the section "Data Types and Constants" beginning on
page 3-42.

Resources

This section describes the 'DRVR' resource type that you provide in a CSAM.

The Driver Resource

The driver resource contains the executable code that implements support for the Catalog Manager API. Listing 3-7 shows the Rez definition of the 'DRVR' resource type.

Listing 3-7   'DRVR' resource definition

```
type 'DRVR' {
   boolean = 0;                             /* unused */
   boolean dontNeedLock,    needLock;       /* lock driver in memory */
   boolean dontNeedTime,    needTime;       /* for periodic action */
   boolean dontNeedGoodbye,needGoodbye;     /* call before heap reinit */
   boolean noStatusEnable, statusEnable;    /* responds to Status */
   boolean noCtlEnable,    ctlEnable;       /* responds to Control */
   boolean noWriteEnable,  writeEnable;     /* responds to Write */
   boolean noReadEnable,   readEnable;      /* responds to Read */
   byte = 0;                                /* unused */
   unsigned integer;                        /* driver delay (ticks) */
   integer;                                 /* DA event mask */
   integer;                                 /* driver menu ID */
   unsigned integer = 50;                   /* offset to DRVRRuntime Open */
   unsigned integer = 54;                   /* offset to DRVRRuntime Prime */
   unsigned integer = 58;                   /* offset to DRVRRuntime Control */
   unsigned integer = 62;                   /* offset to DRVRRuntime Status */
   unsigned integer = 66;                   /* offset to DRVRRuntime Close */
   pstring[31];                             /* driver name */
   hex string;                              /* driver code */
};
```

The driver resource contains the following fields:

n   An unused Boolean value.

n   A Boolean value that indicates if your driver should be locked in memory. You must set this to needLock for a CSAM.

n   A Boolean value that indicates if your driver should receive processor time periodically. Set this according to the needs of your CSAM.

n   A Boolean value that indicates if your driver should be notified before the application heap is reinitialized. Because your CSAM driver must reside in the system heap, this Boolean value is irrelevant.

n   A Boolean value that indicates if your driver responds to Status calls from the
    Device Manager.

n   A Boolean value that indicates if your driver responds to Control calls from the
    Device Manager.

n   A Boolean value that indicates if your driver responds to Write calls from the
    Device Manager.

n   A Boolean value that indicates if your driver responds to Read calls from the
    Device Manager.

n   An unused value.

n   A value that indicates the number of ticks between your periodic time intervals. If you
    have already specified the Boolean value `needTime`, set this according to the needs
    of your CSAM.

n   A value for desk accessories. It is irrelevant to a CSAM.

n   A value for desk accessories. It is irrelevant to a CSAM.

n   Five 4-byte values that specify the offsets to your Open, Prime, Control, Status, and
    Close driver subroutines, respectively.

n   The name of your CSAM driver. You can use uppercase and lowercase letters when
    naming your driver, but the first character must be a period.

n   The hexadecimal representation of your executable code. Your driver subroutines
    must be aligned on a word boundary.

Summary of Catalog Service Access Modules

C Summary

Data Types and Constants

```
enum {      /* feature flag bits */
   kSupportsDNodeNumberBit                  = 0,
   kSupportsRecordCreationIDBit             = 1,
   kSupportsAttributeCreationIDBit          = 2,
   kSupportsMatchAllBit                     = 3,
   kSupportsBeginsWithBit                   = 4,
   kSupportsExactMatchBit                   = 5,
   kSupportsEndsWithBit                     = 6,
   kSupportsContainsBit                     = 7,
   kSupportsOrderedEnumerationBit           = 8,
   kCanSupportNameOrderBit                  = 9,
   kCanSupportTypeOrderBit                  = 10,
   kSupportSortBackwardsBit                 = 11,
   kSupportIndexRatioBit                    = 12,
   kSupportsEnumerationContinueBit          = 13,
   kSupportsLookupContinueBit               = 14,
   kSupportsEnumerateAttributeTypeContinueBit= 15,
   kSupportsEnumeratePseudonymContinueBit   = 16,
   kSupportsAliasesBit                      = 17,
   kSupportsPseudonymsBit                   = 18,
   kSupportsPartialPathNamesBit             = 19,
   kSupportsAuthenticationBit               = 20,
   kSupportsProxiesBit                      = 21,
   kSupportsFindRecordBit                   = 22
};

enum {      /* feature flag masks */
   kSupportsDNodeNumberMask        = 1L<<kSupportsDNodeNumberBit,
   kSupportsRecordCreationIDMask   = 1L<<kSupportsRecordCreationIDBit,
   kSupportsAttributeCreationIDMask = 1L<<kSupportsAttributeCreationIDBit,
   kSupportsMatchAllMask           = 1L<<kSupportsMatchAllBit,
   kSupportsBeginsWithMask         = 1L<<kSupportsBeginsWithBit,
   kSupportsExactMatchMask         = 1L<<kSupportsExactMatchBit,
```

```
    kSupportsEndsWithMask           = 1L<<kSupportsEndsWithBit,
    kSupportsContainsMask           = 1L<<kSupportsContainsBit,
    kSupportsOrderedEnumerationMask = 1L<<kSupportsOrderedEnumerationBit,
    kCanSupportNameOrderMask        = 1L<<kCanSupportNameOrderBit,
    kCanSupportTypeOrderMask        = 1L<<kCanSupportTypeOrderBit,
    kSupportSortBackwardsMask       = 1L<<kSupportSortBackwardsBit,
    kSupportIndexRatioMask          = 1L<<kSupportIndexRatioBit,
    kSupportsEnumerationContinueMask = 1L<<kSupportsEnumerationContinueBit,
    kSupportsLookupContinueMask     = 1L<<kSupportsLookupContinueBit,
    kSupportsEnumerateAttributeTypeContinueMask =
                        1L<<kSupportsEnumerateAttributeTypeContinueBit,
    kSupportsEnumeratePseudonymContinueMask =
                        1L<<kSupportsEnumeratePseudonymContinueBit,
    kSupportsAliasesMask            = 1L<<kSupportsAliasesBit,
    kSupportsPseudonymsMask         = 1L<<kSupportsPseudonymsBit,
    kSupportsPartialPathNamesMask   = 1L<<kSupportsPartialPathNamesBit,
    kSupportsAuthenticationMask     = 1L<<kSupportsAuthenticationBit,
    kSupportsProxiesMask            = 1L<<kSupportsProxiesBit,
    kSupportsFindRecordMask         = 1L<<kSupportsFindRecordBit
};

/* request codes for Catalog Manager functions */
#define kDirEnumerateParse                  0x101
#define kDirLookupParse                     0x102
#define kDirEnumerateAttributeTypesParse    0x103
#define kDirEnumeratePseudonymParse         0x104
#define kDirNetSearchADAPDirectoriesParse   0x105
#define kDirEnumerateDirectoriesParse       0x106
#define kDirFindADAPDirectoryByNetSearch    0x107
#define kDirNetSearchADAPDirectoriesGet     0x108
#define kDirAddRecord                       0x109
#define kDirDeleteRecord                    0x10A
#define kDirAddAttributeValue               0x10B
#define kDirDeleteAttributeValue            0x10C
#define kDirChangeAttributeValue            0x10D
#define kDirVerifyAttributeValue            0x10E
#define kDirAddPseudonym                    0x10F
#define kDirDeletePseudonym                 0x110
#define kDirEnumerateGet                    0x111
#define kDirEnumerateAttributeTypesGet      0x112
#define kDirEnumeratePseudonymGet           0x113
#define kDirGetNameAndType                  0x114
#define kDirSetNameAndType                  0x115
#define kDirGetRecordMetaInfo               0x116
```

```
#define kDirLookupGet                       0x117
#define kDirGetDNodeMetaInfo                0x118
#define kDirGetDirectoryInfo                0x119
#define kDirEnumerateDirectoriesGet         0x11A
#define kDirAbort                           0x11B
#define kDirAddAlias                        0x11C
#define kDirAddDSAM                         0x11D
#define kDirOpenPersonalDirectory           0x11E
#define kDirCreatePersonalDirectory         0x11F
#define kDirRemoveDSAM                      0x120
#define kDirGetDirectoryIcon                0x121
#define kDirMapPathNameToDNodeNumber        0x122
#define kDirMapDNodeNumberToPathName        0x123
#define kDirGetLocalNetworkSpec             0x124
#define kDirGetDNodeInfo                    0x125
#define kDirFindValue                       0x126
#define kDirInstantiateDSAM                 0x127
#define kDirGetOCESetupRefNum               0x128
#define kDirGetDNodeAccessControlGet        0x12A
#define kDirGetRecordAccessControlGet       0x12C
#define kDirGetAttributeAccessControlGet    0x12E
#define kDirGetDNodeAccessControlParse      0x12F
#define kDirDeleteAttributeType             0x130
#define kDirClosePersonalDirectory          0x131
#define kDirMakePersonalDirectoryRLI        0x132
#define kDirAddDSAMDirectory                0x133
#define kDirGetRecordAccessControlParse     0x134
#define kDirRemoveDirectory                 0x135
#define kDirGetExtendedDirectoriesInfo      0x136
#define kDirAddADAPDirectory                0x137
#define kDirGetAttributeAccessControlParse  0x138
#define kDirFindRecordGet                   0x140
#define kDirFindRecordParse                 0x141

struct DirInstantiateDSAMPB {
   AuthDirParamHeader                   /* parameter block header */
   RStringPtr        dsamName;          /* CSAM name */
   OCEDirectoryKind  dsamKind;          /* CSAM kind */
   Ptr               dsamData;          /* CSAM private data */
   ProcPtr           dsamDirProc;       /* catalog service function */
   ProcPtr           dsamDirParseProc;  /* parse function */
   ProcPtr           dsamAuthProc;      /* reserved, set to nil */
};

typedef struct DirInstantiateDSAMPB DirInstantiateDSAMPB;
```

```
struct DirAddDSAMPB {
   AuthDirParamHeader                 /* parameter block header */
   CreationID        dsamRecordCID; /* CSAM record creation ID */
   RStringPtr        dsamName;       /* CSAM name */
   OCEDirectoryKind  dsamKind;       /* CSAM kind */
   FSSpecPtr         fsSpec;         /* CSAM's file specification */
};

typedef struct DirAddDSAMPB DirAddDSAMPB;

struct DirAddDSAMDirectoryPB {
   AuthDirParamHeader                 /* parameter block header */
   CreationID        dsamRecordCID; /* CSAM record creation ID */
   DirectoryNamePtr  directoryName; /* catalog name */
   DirDiscriminator  discriminator; /* catalog discriminator value */
   DirGestalt        features;       /* feature flags for the catalog */
   CreationID        directoryRecordCID;
                                     /* Catalog record creation ID */
};

typedef struct DirAddDSAMDirectoryPB DirAddDSAMDirectoryPB;

struct DirRemoveDSAMPB {
   AuthDirParamHeader                 /* parameter block header */
   CreationID  dsamRecordCID;     /* CSAM record creation ID */
};

typedef struct DirRemoveDSAMPB DirRemoveDSAMPB;
```

CSAM Functions

Initializing a CSAM

```
pascal OSErr DirInstantiateDSAM
                         (DirParamBlockPtr paramBlock);
```

Adding a CSAM and Its Catalogs

```
pascal OSErr DirAddDSAM       (DirParamBlockPtr paramBlock);
pascal OSErr DirAddDSAMDirectory
                         (DirParamBlockPtr paramBlock, Boolean async);
```

Removing a CSAM and Its Catalogs

```
pascal OSErr DirRemoveDSAM  (DirParamBlockPtr paramBlock);
```

Application-Defined Functions

```pascal
pascal OSErr MyDSAMDirProc    (Ptr dsamData, DirParamBlockPtr paramBlock,
                               Boolean async);
pascal OSErr MyDSAMDirParseProc
                              (Ptr dsamData, DirParamBlockPtr paramBlock,
                               Boolean async);
```

Pascal Summary

Data Types and Constants

```pascal
CONST
   { feature flag bits }
   kSupportsDNodeNumberBit                     = 0;
   kSupportsRecordCreationIDBit                = 1;
   kSupportsAttributeCreationIDBit             = 2;
   kSupportsMatchAllBit                        = 3;
   kSupportsBeginsWithBit                      = 4;
   kSupportsExactMatchBit                      = 5;
   kSupportsEndsWithBit                        = 6;
   kSupportsContainsBit                        = 7;
   kSupportsOrderedEnumerationBit              = 8;
   kCanSupportNameOrderBit                     = 9;
   kCanSupportTypeOrderBit                     = 10;
   kSupportSortBackwardsBit                    = 11;
   kSupportIndexRatioBit                       = 12;
   kSupportsEnumerationContinueBit             = 13;
   kSupportsLookupContinueBit                  = 14;
   kSupportsEnumerateAttributeTypeContinueBit  = 15;
   kSupportsEnumeratePseudonymContinueBit      = 16;
   kSupportsAliasesBit                         = 17;
   kSupportsPseudonymsBit                      = 18;
   kSupportsPartialPathNamesBit                = 19;
   kSupportsAuthenticationBit                  = 20;
   kSupportsProxiesBit                         = 21;
   kSupportsFindRecordBit                      = 22;

   { feature flag masks }
   kSupportsDNodeNumberMask                    = $00000001;
   kSupportsRecordCreationIDMask               = $00000002;
   kSupportsAttributeCreationIDMask            = $00000004;
```

```
kSupportsMatchAllMask                             = $00000008;
kSupportsBeginsWithMask                           = $00000010;
kSupportsExactMatchMask                           = $00000020;
kSupportsEndsWithMask                             = $00000040;
kSupportsContainsMask                             = $00000080;
kSupportsOrderedEnumerationMask                   = $00000100;
kCanSupportNameOrderMask                          = $00000200;
kCanSupportTypeOrderMask                          = $00000400;
kSupportSortBackwardsMask                         = $00000800;
kSupportIndexRatioMask                            = $00001000;
kSupportsEnumerationContinueMask                  = $00002000;
kSupportsLookupContinueMask                       = $00004000;
kSupportsEnumerateAttributeTypeContinueMask       = $00008000;
kSupportsEnumeratePseudonymContinueMask           = $00010000;
kSupportsAliasesMask                              = $00020000;
kSupportsPseudonymsMask                           = $00040000;
kSupportsPartialPathNamesMask                     = $00080000;
kSupportsAuthenticationMask                       = $00100000;
kSupportsProxiesMask                              = $00200000;
kSupportsFindRecordMask                           = $00400000;

{ request codes for Catalog Manager requests }
kDirEnumerateParse                                $101
kDirLookupParse                                   $102
kDirEnumerateAttributeTypesParse                  $103
kDirEnumeratePseudonymParse                       $104
kDirNetSearchADAPDirectoriesParse                 $105
kDirEnumerateDirectoriesParse                     $106
kDirFindADAPDirectoryByNetSearch                  $107
kDirNetSearchADAPDirectoriesGet                   $108
kDirAddRecord                                     $109
kDirDeleteRecord                                  $10A
kDirAddAttributeValue                             $10B
kDirDeleteAttributeValue                          $10C
kDirChangeAttributeValue                          $10D
kDirVerifyAttributeValue                          $10E
kDirAddPseudonym                                  $10F
kDirDeletePseudonym                               $110
kDirEnumerateGet                                  $111
kDirEnumerateAttributeTypesGet                    $112
kDirEnumeratePseudonymGet                         $113
kDirGetNameAndType                                $114
kDirSetNameAndType                                $115
kDirGetRecordMetaInfo                             $116
```

```
kDirLookupGet                           $117
kDirGetDNodeMetaInfo                    $118
kDirGetDirectoryInfo                    $119
kDirEnumerateDirectoriesGet             $11A
kDirAbort                               $11B
kDirAddAlias                            $11C
kDirAddDSAM                             $11D
kDirOpenPersonalDirectory               $11E
kDirCreatePersonalDirectory             $11F
kDirRemoveDSAM                          $120
kDirGetDirectoryIcon                    $121
kDirMapPathNameToDNodeNumber            $122
kDirMapDNodeNumberToPathName            $123
kDirGetLocalNetworkSpec                 $124
kDirGetDNodeInfo                        $125
kDirFindValue                           $126
kDirInstantiateDSAM                     $127
kDirGetOCESetupRefNum                   $128
kDirGetDNodeAccessControlGet            $12A
kDirGetRecordAccessControlGet           $12C
kDirGetAttributeAccessControlGet        $12E
kDirGetDNodeAccessControlParse          $12F
kDirDeleteAttributeType                 $130
kDirClosePersonalDirectory              $131
kDirMakePersonalDirectoryRLI            $132
kDirAddDSAMDirectory                    $133
kDirGetRecordAccessControlParse         $134
kDirRemoveDirectory                     $135
kDirGetExtendedDirectoriesInfo          $136
kDirAddADAPDirectory                    $137
kDirGetAttributeAccessControlParse      $138
kDirFindRecordGet                       $140
kDirFindRecordParse                     $141

DirInstantiateDSAMPB = RECORD
   qLink:         Ptr;              { reserved }
   reserved1:     LONGINT;          { reserved }
   reserved2:     LONGINT;          { reserved }
   ioCompletion:  ProcPtr;          { your completion routine }
   ioResult:      OSErr;            { result code }
   saveA5:        LONGINT;          { reserved }
   reqCode:       INTEGER;          { Catalog Manager function request code }
   reserved:      ARRAY[1..2] OF LONGINT;
                                    { reserved }
```

```
serverHint:      AddrBlock;       { PowerShare server's AppleTalk address }
dsRefNum:        INTEGER;         { personal catalog reference number }
callID:          LONGINT;         { reserved }
identity:        AuthIdentity;    { requestor's authentication identity }
gReserved1:      LONGINT;         { reserved }
gReserved2:      LONGINT;         { reserved }
gReserved3:      LONGINT;         { reserved }
clientData:      LONGINT;         { you define this field }
dsamName:        RStringPtr;      { CSAM name }
dsamKind:        OCEDirectoryKind;
                                  { CSAM kind }
dsamData:        Ptr;            { CSAM private data }
dsamDirProc:     ProcPtr;         { CSAM's catalog service routine }
dsamDirParseProc: ProcPtr;        { CSAM's parse routine }
dsamAuthProc:    ProcPtr;         { reserved }
END;

DirAddDSAMPB = RECORD
   qLink:           Ptr;             { reserved }
   reserved1:       LONGINT;         { reserved }
   reserved2:       LONGINT;         { reserved }
   ioCompletion:    ProcPtr;         { your completion routine }
   ioResult:        OSErr;           { result code }
   saveA5:          LONGINT;         { reserved }
   reqCode:         INTEGER;         { Catalog Manager function request code }
   reserved:        ARRAY[1..2] OF LONGINT;
                                     { reserved }
   serverHint:      AddrBlock;       { PowerShare server's AppleTalk address }
   dsRefNum:        INTEGER;         { personal catalog reference number }
   callID:          LONGINT;         { reserved }
   identity:        AuthIdentity;    { requestor's authentication identity }
   gReserved1:      LONGINT;         { reserved }
   gReserved2:      LONGINT;         { reserved }
   gReserved3:      LONGINT;         { reserved }
   clientData:      LONGINT;         { you define this field }
   dsamRecordCID:   CreationID;      { creation ID of CSAM record }
   dsamName:        RStringPtr;      { CSAM name }
   dsamKind:        OCEDirectoryKind;
                                     { CSAM kind }
   fsSpec:          FSSpecPtr;       { CSAM file specification }
   END;
```

```
DirAddDSAMDirectoryPB = RECORD
   qLink:          Ptr;            { reserved }
   reserved1:      LONGINT;        { reserved }
   reserved2:      LONGINT;        { reserved }
   ioCompletion:   ProcPtr;        { your completion routine }
   ioResult:       OSErr;          { result code }
   saveA5:         LONGINT;        { reserved }
   reqCode:        INTEGER;        { Catalog Manager function request code }
   reserved:       ARRAY[1..2] OF LONGINT;
                                   { reserved }
   serverHint:     AddrBlock;      { PowerShare server's AppleTalk address }
   dsRefNum:       INTEGER;        { personal catalog reference number }
   callID:         LONGINT;        { reserved }
   identity:       AuthIdentity;   { requestor's authentication identity }
   gReserved1:     LONGINT;        { reserved }
   gReserved2:     LONGINT;        { reserved }
   gReserved3:     LONGINT;        { reserved }
   clientData:     LONGINT;        { you define this field }
   dsamRecordCID:  CreationID;     { creation ID of CSAM record }
   directoryName:  DirectoryNamePtr;   { catalog name }
   discriminator:  DirDiscriminator;   { discriminator value }
   features:       DirGestalt;         { feature flags for the catalog }
   directoryRecordCID: CreationID;     { creation ID of catalog record }
   END;

DirRemoveDSAMPB = RECORD
   qLink:          Ptr;            { reserved }
   reserved1:      LONGINT;        { reserved }
   reserved2:      LONGINT;        { reserved }
   ioCompletion:   ProcPtr;        { your completion routine }
   ioResult:       OSErr;          { result code }
   saveA5:         LONGINT;        { reserved }
   reqCode:        INTEGER;        { Catalog Manager function request code }
   reserved:       ARRAY[1..2] OF LONGINT;
                                   { reserved }
   serverHint:     AddrBlock;      { PowerShare server's AppleTalk address }
   dsRefNum:       INTEGER;        { personal catalog reference number }
   callID:         LONGINT;        { reserved }
   identity:       AuthIdentity;   { requestor's authentication identity }
   gReserved1:     LONGINT;        { reserved }
   gReserved2:     LONGINT;        { reserved }
   gReserved3:     LONGINT;        { reserved }
   clientData:     LONGINT;        { you define this field }
   dsamRecordCID:  CreationID;     { creation ID of CSAM record }
   END;
```

CSAM Functions

Initializing a CSAM

```
FUNCTION DirInstantiateDSAM (paramBlock: DirParamBlockPtr): OSErr;
```

Adding a CSAM and Its Catalogs

```
FUNCTION DirAddDSAM          (paramBlock: DirParamBlockPtr): OSErr;
FUNCTION DirAddDSAMDirectory(paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
```

Removing a CSAM and Its Catalogs

```
FUNCTION DirRemoveDSAM       (paramBlock: DirParamBlockPtr): OSErr;
```

Application-Defined Functions

```
FUNCTION MyDSAMDirFunc       (dsamData: Ptr; paramBlock: DirParamBlockPtr;
                              async: BOOLEAN): OSErr;
FUNCTION MyDSAMDirParseFunc (dsamData: Ptr; paramBlock: DirParamBlockPtr;
                              async: BOOLEAN): OSErr;
```

Assembly-Language Summary

Trap Macros

Trap Macros Requiring Routine Selectors

```
_oceTBDispatch
```

Selector	Routine
0$127	DirInstantiateDSAM
0$11D	DirAddDSAM
0$133	DirAddDSAMDirectory
0$120	DirRemoveDSAM
0$135	DirRemoveDirectory

Result Codes

noErr	**0**	No error
kOCEParamErr	**−50**	Invalid parameter
kOCEAlreadyExists	**−1510**	Catalog with same name and kind already exists
kOCELocalAuthenticationFail		
	−1561	User hasn't entered Key Chain password
kOCEDSAMInstallErr	**−1628**	CSAM could not be installed or doesn't exist
kOCEOCESetupRequired	**−1633**	Local identity is not set up
kOCEDSAMRecordNotFound	**−1634**	CSAM record not in Setup catalog
kOCEDSAMNotInstantiated	**−1635**	CSAM is not instantiated
kOCEDSAMRecordExists	**−1636**	CSAM record is already in Setup catalog

Service Access Module Setup

Contents

This chapter describes how you install and configure an Apple Open Collaboration Environment (AOCE) catalog service access module (CSAM) and a personal messaging service access module (personal MSAM). It also describes how any messaging service access module, either personal or server type, obtains address information from a user.

You need to read this chapter if you are developing a catalog service access module or a personal messaging service access module to work with the PowerTalk software. For information on initializing a server MSAM, see the chapter "Messaging Service Access Modules" in this book and the *PowerShare System Manager's Guide*.

This chapter assumes you have already read one or both of the chapters that describe service access modules, "Catalog Service Access Modules" and "Messaging Service Access Modules," both in this book. To use this chapter, you must also know how to write an AOCE template. The chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces* describes how to write an AOCE template. In addition, you must have a general understanding of AOCE catalogs and the Catalog Manager application programming interface (API). See the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* for information about AOCE catalogs.

This chapter begins with a brief introduction to the setup process. It then describes the types of records in the PowerTalk Setup catalog and the setup and address AOCE templates. Then it explains how you can

- set up a combined CSAM/MSAM
- set up a CSAM
- set up an MSAM
- modify existing CSAM or MSAM configuration information
- add, modify, and remove address information

Introduction to SAM Setup

A user of PowerTalk software who wants to add a catalog or messaging service uses the PowerTalk Key Chain to add and configure the service. The PowerTalk Key Chain, in turn, uses special AOCE templates that you provide with your service access module to supply information pages that let the user enter and modify setup information.

Each CSAM or personal MSAM requires a setup template. A **setup template** is a set of AOCE templates that allow a user to install and configure a service access module. In addition, both personal and server MSAMs require an **address template** that allows a user to enter address information into a User record.

The templates provide the only human interface for CSAMs and personal MSAMs. Server MSAMs, being foreground applications, have their own user interface. However, the address template is the mechanism through which a server MSAM obtains the address information that it requires.

There are three types of CSAM and personal MSAM files:

n   CSAM only, or stand-alone CSAM (file type `'dsam'`)

n   MSAM only, or stand-alone MSAM (file type `'msam'`)

n   combined CSAM/MSAM (file type `'csam'`)

Note

The abbreviation "dsam"—found in the `'dsam'` file type and in function names and data structures in the AOCE interface files—stands for "directory service access module," the name used for catalog service access modules in early versions of the AOCE software. The `'csam'` file type is so named because it implements a "combined service access module." Therefore, a file of type `'dsam'` implements a CSAM, and a file of type `'csam'` implements both a CSAM and an MSAM. u

A CSAM-only file contains the CSAM driver and a setup template. It might also contain additional templates to allow items in its catalogs to be viewed in the Finder. An MSAM-only file contains a background application (the MSAM itself), a setup template, and an address template. A combined CSAM/MSAM file contains a CSAM driver, an MSAM background application, a setup template, and an address template (and possibly other templates to allow items in its catalogs to be viewed in the Finder).

A server MSAM application file (file type `'APPL'`) contains an address template among its resources.

The setup templates add information to the PowerTalk Setup catalog. The **PowerTalk Setup catalog** is a personal catalog named "PowerTalk Setup Preferences" and kept in the Preferences folder in the System Folder. The Setup catalog stores information about the catalog and messaging services that are available to the principal user of a given Macintosh computer.

The following sections describe how you use routines from the MSAM and Catalog Manager APIs and AOCE utility routines to install and configure a service access module. The reference section describes the records in the PowerTalk Setup catalog, the AOCE templates that create and modify those records, and the resources required for these templates.

About Personal MSAMs and Addresses

When a user sends an AppleMail letter, the AOCE software examines the recipients in the mailer and determines how to route the letter based on the catalog names in the recipient addresses. For example, if an address contains the name of a PowerShare catalog, the AOCE toolbox hands the letter to the PowerShare router. For this reason, every personal MSAM must have an associated catalog name, even if there is no CSAM associated with the MSAM. A CSAM provides the user with the ability to browse an external catalog. The catalog name associated with an MSAM, by contrast, is used by the AOCE software for routing the letter. It can be the name of an external catalog for which the user has a CSAM, but need not be.

Note

In the current version of the PowerTalk system software, each personal MSAM must be associated with a unique catalog name. In future versions of PowerTalk, two or more personal MSAMs may be able to use the same catalog name for routing purposes. In that case, the AOCE software would look at the extension type in the address as well as the catalog name. (Addresses and address extension types are described in the chapter "Messaging Service Access Modules" in this book.) u

Your address template must allow the user to enter addressing information and be capable of formulating an address in the format required by AOCE.

For incoming mail, the personal MSAM has no responsibility to forward mail to other recipients. However, to be most useful to the user, your personal MSAM should translate the addresses from its own external messaging system into addresses intelligible to AOCE, including the catalog name and the address extension information, and place them in the mailer. Then the user can use these addresses for replies to the letter and can drag them out of the mailer into a personal catalog or information card for future use.

Adding Catalog and Mail Services

This section describes what your setup template must do to add catalog and mail services to a user's AOCE system. It first discusses what your setup template must do when you provide a combined CSAM/MSAM. Then it explains what the setup template must do when you provide only a CSAM or an MSAM, but not both.

The process of adding a service requires actions from the user, the PowerTalk Key Chain, and your setup template. This section describes the user and Key Chain actions only to the extent that it clarifies the actions your setup template must take. If you want more detail on the user actions, see the *PowerTalk User's Guide*.

As you will see in the following sections, before you can add a service of any type (catalog, mail, or combined), you must have added the SAM itself. The procedure for adding a SAM is covered in the description of adding the relevant service.

Each section identifies the records and attributes in the PowerTalk Setup catalog that your setup template creates or manipulates in the course of adding a given service, and it identifies the functions you use to do that. Some of these functions are described in the other chapters in this book. Others are described in the chapters "AOCE Utilities," "AOCE Templates," "Catalog Manager," and "Authentication Manager" in *Inside Macintosh: AOCE Application Interfaces*.

AOCE record and attribute types that are discussed in this section are often identified by an index constant. See the chapter "AOCE Utilities" for an explanation of indexed record and attribute types.

Adding a Combined Service

This section tells you how to add a combined catalog and mail service. Read this section if you are providing both a CSAM and an MSAM. It explains what your setup template must do to allow a user to add and configure a combined mail and catalog service.

The setup template for a combined MSAM and CSAM includes a main aspect template for a Combined record. Your code resource for the Combined record must create a CSAM record and check whether an MSAM record already exists. If the MSAM record does not exist, the template's code resource must create that record as well.

Listing 4-1 illustrates a setup template for a combined catalog and mail service.

Listing 4-1   Combined catalog and mail service setup template

```
#define SystemSevenOrLater 1

#include "Types.r"
#include "OCETemplates.h"
#include "OCE.r"

#define kSurfWriterAspect          1000
#define kSurfWriterInfoPage        1250

#define kSignature                 'WAVE'
#define kServiceRecordType         kCombinedRecTypeBody "WAVE"
#define kAspectName                "SurfWriter_Service_Aspect"
#define kInfoPageName              "SurfWriter_Service_Info_Page"

#define kWindowWidth        259
#define kWindowHeight       200
#define kZeroRect           {0, 0, 0, 0}
#define kDoubleLineLeft     kDETSubpageIconLeft
#define kDoubleLineRight    kWindowWidth - kDETSubpageIconLeft
#define kDoubleLineTop      kTopBorder + kFieldHeight + 8
#define kDoubleLineBottom   kDoubleLineTop + 1
#define kDoubleLineRect     {kDoubleLineTop, kDoubleLineLeft,\
                            kDoubleLineBottom, kDoubleLineRight}

resource 'deta' (kSurfWriterAspect,purgeable)
{
   0, dropCheckConflicts, isMainAspect
};
```

```
resource 'rstr' (kSurfWriterAspect + kDETTemplateName, purgeable)
{
   kAspectName
};


resource 'rstr' (kSurfWriterAspect + kDETRecordType, purgeable)
{
   kServiceRecordType
};


resource 'rstr' (kSurfWriterAspect + kSAMAspectKind, purgeable)
{
   "Full SurfWriter Service"
};


resource 'rstr' (kSurfWriterAspect + kDETAspectKind, purgeable)
{
   "Combined SurfWriter Service"
};


resource 'rstr' (kSurfWriterAspect + kDETAspectName, purgeable) {
   "unconfigured SurfWriter Service"
};


resource 'rstr' (kSurfWriterAspect + kSAMAspectUserName, purgeable) {
   "SurfWriter User"
};


resource 'sami' (kSurfWriterAspect + kSAMAspectSlotCreationInfo, purgeable)
{
   2,                                     // max number of catalogs/slots
   kSignature,                            // catalog signature, MSAM type
   servesMSAM,                            // an MSAM template
   servesDSAM,                            // a CSAM template
   "SurfWriter Combined Service ",        // display when user clicks Add
   "untitled combined SurfWriter Service" // new record name
};
```

```
// Custom window

resource 'detw' (kSurfWriterAspect + kDETAspectInfoPageCustomWindow,
                 purgeable)
    {
    {-1, -1, kWindowHeight, kWindowWidth},
    includePopup
    };

// Include code resource

include "SurfWriterCode" 'detc'(0) as
     'detc'(kSurfWriterAspect+kDETAspectCode, purgeable);

// Include icons

include "SurfWriterIcons" 'ICN#'(0) as
                  'ICN#'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'icl4'(0) as
                  'icl4'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'icl8'(0) as
                  'icl8'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'ics#'(0) as
                  'ics#'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'ics4'(0) as
                  'ics4'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'ics8'(0) as
                  'ics8'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'SICN'(0) as
                  'SICN'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);

//-------------------------------------------------------------------------
// Info-page

resource 'deti' (kSurfWriterInfoPage, purgeable)
{
   kDefaultSortIndex,
   kZeroRect,
   noSelectFirstText,
   {
      kDETNoProperty, kDETNoProperty, kSurfWriterInfoPage;
   },
   {
```

```
    }
};

resource 'rstr' (kSurfWriterInfoPage + kDETTemplateName, purgeable) {
    kInfoPageName
};

resource 'rstr' (kSurfWriterInfoPage + kDETRecordType, purgeable) {
    kServiceRecordType
};

resource 'rstr' (kSurfWriterInfoPage + kDETInfoPageName, purgeable) {
    "SurfWriter Combined Service"
};

resource 'rstr' (kSurfWriterInfoPage + kDETInfoPageMainViewAspect,
                purgeable) {
    kAspectName
};

resource 'detv' (kSurfWriterInfoPage, "subpageview", purgeable)
    {
    {
    kDoubleLineRect, kDETNoFlags, kDETNoProperty,
    Box { kDETBoxIsGrayed };

    kDETSubpageIconRect, kDETNoFlags, kDETAspectMainBitmap,
    Bitmap { kDETLargeIcon };
    };
    };
```

During system initialization, the Key Chain loads all setup templates. As specified by the `'sami'` resource in Listing 4-1, the Key Chain offers the user the choice "SurfWriter Combined Service" when the user clicks the Add button. When the user selects this choice, the Key Chain adds a new record to the PowerTalk Setup catalog. As specified by Listing 4-1, this record's name is "untitled combined SurfWriter Service," and its type is "aoce CombinedWAVE". The Key Chain also writes four attributes into this record, as follows:

n An associated catalog attribute that points to the record that contains information about the catalog associated with this service. In the case of a combined service, this attribute points back to the Combined record itself.

n An associated mail service attribute that points to the record that contains information about the mail slot associated with this service. In the case of a combined service, this attribute points back to the Combined record itself.

n An attribute of type "aoce Unconfigured" (attribute type index
   `kUnconfiguredAttrTypeNum`) indicating that the service has not
   yet been set up.

n A version attribute that contains the version number of the Key Chain at the time it
   created the record.

The Key Chain adds a line to the Key Chain window representing the new record. The
line includes the key icon used by the Key Chain, the service name you specified in the
`kDETAspectName` resource ("unconfigured SurfWriter Service" in Listing 4-1), and
whatever default values your template provides for the Name and Kind fields in the
`kSAMAspectUserName` and `kSAMAspectKind` resources ("SurfWriter User" and "Full
SurfWriter Service" in Listing 4-1).

Your code resource should create no new records or attributes at this time. However,
when the user opens the Key Chain entry, the Key Chain opens your information page
and calls your code resource with the `kDETcmdInstanceInit` routine selector. Your
setup template must follow the steps in the following two sections in order to set up a
combined mail and catalog service.

Adding the Catalog Service

To add a catalog service to a combined service, you must write certain attributes and also
activate the catalog. The attributes in the Combined record are summarized in Table 4-6
on page 4-70. Use the Catalog Manager function `DirAddAttributeValue` to add
attributes to a record. The function is described in the chapter "Catalog Manager" in
Inside Macintosh: AOCE Application Interfaces; see the `DoAddAttribute` function on
page 4-18 in Listing 4-2 for an example of its use.

Note
If you are adding a catalog service only, you follow these same steps but
add and modify attributes in the Catalog record rather than in the
Combined record. See "Adding a Catalog Service Only" beginning on
page 4-28 for more information. u

1. Write the comment attribute. This is a string that you can use for any purpose.
   An application or template code resource can read this string by calling the
   `DirGetExtendedDirectoriesInfo` function; see the chapter "Catalog Manager"
   in *Inside Macintosh: AOCE Application Interfaces* for a description of this function.

2. Write the real name attribute, containing the external name of the catalog. This name
   is for your own use; it is not read by the AOCE software. An application or template
   code resource can read this name by calling the `DirGetExtendedDirectoriesInfo`
   function. Whereas the catalog name you provide to the `DirAddDSAMDirectory`
   function (see step 5) must be unique within the AOCE system, the "real" name need
   not be. For example, the user might have accounts on two different SurfWriter mail
   servers. Each would have to have a distinct name for display in the Key Chain, but
   the Combined records for both could contain the name "SurfWriter Mail" for the real
   name attribute.

3. If you wish, you can write a private data attribute. This attribute can contain binary data of any length (up to the maximum length of an attribute) and is for your own use. For example, you can store information about address formats for use by your address template. Your application or template code resource can read this data by calling the `DirGetExtendedDirectoriesInfo` function.

4. Call the `DirAddDSAM` function, passing it the CSAM's name and signature and the file system specification of the CSAM file. You can use the `kDETcmdGetTemplateFSSpec` template callback function to obtain the file system specification. (Template callback functions are described in "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces.*) The `DirAddDSAM` function creates a CSAM record, starts the CSAM driver (if it's not already running), and returns the creation ID of the CSAM record. Your CSAM driver's Open routine should call the `DirInstantiateDSAM` function at this time, providing it the entry point into your CSAM.

5. Determine the name and discriminator of the catalog and then call the `DirAddDSAMDirectory` function, passing it the CSAM record's creation ID, the Combined record's creation ID, the catalog's name and discriminator, and the capability flags that indicate the abilities of the catalog and CSAM. The catalog name must be unique; the record name is the same as the catalog name and cannot be changed. You can use the `kDETcmdGetDSSpec` template callback function to determine the creation ID of the Combined record. The `DirAddDSAMDirectory` function writes to your Combined record the capability flags, catalog discriminator, and parent CSAM attributes. It then calls your CSAM's catalog service routine with the `kDirAddDSAMDirectory` request code so that the CSAM receives all the information about the catalog that you passed to the `DirAddDSAMDirectory` function. The CSAM can use the record's creation ID to read any of the attributes in the Combined record.

6. If the catalog requires a user name and password, call the `OCESetupAddDirectoryInfo` function to add that information to the Combined record. You provide the Combined record's creation ID, the record ID that represents the user, and the password. The function encrypts the password. You can change this information later by calling the `OCESetupChangeDirectoryInfo` function and extract it by calling the `OCESetupGetDirectoryInfo` function. These functions are described in the chapter "Authentication Manager" in *Inside Macintosh: AOCE Application Interfaces.*

7. If the name of the User record does not correspond to the user's account name in the external catalog, you can also call the `DirAddAttributeValue` function to add a native name attribute. This attribute contains the user's name or account name in the external catalog. This name is for your own use; it is not read by the AOCE software. An application or template code resource can read this name by calling the `OCESetupGetDirectoryInfo` function.

Adding the Mail Service

Once you've added the catalog service, you can add the mail service. Perform the following steps to do so.

8. Determine if an MSAM record exists for your MSAM. You can do this by comparing your MSAM file's file ID with the file ID stored in the gateway file ID attribute in each MSAM record currently in the PowerTalk Setup catalog.

9. If no MSAM record exists for your MSAM, create one and add a version attribute and a gateway file ID attribute to it.

   Listing 4-2 shows a function that compares file IDs, creates the MSAM record if necessary (adding the two attributes), and returns the record's creation ID.

Listing 4-2   Matching an MSAM file ID

```
#define kEnumBufferSize 1024
#define kInitialLookupBuffer 256

struct LookupInfo {
   unsigned long  fileID;
   Boolean      found;
};

#ifndef __cplusplus
typedef struct LookupInfo LookupInfo;
#endif

struct EnumInfo {
   short         setupRef;
   unsigned long  fileID;
   CreationID     msamCID;
   DirEnumSpec    lastEnumSpec;
   RString        name;
   RString        type;
};

#ifndef __cplusplus
typedef struct EnumInfo EnumInfo;
#endif

/* Return the given file's file ID. */

OSErr DoGetIDFromFSSpec(const FSSpec *spec, unsigned long *id)
   {
   OSErr err;
```

```
    CInfoPBRec pb;

    pb.dirInfo.ioCompletion = nil;
    pb.dirInfo.ioVRefNum = spec->vRefNum;
    pb.dirInfo.ioNamePtr = spec->name;
    pb.dirInfo.ioFDirIndex = 0;
    pb.dirInfo.ioDrDirID = spec->parID;
    err = PBGetCatInfoSync(&pb);
    if (err == noErr)
        {
        *id = pb.dirInfo.ioDrDirID;
        }
    return err;
    }

/* Return the dsRefNum of the Setup catalog. */

OSErr DoGetSetupDirectoryRefNum(short *refNum)
    {
    OSErr err;
    DirParamBlock dspb;

    err = DirGetOCESetupRefNum(&dspb, false);

    if (err == noErr)
        {
        *refNum = dspb.header.dsRefNum;
        }

    return err;
    }


/* Does this record have the matching file ID as an attribute? */

pascal Boolean DoLookupCB(long lInfo, const Attribute *thisAttribute)
    {
    LookupInfo* info = (LookupInfo*) lInfo;

    if (info->fileID == * (long*) thisAttribute->value.bytes)
        {
        info->found = true;
        }
```

```
   /* Stop the parse once you find an attribute value with the matching
      file ID. */
   return info->found;
   }

/* Do a lookup into the given record in the Setup catalog to
    see if its kGatewayFileIDAttrTypeNum attribute stores the given
    file ID. */

Boolean DoRecordHasFileID(const LocalRecordID *lrid, short setupRef,
                          unsigned long fileID)
   {
   OSErr err;
   LookupInfo info;
   DirParamBlock dspb;
   RecordID rid;
   RecordIDPtr recordList[1];
   AttributeTypePtr attrTypeList[1];
   Boolean includeStartingPoint = false;
   unsigned long bufferSize = kInitialLookupBuffer;
   void *dataBuffer;

   info.fileID = fileID;
   info.found = false;

   rid.local = *lrid;   /* The lookup requires a RID, not a local RID. */
   rid.rli = nil;

   recordList[0] = &rid;
   attrTypeList[0] = OCEGetIndAttributeType(kGatewayFileIDAttrTypeNum);


/* Create a buffer that's big enough to get at least one attribute value. */

   dataBuffer = NewPtr(bufferSize);
   err = MemError();
   if (err == noErr)
      {
      *(long *)&dspb.lookupGetPB.serverHint = 0;
      dspb.lookupGetPB.dsRefNum = setupRef;
      dspb.lookupGetPB.aRecordList = recordList;
      dspb.lookupGetPB.attrTypeList = attrTypeList;
      dspb.lookupGetPB.recordIDCount = 1;
```

```
      dspb.lookupGetPB.attrTypeCount = 1;
      dspb.lookupGetPB.includeStartingPoint = false;
      dspb.lookupGetPB.getBuffer = dataBuffer;
      dspb.lookupGetPB.getBufferSize = bufferSize;
      dspb.lookupGetPB.startingRecordIndex = 1;
      dspb.lookupGetPB.startingAttrTypeIndex = 1;
      OCESetCreationIDtoNull(&dspb.lookupGetPB.startingAttribute.cid);

      dspb.lookupParsePB.clientData = &info;
      dspb.lookupParsePB.eachRecordID = nil;
      dspb.lookupParsePB.eachAttrType = nil;
      dspb.lookupParsePB.eachAttrValue = DoLookupCB;

      do
         {
         err = DirLookupGet(&dspb,false);

         if ((err == noErr) || (err == kOCEMoreData))
            {
            err = DirLookupParse(&dspb,false);
            }
         } while (err == kOCEMoreData);

      DisposePtr((Ptr) dataBuffer);
      }

   return info.found;
   }


/* Check whether the current record is your MSAM record. To do so, check
   whether the current record's kGatewayFileIDAttrTypeNum attribute matches
   the file ID of your MSAM file. */

pascal Boolean DoEnumCB(long eData, const DirEnumSpec *enumSpecPtr)
   {
   Boolean found;
   EnumInfo* enumData = (EnumInfo*) eData;

   /* First save your current DirEnumSpec so that you can continue
      if the buffer couldn't hold all the records. */
   BlockMove(enumSpecPtr, &enumData->lastEnumSpec, sizeof(DirEnumSpec));
   OCECopyRString(enumSpecPtr->u.recordIdentifier.recordName,
```

```
 &enumData->name, kRStringMaxBytes);
OCECopyRString(enumSpecPtr->u.recordIdentifier.recordType,
 &enumData->type, kRStringMaxBytes);
enumData->lastEnumSpec.u.recordIdentifier.recordName = &enumData->name;
enumData->lastEnumSpec.u.recordIdentifier.recordType = &enumData->type;

/* Does this record have the matching file ID? */
found = DoRecordHasFileID(&enumSpecPtr->u.recordIdentifier,
                          enumData->setupRef, enumData->fileID);

/* If so, save its creation ID. */
if (found)
    {
    OCECopyCreationID(&enumSpecPtr->u.recordIdentifier.cid,
     &enumData->msamCID);
    }

/* Stop the parse once you find a record with the matching file ID. */
return found;
}

/* Enumerate all MSAM records in the Setup catalog, looking
    for one that corresponds to the MSAM with the given file ID. */

OSErr DoFindMSAMRecordWithFileID(short setupRef, unsigned long fileID,
                                 CreationID *msamCID)
    {
    OSErr err;
    DirParamBlock dspb;
    void *buffer;
    RString *recordType;
    EnumInfo enumData;

    buffer = NewPtr(kEnumBufferSize);
    err = MemError();

    if (err == noErr)
        {
        recordType = OCEGetIndRecordType(kMSAMRecTypeNum);

        enumData.fileID = fileID;
        enumData.setupRef = setupRef;
        OCESetCreationIDtoNull(&enumData.msamCID);
```

```
      *(long *)&dspb.enumerateGetPB.serverHint = 0;
      dspb.enumerateGetPB.dsRefNum = setupRef;
      dspb.enumerateGetPB.clientData = &enumData;
      dspb.enumerateGetPB.aRLI = nil;
      dspb.enumerateGetPB.startingPoint = nil;
      dspb.enumerateGetPB.sortBy = kSortByName;
      dspb.enumerateGetPB.sortDirection = kSortForwards;
      dspb.enumerateGetPB.nameMatchString = nil;
      dspb.enumerateGetPB.typesList = &recordType;
      dspb.enumerateGetPB.typeCount = 1;
      dspb.enumerateGetPB.enumFlags = kEnumDistinguishedNameMask;
      dspb.enumerateGetPB.includeStartingPoint = false;
      dspb.enumerateGetPB.matchNameHow = kMatchAll;
      dspb.enumerateGetPB.matchTypeHow = kExactMatch;
      dspb.enumerateGetPB.getBuffer = buffer;
      dspb.enumerateGetPB.getBufferSize = kEnumBufferSize;

      dspb.enumerateParsePB.eachEnumSpec = DoEnumCB;

      do
         {
         err = DirEnumerateGet(&dspb, false);
         if ((err == noErr) || (err == kOCEMoreData))
            {
            err = DirEnumerateParse(&dspb, false);
            }
         dspb.enumerateGetPB.startingPoint = &enumData.lastEnumSpec;
         } while (err == kOCEMoreData);

      DisposPtr((Ptr) buffer);
      }

   OCECopyCreationID(&enumData.msamCID, msamCID);

   return err;
   }

/* Create an MSAM record, returning the record's creation ID. */

OSErr DoCreateMSAMRecord(short setupRef, CreationID *msamCID)
   {
   OSErr err;
   RString name;
```

```
    RecordID rid;
    DirParamBlock dspb;

    OCEPToRString("\pMy MSAM", smRoman, &name, kRStringMaxBytes);
    rid.local.recordName = &name;
    rid.local.recordType = OCEGetIndRecordType(kMSAMRecTypeNum);
    OCESetCreationIDtoNull(&rid.local.cid);
    rid.rli = nil;

    *(long *)&dspb.addRecordPB.serverHint = 0;
    dspb.addRecordPB.dsRefNum = setupRef;
    dspb.addRecordPB.aRecord = &rid;
    dspb.addRecordPB.allowDuplicate = true;

    err = DirAddRecord(&dspb, false);

    OCECopyCreationID(&rid.local.cid, msamCID);

    return err;
    }

/* Add an attribute value to the given record in the Setup catalog. */

OSErr DoAddAttribute(short setupRef,
            const CreationID *recordCID,
            const AttributeType *attrType,
            unsigned long length,
            Ptr bytes)
    {
    OSErr err;
    RecordID rid;
    Attribute attr;
    DirParamBlock dspb;

    rid.local.recordName = nil;
    rid.local.recordType = nil;
    OCECopyCreationID(recordCID, &rid.local.cid);
    rid.rli = nil;

    OCECopyRString((RString*) attrType, (RString*) &attr.attributeType,
                    kAttributeTypeMaxBytes);
    OCESetCreationIDtoNull(&attr.cid);
    attr.value.tag = typeBinary;
```

```
    attr.value.dataLength = length;
    attr.value.bytes = bytes;

    *(long *)&dspb.addAttributeValuePB.serverHint = 0;
    dspb.addAttributeValuePB.dsRefNum = setupRef;
    dspb.addAttributeValuePB.aRecord = &rid;
    dspb.addAttributeValuePB.attr = &attr;

    err = DirAddAttributeValue(&dspb, false);

    return err;
    }

/* Given an FSSpec representing an MSAM file, return the corresponding
    MSAM record's creation ID, creating the MSAM record if necessary. */

OSErr DoGetMSAMCreationID(const FSSpec *spec, CreationID *msamCID)
    {
    OSErr err;
    unsigned fileID;
    short setupRef;
    long version = 1;

    err = DoGetIDFromFSSpec(spec, &fileID);

    if (err == noErr)
        {
        err = DoGetSetupDirectoryRefNum(&setupRef);
        }

    if (err == noErr)
        {
        err = DoFindMSAMRecordWithFileID(setupRef, fileID, msamCID);
        }

    /* If you couldn't find the record, create it. */
    if ((err == noErr) && OCEEqualCreationID(msamCID, OCENullCID()))
        {
        err = DoCreateMSAMRecord(setupRef, msamCID);

        if (err == noErr)
            {
```

```
        /* Add the gateway file ID attribute. */
        err = DoAddAttribute(setupRef, msamCID,OCEGetIndAttributeType
                            (kGatewayFileIDAttrTypeNum), sizeof(long),
                            (Ptr) &fileID);
        }

    if (err == noErr)
        {

        /* Add the version attribute. */
        err = DoAddAttribute(setupRef, msamCID, OCEGetIndAttributeType
                            (kVersionAttrTypeNum), sizeof(long),
                            (Ptr) &version);
        }
    }

    return err;
    }

void Initialize()
{
    InitGraf((Ptr) &qd.thePort);
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(nil);
    InitCursor();
} /* initialize */

main()
    {
    OSErr err;
    StandardFileReply reply;
    CreationID cid;

    Debugger();
    MaxApplZone(); /* expand the heap so that code segments load at the top */

    Initialize();  /* initialize the program */

    StandardGetFile(nil, 0, nil, &reply);
    if (reply.sfGood)
```

```
      {
      err = DoGetMSAMCreationID(&reply.sfFile, &cid);
      Debugger();
      }
   }
```

10. Add a mail service attribute to the MSAM record that points to the Combined record. You can get the record reference, which is a packed record ID, by calling the `kDETcmdGetDSSpec` template callback function.

11. Add a parent MSAM attribute to the Combined record that points to the MSAM record.

The sample function `DoAddRecordReference` in Listing 4-3 illustrates how to insert a record reference into a record in the Setup catalog.

Listing 4-3   Inserting a record reference into a record

```
OSErr DoAddRecordReference(const CreationID *recordToAddReference,
                 const AttributeType *attrType,
                 const CreationID *referenceCID)
   {
   OSErr err;
   short setupRef;
   RecordID recordReference;
   PackedRecordID *packedReference;
   unsigned short size;

   err = DoGetSetupDirectoryRefNum(&setupRef);

   if (err == noErr)
      {
      recordReference.local.recordName = nil;
      recordReference.local.recordType = nil;
      OCECopyCreationID(referenceCID, &recordReference.local.cid);
      recordReference.rli = nil;

      size = OCEPackedRecordIDSize(&recordReference);
      packedReference = (PackedRecordID*) NewPtr(size);
      err = MemError();
      }

   if (err == noErr)
      {
      OCEPackRecordID(&recordReference, packedReference, size);
```

```
        err = DoAddAttribute(setupRef, recordToAddReference, attrType, size,
                             (Ptr) packedReference);
    }

    return err;
}
```

12. Add a standard slot information attribute to the Combined record. This attribute contains a `MailStandardSlotInfoAttribute` structure. See the chapter "Messaging Service Access Modules" in this book for a description of this data structure.

13. Call the `MailCreateMailSlot` function asynchronously to tell the MSAM to create the new slot. You pass this function the MSAM record creation ID, the Combined record creation ID, and some other information. You must call the `kDETcmdBusy` callback routine while waiting for the `MailCreateMailSlot` function to complete. The AOCE system launches the MSAM, which generates a unique slot ID for the new slot and adds it to the Combined record in a slot ID attribute.

14. Delete the "aoce Unconfigured" attribute from the Combined record, because the service is now configured. At this point, the mail and catalog services are available to the user.

Your setup information pages obtain from the user whatever information is required to access the external messaging system, such as the user's account name and password, a telephone number, connection information, and so forth.

Table 4-2 on page 4-65, Table 4-3 on page 4-66, and Table 4-6 on page 4-70 summarize the contents of the MSAM, CSAM, and Combined records.

Adding a Mail Service Only

This section tells you how to add a mail service without adding a catalog service. Read this section if you are providing only an MSAM. It explains what your setup template must do to allow a user to add and configure a mail service.

The setup template for an MSAM includes main aspect templates for a Mail Service record and a Catalog record. Your code resource for the Mail Service record should check whether an MSAM record already exists. If the MSAM record does not exist, your template's code resource must create that record as well.

Listing 4-4 illustrates a setup template for a mail service. Except for changing several strings from "combined" to "mail," using the constant `kMailServiceRecTypeBody` instead of `kCombinedRecTypeBody`, and specifying `notDSAM` instead of `servesDSAM` in the `'sami'` resource, Listing 4-4 is identical to Listing 4-1 on page 4-6.

Listing 4-4   Mail service setup template

```
#define SystemSevenOrLater 1

#include "Types.r"
#include "OCETemplates.h"
#include "OCE.r"

#define kSurfWriterAspect        1000
#define kSurfWriterInfoPage      1250

#define kSignature               'WAVE'
#define kServiceRecordType       kMailServiceRecTypeBody "WAVE"
#define kAspectName              "SurfWriter_MS_Aspect"
#define kInfoPageName            "SurfWriter_MS_Info_Page"

#define kWindowWidth             259
#define kWindowHeight            200
#define kZeroRect                {0, 0, 0, 0}
#define kDoubleLineLeft          kDETSubpageIconLeft
#define kDoubleLineRight         kWindowWidth - kDETSubpageIconLeft
#define kDoubleLineTop           kTopBorder + kFieldHeight + 8
#define kDoubleLineBottom        kDoubleLineTop + 1
#define kDoubleLineRect          {kDoubleLineTop, kDoubleLineLeft,\
                                   kDoubleLineBottom, kDoubleLineRight}


resource 'deta' (kSurfWriterAspect, purgeable)
{
   0, dropCheckConflicts, isMainAspect
};

resource 'rstr' (kSurfWriterAspect + kDETTemplateName, purgeable)
{
   kAspectName
};

resource 'rstr' (kSurfWriterAspect + kDETRecordType, purgeable)
{
   kServiceRecordType
};
```

```
resource 'rstr' (kSurfWriterAspect + kSAMAspectKind, purgeable)
{
    "SurfWriter Mail Service"
};


resource 'rstr' (kSurfWriterAspect + kDETAspectKind, purgeable)
{
    "SurfWriter Mail Service"
};


resource 'rstr' (kSurfWriterAspect + kDETAspectName, purgeable) {
    "unconfigured SurfWriter Mail"
};


resource 'rstr' (kSurfWriterAspect + kSAMAspectUserName, purgeable) {
    "SurfWriter User"
};



resource 'sami' (kSurfWriterAspect + kSAMAspectSlotCreationInfo, purgeable)
{
    2,                                   // max number of catalogs/slots
    kSignature,                          // catalog signature, MSAM type
    servesMSAM,                          // an MSAM template
    notDSAM,                             // not a CSAM template
    "SurfWriter Mail Service ",          // display when user clicks Add
    "untitled SurfWriter Mail Service"   // new record name
};

// Custom window

resource 'detw' (kSurfWriterAspect + kDETAspectInfoPageCustomWindow,
                 purgeable)
    {
    {-1, -1, kWindowHeight, kWindowWidth},
    includePopup
    };

// Include code resource

include "SurfWriterCode" 'detc'(0) as
    'detc'(kSurfWriterAspect+kDETAspectCode, purgeable);
```

```
// Include icons

include "SurfWriterIcons" 'ICN#'(0) as
                  'ICN#'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'icl4'(0) as
                  'icl4'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'icl8'(0) as
                  'icl8'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'ics#'(0) as
                  'ics#'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'ics4'(0) as
                  'ics4'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'ics8'(0) as
                  'ics8'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'SICN'(0) as
                  'SICN'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);


//-------------------------------------------------------------------------
// Info-page

resource 'deti' (kSurfWriterInfoPage, purgeable)
{
   kDefaultSortIndex,
   kZeroRect,
   noSelectFirstText,
   {
      kDETNoProperty, kDETNoProperty, kSurfWriterInfoPage;
   },
   {
   }
};

resource 'rstr' (kSurfWriterInfoPage + kDETTemplateName, purgeable) {
   kInfoPageName
};

resource 'rstr' (kSurfWriterInfoPage + kDETRecordType, purgeable) {
   kServiceRecordType
};

resource 'rstr' (kSurfWriterInfoPage + kDETInfoPageName, purgeable) {
   "SurfWriter Mail Service"
};
```

```
resource 'rstr' (kSurfWriterInfoPage + kDETInfoPageMainViewAspect,
               purgeable) {
   kAspectName
};


resource 'detv' (kSurfWriterInfoPage, "subpageview", purgeable)
   {
   {
   kDoubleLineRect, kDETNoFlags, kDETNoProperty,
   Box { kDETBoxIsGrayed };

   kDETSubpageIconRect, kDETNoFlags, kDETAspectMainBitmap,
   Bitmap { kDETLargeIcon };
   };
   };
```

During system initialization, the Key Chain loads all setup templates. As specified by the 'sami' resource in Listing 4-4, the Key Chain offers the user the choice "SurfWriter Mail Service" when the user clicks the Add button. When the user selects this choice, the Key Chain scans the Setup catalog looking for catalogs associated with the new mail service. It identifies associated catalogs by looking for Catalog records whose type ends in the catalog signature you specified in the 'sami' resource of your setup template. In our example, these records would be of type "aoce DirectoryWAVE". If the Key Chain finds any such catalogs that do not already have an associated mail service, it displays a dialog box (Figure 4-1) allowing the user to join one of these existing catalogs.

Figure 4-1   Catalog-choice dialog box
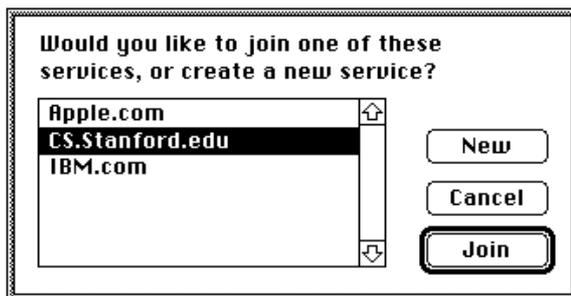
If the user selects one of these catalogs, the Key Chain adds a Mail Service record to the Setup catalog (using the aspect template you provided) and puts an associated catalog attribute in that record pointing to the Catalog record the user selected. The Key Chain also adds to the Catalog record an associated mail service attribute pointing to the Mail Service record it just created.

If the user clicks New or if the Key Chain does not find any associated catalogs that do not already have an associated mail service, it adds to the Setup catalog both a Mail Service record and a new Catalog record. It puts an "aoce Fake" attribute (attribute type index `kFakeAttrTypeNum`) in the Catalog record, indicating that the MSAM does not have a CSAM associated with it. The Key Chain also puts an associated catalog attribute in the Mail Service record and an associated mail service attribute in the Catalog record.

As specified by Listing 4-4, the new Mail Service record's name is "untitled SurfWriter Mail Service", and its type is "aoce Mail ServiceWAVE". The Key Chain writes three additional attributes into this record, as follows:

n the associated catalog attribute

n an attribute of type "aoce Unconfigured" (attribute type index `kUnconfiguredAttrTypeNum`) indicating that the service has not yet been set up

n a version attribute that contains the version number of the Key Chain at the time it created the record

The Key Chain adds a line to the Key Chain window representing the new record. The line includes the key icon used by the Key Chain, the service name you specified in the `kDETAspectName` resource ("unconfigured SurfWriter Mail" in Listing 4-4), and whatever default values your template provides for the Name and Kind fields in the `kSAMAspectUserName` and `kSAMAspectKind` resources ("SurfWriter user" and "SurfWriter Mail Service" in Listing 4-4).

Your code resource should create no new records or attributes at this time. However, when the user opens the Key Chain entry, the Key Chain opens your information page and calls your code resource with the `kDETcmdInstanceInit` routine selector. Your setup template must follow the steps in the following two sections to set up a mail service.

The attributes in the Mail Service record are summarized in Table 4-4 on page 4-67, and the attributes in the Catalog record are summarized in Table 4-5 on page 4-68. Use the Catalog Manager function `DirAddAttributeValue` to add attributes to a record. The function is described in the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces*; see the `DoAddAttribute` function on page 4-18 in Listing 4-2 for an example of its use.

Setting Up the Associated Catalog Service

If the MSAM does not have a CSAM associated with it (that is, if the Catalog record contains an "aoce Fake" attribute), you need to add some configuration information to the Catalog record and rename the record. You can find your Catalog record by unpacking the record reference stored in the associated catalog attribute in your Mail Service record. To add a CSAM and associate it with an existing MSAM, see "Adding a Catalog Service Only" on page 4-28.

1. Write the comment attribute. This is a string that you can use for any purpose. An application developer can read this string by calling the `DirGetExtendedDirectoriesInfo` function; see the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* for a description of this function.

2. Write the real name attribute, containing the external name of the catalog. This name is for your own use; it is not read by the AOCE software. An application developer can read this name by calling the `DirGetExtendedDirectoriesInfo` function. Whereas the catalog name you provide to the `DirSetNameAndType` function (see step 4) must be unique within the AOCE system, the "real" name need not be. For example, the user might have accounts on two different SurfWriter mail servers. Each would have to have a distinct name for display in the Key Chain, but the Catalog records for both could contain the name SurfWriter Mail for the real name attribute.

3. If you wish, you can write a private data attribute. This attribute can contain binary data of any length (up to the maximum length of an attribute) and is for your own use. For example, you can store information about address formats for use by your address template. Your application or template code resource can read this data by calling the `DirGetExtendedDirectoriesInfo` function.

4. Determine the name of the catalog to be used in the Setup catalog, and call the `DirSetNameAndType` function to set the name of the Catalog record to be the same as the catalog name. This name must be unique within the Setup catalog, and once set, it must never be changed.

5. Call the `OCESetupAddDirectoryInfo` function or the `DirAddAttributeValue` function to add the user's record ID attribute to the Catalog record. You must provide the Catalog record's creation ID and a record ID that includes the catalog's name. You can leave blank the other fields in the record ID and the password if the MSAM does not require an account name and password. The `OCESetupAddDirectoryInfo` function is described in the chapter "Authentication Manager" in *Inside Macintosh: AOCE Application Interfaces*.

6. If the name of the User record does not correspond to the user's account name, you can also call the `DirAddAttributeValue` function to add a native name attribute to the Catalog record. This attribute contains the user's name or account name in the external system. This name is for your own use; it is not read by the AOCE software. An application or template code resource can read this name by calling the `OCESetupGetDirectoryInfo` function, described in the chapter "Authentication Manager" in *Inside Macintosh: AOCE Application Interfaces*.

7. Write the discriminator attribute, giving it the value of the address extension type of the messaging system to which your MSAM provides access.

Setting Up the Mail Service

You next need to activate the mail slot. You do this by following the steps described in "Adding the Mail Service" beginning on page 4-12.

Adding a Catalog Service Only

This section tells you how to add a catalog service without adding a mail service. Read this section if you are providing only a CSAM. It explains what your setup template must do to allow a user to add and configure a catalog service.

The setup template for a CSAM includes a main aspect template for a Catalog record.

A basic setup template for a catalog service is identical to the mail service template in Listing 4-4 on page 4-23 with these two exceptions: the word "mail" is replaced with "catalog" throughout, and the following `'sami'` resource is used:

```
resource 'sami' (kSurfWriterAspect + kSAMAspectSlotCreationInfo, purgeable)
{
    2,                                          // max number of catalogs/slots
    kSignature,                                 // catalog signature
    notMSAM,                                    // not an MSAM template
    servesDSAM,                                 // a CSAM template
    "SurfWriter Catalog Service ",              // display when user clicks Add
    "untitled SurfWriter Catalog Service"       // new record name
};
```

During system initialization, the Key Chain loads all setup templates. As specified by the preceding `'sami'` resource, the Key Chain offers the user the choice "SurfWriter Catalog Service" when the user clicks the Add button. When the user selects this choice, the Key Chain scans the Setup catalog looking for unconfigured Catalog records of the type specified by the setup template. In this example, these records would be of type "aoce DirectoryWAVE". It identifies an unconfigured Catalog record by looking for an attribute of type "aoce Fake" in the record. If the Key Chain finds any such Catalog records, it displays a dialog box allowing the user to join one of these existing catalogs (see Figure 4-1 on page 4-26).

If the user selects one of these catalogs, the Key Chain replaces the "aoce Fake" attribute with an "aoce Joined" attribute (attribute type index `kJoinedAttrTypeNum`).

If the user clicks New or if the Key Chain does not find any unconfigured catalogs of the right type, it adds to the Setup catalog a new Catalog record, using the aspect template you provided in your setup template. It puts an "aoce Unconfigured" attribute (attribute type index `kUnconfiguredAttrTypeNum`) in the Catalog record, indicating that it has not yet been configured.

Your code resource should create no new records or attributes at this time. However, when the user opens the Key Chain entry, the Key Chain opens your information page and calls your code resource with the `kDETcmdInstanceInit` routine selector. Your setup template must then set up the catalog as follows:

1. Determine whether you are joining an existing catalog by checking for an "aoce Joined" attribute.

2. If you are joining an existing catalog, the catalog name (which is the record name) is already set and must not be changed. If there is no "aoce Joined" attribute, determine the catalog name and call the `DirSetNameAndType` function to set the name of the Catalog record to be the same as the catalog name. This name must be unique within the Setup catalog, and once set, it must never be changed.

3. If you are joining an existing catalog, check the attributes in the Catalog record (see Table 4-5 on page 4-68) for existing information.

4. Preserve any existing information and follow the steps in "Adding the Catalog Service" beginning on page 4-10 as appropriate to provide any attributes that don't already exist.

5. Delete the "aoce Unconfigured" or "aoce Joined" attribute.

Modifying an Existing Service

Each time the user restarts his or her system, the Key Chain calls your code resource with the `kDETcmdInit` routine selector. At that time you should obtain the file system specifier (`FSSpec` structure) for your MSAM file and make sure that the file ID is the same as that saved in the gateway file ID attribute in the MSAM record (Table 4-2 on page 4-65). If the user has replaced the MSAM record, the file ID for your MSAM file will not match the one saved in the gateway file ID attribute. In this case, you must replace the gateway file ID attribute with the correct file ID. Because the Collaboration toolbox reads the file IDs from the MSAM record before calling your code resource, it cannot open the new MSAM file until the user restarts the system. Therefore, after updating the MSAM record, your code resource must display a dialog box telling the user to restart the system in order to use the new file.

Your setup template must provide one or more information pages that allow a user to modify an existing service. The implementation of this feature is up to you.

Writing and Modifying Addresses

To allow users to add addresses of the type used by your MSAM to their User records, you must provide an address template. An address template consists of a main aspect template for addresses of the type used by your mail service plus at least one information page. All addresses used by AOCE are in the format of an `OCEPackedRecipient` data structure, as described in the chapter "Messaging Service Access Modules" in this book.

Whenever practical, an address information page should provide two alternative but equivalent methods of entering addresses: a set of fields and a single input string. Internally, the form of the address is defined by the MSAM providing the address template. Figure 4-2 shows the two views of an address template for the SurfWriter application.

Figure 4-2 Alternate forms of a single address information page
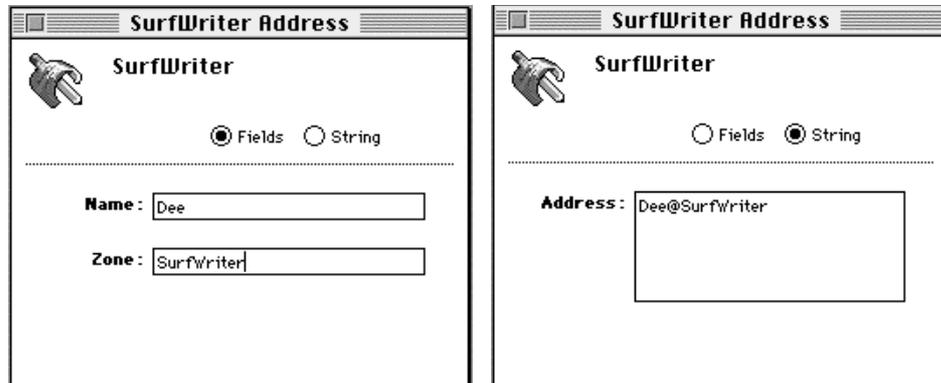
Writing an Address Template

Listing 4-5 shows the AOCE aspect and information page templates that create the information page shown in Figure 4-2. The aspect template defines an attribute of type `kMailSlotsAttrTypeBody`. The Collaboration toolbox expects all addresses to be stored in a multivalued attribute of that type. The attribute tag specifies the address format. In this example, the attribute tag is `'WAVE'`, the signature of the fictional application SurfWriter. The lookup table lists the properties that must be processed by the code resource and uses a custom lookup-table element so that the CE calls the code resource each time it processes the lookup table.

The information page in Listing 4-5 has two conditional views to provide the two methods of entering addresses: Fields or String. One view or the other is displayed, depending on which radio button the user clicked.

Because an address is of type `OCEPackedRecipient`, which is a packed, private data structure, you must use a code resource to pack and unpack this structure. The code resource can call the utility routines described in the chapter "Interprogram Messaging Manager" in *Inside Macintosh: AOCE Application Interfaces* to pack and unpack `OCEPackedRecipient` structures. The code resource also converts between the fields and string forms of the address. Listing 4-6 on page 4-41 shows the code resource for the address template in Listing 4-5.

Listing 4-5 Address template

```
// Defines for the headers of the templates

#define kZeroRect {0, 0, 0, 0}

#define kWindowWidth 259
#define kWindowHeight 200
```

```
#define kDETMenuLeft kDETSubpageIconRight + 16  // left edge of subpage menu
#define kDETMenuRight kDETMenuLeft + 150        // right edge of subpage menu
#define kDETMenuBottom kDETSubpageIconTop + 22  // bottom of subpage menu

#define kTopBorder kDETSubpageIconBottom + 8    // top of first item
#define kFieldHeight 16                         // height of fields
#define kMenuWidth 75                           // width of "View as" menu
#define kMenuTitleWidth 60                  // width allowed for field titles
#define kFieldTitleSeparator 5

// These defines are for the "View As:" radio buttons at the top of
// most of the address templates.

#define kViewAsTextWidth 50
#define kViewButton1Width 43
#define kViewButton2Width 45
#define kFirstFieldTop kHeaderBottom + 10

#define kViewAsTextTop kTopBorder + 1
#define kViewAsTextLeft kDETMenuLeft
#define kViewAsTextBottom kViewAsTextTop + 14
#define kViewAsTextRight kViewAsTextLeft + kViewAsTextWidth
#define kViewAsTextRect {kViewAsTextTop, kViewAsTextLeft, kViewAsTextBottom,
                         kViewAsTextRight}

#define kFieldsButtonTop kTopBorder
#define kFieldsButtonBottom kFieldsButtonTop + kFieldHeight
#define kFieldsButtonLeft kViewAsTextRight + kFieldTitleSeparator
#define kFieldsButtonRight kFieldsButtonLeft + kViewButton1Width
#define kFieldsButtonRect {kFieldsButtonTop, kFieldsButtonLeft,
                           kFieldsButtonBottom, kFieldsButtonRight}

#define kStringButtonTop kTopBorder
#define kStringButtonBottom kStringButtonTop + kFieldHeight
#define kStringButtonLeft kFieldsButtonRight + kFieldTitleSeparator +
                          kFieldTitleSeparator
#define kStringButtonRight kStringButtonLeft + kViewButton2Width
#define kStringButtonRect {kStringButtonTop, kStringButtonLeft,
                           kStringButtonBottom, kStringButtonRight}

// These defines are for miscellaneous components, such as the dotted line at
// the top of the template.
```

```
#define kDoubleLineLeft kDETSubpageIconLeft
#define kDoubleLineRight kWindowWidth - kDETSubpageIconLeft
#define kDoubleLineTop kTopBorder + kFieldHeight + 8
#define kDoubleLineBottom kDoubleLineTop + 1
#define kDoubleLineRect {kDoubleLineTop, kDoubleLineLeft, kDoubleLineBottom,
                         kDoubleLineRight}

#define kHeaderBottom kDoubleLineBottom

#define kLeftBorder 35                    // left border
#define kTitleWidth 45                    // width allowed for field titles
#define kFieldLeft kLeftBorder + kTitleWidth
                                          // left border of field items
#define kTextLeft 10                      // left border of static text items
#define kTextRight kFieldLeft - kFieldTitleSeparator
                                          // right border of static text items
#define kFieldWidth 155                   // width of text fields

// Error messages for use by the code resource
resource 'STR#' (kSurfInfoPageAspect, purgeable) {
    {
    /* [1] */    "An unspecified problem occurred.",
    /* [2] */    "This address must contain a name. Please enter a name in "
                 "the appropriate field.",
    }
};

// Properties

#define prMyName (kDETFirstDevProperty + 0)
#define prMyZone (kDETFirstDevProperty + 1)
#define prMyAddress (kDETFirstDevProperty + 2)
#define prViewMenu (kDETFirstDevProperty + 3)
#define prInited (kDETFirstDevProperty + 4)
#define prOldName (kDETFirstDevProperty + 5)
#define prOldZone (kDETFirstDevProperty + 6)
#define prOldAddress (kDETFirstDevProperty + 7)
#define prDefaultName (kDETFirstDevProperty+8)
#define prDefaultZone (kDETFirstDevProperty+9)
#define prDefaultDisplayName (kDETFirstDevProperty+10)
```

```
//-------------------------------------------------------------------------
// These custom lookup-table elements cause the Catalogs Extension to call
// the code resource.

#define kProcessData 'Hey!'
#define kPostProcessData 'Done'

#define kNullNameError 1000

//-------------------------------------------------------------------------*/
// Aspect template
include "SurfAddressCode" 'detc'(0) as 'detc'(kSurfInfoPageAspect +
    kDETAspectCode, purgeable);

// Aspect template signature resource
resource 'deta' (kSurfInfoPageAspect, purgeable) {
   1000,                  // drop-operation order
   dropCheckConflicts,  // drop check flag
   isMainAspect         // is the main aspect
};

// Template name
resource 'rstr' (kSurfInfoPageAspect + kDETTemplateName, purgeable) {
   kNewSurfAddressAspectName
};

// This template applies to the kMailSlotsAttrTypeBody attribute
resource 'rstr' (kSurfInfoPageAspect + kDETAttributeType, purgeable) {
   kMailSlotsAttrTypeBody
};

// Tag of attribute this applies to
resource 'detn' (kSurfInfoPageAspect + kDETAttributeValueTag){
   'WAVE'
};

// Template kind
resource 'rstr' (kSurfInfoPageAspect + kDETAspectKind, purgeable) {
   "SurfWriter mail address"
};

// String for Add dialog box
resource 'rstr' (kSurfInfoPageAspect + kDETAspectNewMenuName, purgeable) {
```

```
   "SurfWriter"
};


// Category for template
resource 'rst#' (kSurfInfoPageAspect+kDETAspectCategory,purgeable)
{
   {
      kDETCategoryAddressItems,
   }
};


// Open info page automatically when user creates new attribute value.
resource 'detn' (kSurfInfoPageAspect + kDETAspectSublistOpenOnNew){
   1
};


// Attribute tag followed by default new value for attribute. This resource
// must be present if user is allowed to add a new attribute. The code
// resource routine DoCreateNewAttribute (page 4-44) appends the default
// attribute value to the attribute tag to create the new attribute value.
data 'detb' (kSurfInfoPageAspect + kDETAspectNewValue, purgeable) {
   $"5741 5645"                // 'WAVE' tag
};


// Lookup table. Most property values are actually set by the code resource,
// but it is necessary to include them all here so the Catalogs Extension
// to the Finder will know they exist and so that it will call the code
// resource when the user changes their values.
resource 'dett' (kSurfInfoPageAspect + kDETAspectLookup, purgeable)
{{
   { kMailSlotsAttrTypeBody },
   'WAVE',
   useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
      kProcessData,kDETNoProperty,  0; // custom property element,
                                       //  causing CE to call code resource
      'prop', prMyZone, 0;             // declare all the properties
      'prop', prMyName, 0;
      'prop', prMyAddress, 0;
      'prop', kDETAspectName, 0;
```

```
      'awrd', kDETNoProperty, 0;        // align to a word boundary
      kPostProcessData, kDETNoProperty, 0;
                                        // post-process the data
                                           (if necessary)
      'Pref', kDETNoProperty, 0;        // ask CE to save preferred
                                        // mailslot info
      };
}};


// Custom window information
resource 'detw' (kSurfInfoPageAspect + kDETAspectInfoPageCustomWindow,
purgeable)
   {
   {0, 0, kWindowHeight, kWindowWidth},
   includePopup          // this places label in info page
   };



// Default values for some properties

resource 'detn' (kSurfInfoPageAspect + prViewMenu, purgeable) {
   1           // make "fields" the default conditional view
};

resource 'rstr'(kSurfInfoPageAspect + prDefaultName, purgeable) {
   "<Name>"
};
resource 'rstr'(kSurfInfoPageAspect + prDefaultZone, purgeable) {
   "<Zone>"
};
resource 'rstr'(kSurfInfoPageAspect + prDefaultDisplayName, purgeable) {
   "untitled SurfWriter address"
};

// Text for help balloons

// Text for help balloon for the item in a sublist
resource 'rstr' (kSurfInfoPageAspect+kDETAspectWhatIs, purgeable) {
   "Contains a SurfWriter mail address."
};

// Text for help balloon for an alias to the item
resource 'rstr' (kSurfInfoPageAspect+kDETAspectAliasWhatIs, purgeable) {
```

```
      "Contains an alias to a SurfWriter mail address."
};


// Text for help balloons for the properties
resource 'rst#' (kSurfInfoPageAspect+kDETAspectBalloons, purgeable) {
    {
    "Shows the name for this SurfWriter mail address. You can edit this "
       "name.",
    "Shows the name for this SurfWriter mail address. You cannot edit this "
       "name.",
    "Shows the zone location for this SurfWriter mail address. You can edit "
       "this zone name.",
    "Shows the zone location for this SurfWriter mail address. You cannot "
       "edit this zone name.",
    "Shows this SurfWriter mail address as a series of characters. "
       "You can edit this address.",
    "Shows this SurfWriter mail address as a series of characters. "
       "You cannot edit this address.",
    "Controls the display of address information. Click a button to change "
       "the display.",
    "",
    }
};



//-------------------------------------------------------------------------
// Information page
//
#define kTwoProperty kDETFirstConstantProperty + 2

// Information page signature resource
resource 'deti' (kSurfInfoPage, purgeable) {
    1000,                 // sort order
    kZeroRect,            // no sublist
    noSelectFirstText,    // don't select first text field
    {
    kDETNoProperty, kDETNoProperty, kSurfInfoPage;  // common view list
    prViewMenu, kDETOneProperty, kSurfInfoPage + 1; // "fields" view list
    prViewMenu, kTwoProperty, kSurfInfoPage + 2;    // "strings" view list
    },
    {                     // no subview view lists
    }};
```

```
// Template name
resource 'rstr' (kSurfInfoPage + kDETTemplateName, purgeable) {
   kNewSurfAddressInfoPageName
};


// Attribute type
resource 'rstr' (kSurfInfoPage + kDETAttributeType, purgeable) {
   kMailSlotsAttrTypeBody
};


// Name of information page that appears as label on page
resource 'rstr' (kSurfInfoPage + kDETInfoPageName, purgeable) {
   "SurfWriter"
};


// Name of related aspect template
resource 'rstr' (kSurfInfoPage + kDETInfoPageMainViewAspect, purgeable) {
   kNewSurfAddressAspectName
};


// Defines for the rest of the info page

#define kAddressWidth 155            // width of string address field
#define kAddressHeight 64            // height of string address field

#define kStatText1Top kHeaderBottom + kFieldHeight
#define kStatText1Left kTextLeft
#define kStatText1Bottom kStatText1Top + kFieldHeight
#define kStatText1Right kTextRight
#define kStatText1Rect {kStatText1Top, kStatText1Left, kStatText1Bottom,
kStatText1Right}

#define kNameTextTop kStatText1Top
#define kNameTextLeft kFieldLeft
#define kNameTextBottom kStatText1Bottom
#define kNameTextRight kNameTextLeft + kFieldWidth
#define kNameTextRect {kNameTextTop, kNameTextLeft, kNameTextBottom,
kNameTextRight}

#define kStatText2Top kStatText1Bottom + kFieldHeight
#define kStatText2Left kTextLeft
#define kStatText2Bottom kStatText2Top + kFieldHeight
#define kStatText2Right kTextRight
```

```
#define kStatText2Rect {kStatText2Top, kStatText2Left, kStatText2Bottom,
kStatText2Right}

#define kZoneTextTop kStatText2Top
#define kZoneTextLeft kFieldLeft
#define kZoneTextBottom kStatText2Bottom
#define kZoneTextRight kZoneTextLeft + kFieldWidth
#define kZoneTextRect {kZoneTextTop, kZoneTextLeft, kZoneTextBottom,
kZoneTextRight}

#define kAddressTextTop kNameTextTop
#define kAddressTextLeft kNameTextLeft
#define kAddressTextBottom kAddressTextTop + kAddressHeight
#define kAddressTextRight kAddressTextLeft + kAddressWidth
#define kAddressTextRect {kAddressTextTop, kAddressTextLeft,
kAddressTextBottom, kAddressTextRight}

// Nonconditional view
resource 'detv' (kSurfInfoPage, purgeable)
    {
    {
    kDoubleLineRect, kDETNoFlags, kDETNoProperty,
    Box { kDETBoxIsGrayed };

    kDETSubpageIconRect, kDETNoFlags, kDETAspectMainBitmap,
    Bitmap { kDETLargeIcon };

    kViewAsTextRect, kDETNoFlags, kDETNoProperty,
    StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
                         kDETLeft, kDETBold, "View as:" };

    kFieldsButtonRect, kDETEnabled, prViewMenu,
    RadioButton { kDETApplicationFont, kDETApplicationFontSize,
                 kDETLeft, kDETNormal, "Fields", prViewMenu, 1 };

    kStringButtonRect, kDETEnabled, prViewMenu,
    RadioButton { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
                 kDETNormal, "String", prViewMenu, 2 };
    };
    };

// "Fields" conditional view
resource 'detv' (kSurfInfoPage + 1, purgeable)
```

```
   {
   {
   kStatText1Rect, kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
                        kDETRight, kDETBold, "Name:" };

   kStatText2Rect, kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
                        kDETRight, kDETBold, "Zone:" };

   kNameTextRect, kDETNoFlags, prMyName,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
              kDETNormal };

   kZoneTextRect, kDETNoFlags, prMyZone,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
              kDETNormal };
   };
   };

// "String" conditional view
resource 'detv' (kSurfInfoPage + 2, purgeable)
   {
   {
   kStatText1Rect, kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
kDETRight,
                  kDETBold, "Address:" };

   kAddressTextRect, kDETMultiLine, prMyAddress,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
kDETNormal };
   };
   };

//
// Icons

include "AlbumIcons" 'ICN#'(0) as
      'ICN#'(kSurfInfoPageAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'icl4'(0) as
      'icl4'(kSurfInfoPageAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'icl8'(0) as
      'icl8'(kSurfInfoPageAspect + kDETAspectMainBitmap, purgeable);
```

```
include "AlbumIcons" 'ics#'(0) as
      'ics#'(kSurfInfoPageAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'ics4'(0) as
      'ics4'(kSurfInfoPageAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'ics8'(0) as
      'ics8'(kSurfInfoPageAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'SICN'(0) as
      'SICN'(kSurfInfoPageAspect + kDETAspectMainBitmap, purgeable);
```

Writing an Address Template Code Resource

The code resource for the address template shown in Listing 4-5 consists of a dispatcher routine and several routines to handle standard requirements of the CE, as described in the chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces.* The dispatcher routine and main code resource routines are shown in the following section. The sections "Data Input Subroutines for the Address Template" beginning on page 4-47, "Data Output Subroutines for the Address Template" beginning on page 4-51, and "Miscellaneous Subroutines" beginning on page 4-57 show the subroutines called by the main routines and by other subroutines. Subroutines with names beginning with My (for example, MyCreateFieldsFromOCERecipient) are described but not shown here.

Main Routines for the Address Template Code Resource

Listing 4-6 shows the main routines for the code resource. The DoSurfAddress routine is the dispatcher called by the Catalogs Extension (CE). When the CE calls this routine, it calls one of the other routines shown in this section, depending on the routine selector passed by the CE.

Listing 4-6 Main routines of the address template code resource

```
/* Code resource main routine */
pascal OSErr DoSurfAddress(DETCallBlockPtr callBlockPtr)
{
   OSErr err = 1;
/* Process only calls targeted to this template plus untargeted calls. */
   if ((callBlockPtr->protoCall.reqFunction < kDETcmdTargetedCall) ||
(callBlockPtr->protoCall.target.selector == kDETSelf))
   {
      switch (callBlockPtr->protoCall.reqFunction)
      {
         case kDETcmdInit:
            err = DoInitTemplate(callBlockPtr);
            break;
```

```
        case kDETcmdAttributeCreation:
            err = DoCreateNewAttribute(callBlockPtr);
            break;

        case kDETcmdInstanceInit:
            err = DoInitInstance(callBlockPtr);
            break;

        case kDETcmdInstanceExit:
            err = DoExitInstance(callBlockPtr);
            break;

        case kDETcmdPropertyDirtied:
            err = DoPropertyDirty(callBlockPtr);
            break;

        case kDETcmdValidateSave:
            err = DoPrepareToSave(callBlockPtr);
            break;

        case kDETcmdPatternIn:
            err = DoPatternIn(callBlockPtr);
            break;

        case kDETcmdPatternOut:
            err = DoPatternOut(callBlockPtr);
            break;

        default:
            break;
    }
}

    return err;
}

/* ------------------------------------------------------------------------ */
/* Template initialization. Set call-for mask. */
static OSErr DoInitTemplate(DETCallBlockPtr callBlockPtr)
{
    OSErr err = noErr;
    callBlockPtr->init.newCallFors = kDETCallForValidation +
        kDETCallForAttributes + kDETCallForViewChanges;
```

```
   return err;
}


/* ------------------------------------------------------------------------ */
/* Aspect initialization */
static OSErr DoInitInstance(DETCallBlockPtr callBlockPtr)
{
   OSErr err = noErr;
/* Set value of property prInited to false. */
   DoSetInited(callBlockPtr, false);

   return err;
}


/* ------------------------------------------------------------------------ */
/* Exit routine */
static OSErr DoExitInstance(DETCallBlockPtr callBlockPtr)
{
   OSErr err = noErr;
/* Set value of property prInited to false. */
   DoSetInited(callBlockPtr, false);

   return err;
}


/* ------------------------------------------------------------------------ */
/* The CE is about to save property values. */
static OSErr DoPrepareToSave(DETCallBlockPtr callBlockPtr)
{
   OSErr err = kDETDidNotHandle;
   Handle errorStr = nil;
   if (!DoIsInited(callBlockPtr))
      return 1;

/* Check the data to make sure it's valid. */
   err = MyCheckData(callBlockPtr);
   return err;
}


/* ------------------------------------------------------------------------ */
/* The CE calls this routine to process the custom lookup-table element that
   processes input data. This routine is discussed in
```

"Data Input Subroutines for the Address Template" beginning on
page 4-47.*/

```
static OSErr DoPatternIn(DETCallBlockPtr callBlockPtr)
{
    OSErr err = 1;
    Boolean enabled = true;
    /* For input processing only */
    if (callBlockPtr->patternIn.elementType == kProcessData)
    {
        err = DoExtractInformation(callBlockPtr);
    }
    else if (callBlockPtr->patternIn.elementType == kPostProcessData)
    {
        err = DoSetDisplayName(callBlockPtr);
    }

    DoHandleError(callBlockPtr, err);

    return err;
}


/* ------------------------------------------------------------------------ */
/* The CE calls this routine when it is about to add a new attribute value
   to the sublist. This routine gets the default values for the field
   properties and the default string to be displayed in the sublist for the
   attribute value. These values are provided by the template (page 4-36).
   It packs the fields into a string and appends them to the default value
   of the attribute. The CE provides a pointer to this default attribute
   value; it gets the default attribute value from the kDETAspectNewValue
   resource in the aspect template (page 4-35). See
   beginning on page 4-51 for a discussion of the DoWriteNameAndZone
   subroutine. */

static OSErr DoCreateNewAttribute(DETCallBlockPtr callBlockPtr)
{
    OSErr err = 1;
    RStringPtr *name, *zone, *dName;
    /* Initialize the data. */
    name = nil;
    zone = nil;
    dName = nil;
```

```
    /* Get the name. */
    name = (RStringPtr*) GetResource('rstr', kSurfInfoPageAspect +
                                    prDefaultName);


    /* Get the zone. */
    zone = (RStringPtr*) GetResource('rstr', kSurfInfoPageAspect +
                                    prDefaultZone);


    /* Get the display name. */
    dName = (RStringPtr*) GetResource('rstr', kSurfInfoPageAspect +
                                      prDefaultDisplayName);


    /* Lock everything. */
    HLock((Handle) name);
    HLock((Handle) zone);
    HLock((Handle) dName);


    /* Write out the data. */
    err = DoWriteNameAndZone(callBlockPtr, *name, *dName, *zone,
                             callBlockPtr->attributeCreationBlock.value);


    /* Unlock everything. */
    HUnlock((Handle) name);
    HUnlock((Handle) zone);
    HUnlock((Handle) dName);


    /* Clean up. */
    ReleaseResource((Handle) name);
    ReleaseResource((Handle) zone);
    ReleaseResource((Handle) dName);


    if (err == noErr)
        err = kDETDidNotHandle;


    return err;
}


/* ------------------------------------------------------------------------- */
/* The Catalogs Extension calls this routine when you call the
   kDETcmdDirtyProperty callback routine to indicate that a property value
   has changed, requiring a view to be redrawn. The CE also calls this
   routine when the CE completes its first catalog lookup. If the user has
   changed either of the text fields in the "Fields" view of the info page,
```

this routine updates the value of the address string. If the user has
changed the address in the "String" view of the info page, this routine
updates the values of the address fields. The DoUpdateNameAndZone and
DoUpdateAddress functions are shown in
"Data Output Subroutines for the Address Template" beginning on
page 4-51. */

```
static OSErr DoPropertyDirty(DETCallBlockPtr callBlockPtr)
{
   OSErr err = noErr;
   short dirtyProperty;
   if (!DoIsInited(callBlockPtr))
      return 1;

   dirtyProperty = callBlockPtr->propertyDirtied.property;

   if (dirtyProperty == prMyAddress)
   {
      err = DoUpdateNameAndZone(callBlockPtr);
   }
   else if ((dirtyProperty == prMyName) || (dirtyProperty == prMyZone))
   {
      err = DoUpdateAddress(callBlockPtr);
   }

   if (err == noErr)
   {
      err = 1;
   }

   DoHandleError(callBlockPtr, err);

   return err;
}


/* ------------------------------------------------------------------------ */
/* The CE calls this routine when processing the custom lookup-table
    element that processes output data. This routine is discussed in
    "Data Output Subroutines for the Address Template" on page 4-51.*/
```

```
static OSErr DoPatternOut(DETCallBlockPtr callBlockPtr)
{
    OSErr err = 1;
    long message;
    message = callBlockPtr->patternIn.elementType;

    if (!DoIsInited(callBlockPtr))
        return 1;

    /* Write data only if you're supposed to be writing */
    if (message == kProcessData)
    {
        err = DoWriteData(callBlockPtr);
    }

    return err;
}
```

Data Input Subroutines for the Address Template

The CE calls your code resource's DETcmdPatternIn routine when it has to process a custom lookup-table element for input data. The DoPatternIn function shown in Listing 4-6 calls the DoExtractInformation function shown in Listing 4-7. The DoExtractInformation function calls the DoReadData function, which in turn calls the MyCreateFieldsFromOCERecipient function to read the address fields from the OCERecipient structure. Then the DoReadData function calls the MyCreateAddressString function, which creates an address string from the address fields. Finally, the DoReadData function calls the DoSetAllStringProperties function, which sets the values of the field and string properties for display in the information page.

Next, the DoExtractInformation function calls the DoExtractDisplayName function, passing it the attribute data; that is, the address in the form of an OCEPackedRecipient structure. The DoExtractDisplayName function unpacks the OCEPackedRecipient structure and extracts the record name. It sets the name of the attribute value to the record name. The Electronic Addresses information page uses this name for display in the sublist.

Finally, the DoExtractInformation function calculates the size of the data it just read and sets the data offset and bit offset fields to the end of the attribute data so that the CE stops processing the attribute.

The DoPatternIn function (page 4-44) also calls the DoSetDisplayName function (page 4-51). The DoSetDisplayName function checks whether the attribute value has been assigned a display name (that is, a name to display for the attribute value in the sublist or for a stand-alone attribute). If not, it sets the display name to be the same as the string in the property prMyName. That property is also used for the Name field in the Fields address information page (see Figure 4-2 on page 4-31).

Listing 4-7   Input subroutines for the address template code resource

```
static OSErr DoExtractInformation(DETCallBlockPtr callBlockPtr)
{

    OSErr err = 1;
    PackedDSSpecPtr pds;
    Boolean enabled = true;
    short size;
    pds = (PackedDSSpecPtr) callBlockPtr->patternIn.attribute->value.bytes;
    if (DoGetXtnType(pds) == 'WAVE')
    {
        /* Read the data in */
        err = DoReadData(callBlockPtr, pds);

        err = DoExtractDisplayName(callBlockPtr, pds);

        size = (* ((short *) callBlockPtr->patternIn.attribute->value.bytes))
                + 2;
        callBlockPtr->patternIn.dataOffset = size;
        callBlockPtr->patternIn.bitOffset = 0;
    }
    else
    {
        /* This isn't a SurfWriter address.  Abort! */
        enabled = false;
    }

    if ((err == noErr) && enabled)
    {
        DoSetInited(callBlockPtr, true);
    }

    return err;
}

/* ----------------------------------------------------------------------- */

static long DoGetXtnType(PackedDSSpecPtr pds)
{
    long extensionType;
    DSSpec spec;
    RecordID rid;
```

```
   OCEUnpackDSSpec(pds, &spec, &rid);
   extensionType = spec.extensionType;

   return extensionType;
}
/* ------------------------------------------------------------------------ */

static OSErr DoReadData(DETCallBlockPtr callBlockPtr, PackedDSSpecPtr pds)
{
   OSErr err = noErr;
   RStringPtr name, zone, address;
   /* Initialize data */
   name = nil;
   zone = nil;
   address = nil;

   /* Extract the info we want */
   err = MyCreateFieldsFromOCERecipient(pds, &name, &zone);

   if (err == noErr)
   {
      /* Create the address string for display */
      err = MyCreateAddressString(name, zone, &address);
   }

   if (err == noErr)
   {
      err = DoSetAllStringProperties(callBlockPtr, name, zone, address);
   }

   DisposeIfPtr(name);
   DisposeIfPtr(zone);
   DisposeIfPtr(address);

   return err;
}
/* ------------------------------------------------------------------------ */

static OSErr DoSetAllStringProperties(DETCallBlockPtr callBlockPtr,
                                      RStringPtr name,
                                      RStringPtr zone,
                                      RStringPtr address)
```

```
{
    OSErr err = noErr;
    /* Set the zone property. */
    err = DoSetRStringProperty(callBlockPtr, prMyZone, zone, false);

    /* Set the name property for the list view. */
    err = DoSetRStringProperty(callBlockPtr, prMyName, name, false);

    /* Set the OldName property for the list view. */
    err = DoSetRStringProperty(callBlockPtr, prOldName, name, false);

    /* Create the address for display. */
    err = DoSetRStringProperty(callBlockPtr, prMyAddress, address, false);

    /* Create the OldAddress for display. */
    err = DoSetRStringProperty(callBlockPtr, prOldAddress, address, false);
    return err;
}
/* ------------------------------------------------------------------------- */

static OSErr DoExtractDisplayName(DETCallBlockPtr callBlockPtr,
PackedDSSpecPtr pds)
{
    OSErr err = noErr;
    DSSpec spec;
    RecordID rid;
    RStringPtr dName;

    OCEUnpackDSSpec(pds, &spec, &rid);
    dName = rid.local.recordName;

    if (dName != nil)
    {
        err = DoSetRStringProperty(callBlockPtr, kDETAspectName, dName, false);
        if (err == kDETPropertyBusy)
        {
            err = noErr;
        }
    }

    return err;
}
```

```
/* ---------------------------------------------------------------------- */

static OSErr DoSetDisplayName(DETCallBlockPtr callBlockPtr)
{
    OSErr err = noErr;
    RStringPtr name, dName;

    name = nil;
    dName = nil;

    /* Get the name. */
    err = DoGetRStringPtrProperty(callBlockPtr, prMyName, &name);

    /* Get the display name. */
    err = DoGetRStringPtrProperty(callBlockPtr, kDETAspectName, &dName);

    if (dName->dataLength <= 0)
    {
        err = DoSetRStringProperty(callBlockPtr, kDETAspectName, name, false);
    }

    DisposeIfPtr(name);
    DisposeIfPtr(dName);

    return err;
}
```

Data Output Subroutines for the Address Template

The CE calls your code resource's `DETcmdPatternOut` routine when it has to process a custom lookup-table element for output data. The parameter block that the CE provides with the `DETcmdPatternOut` call to your code resource includes a handle to the attribute. The attribute already has an attribute tag assigned but lacks the data length and data fields. The `DoPatternOut` function shown in Listing 4-6 calls the `DoWriteData` function shown in Listing 4-8, reads the address fields from the current property values, and calls the `DoWriteNameAndZone` function. The `DoWriteNameAndZone` function calls the `DoPackNameAndZone` function, which verifies the name and zone and calls the `MyCreateOCERecipient` function. The `MyCreateOCERecipient` function packs the address fields into a string and creates an `OCEPackedRecipient` structure that includes the address string as an extension. Then the `DoWriteNameAndZone` function appends the `OCEPackedRecipient` structure to the data handle provided by the CE, thus updating the attribute value. The `MyCreateOCERecipient` function is not shown here. For a sample function that creates an `OCEPackedRecipient` structure for an external messaging system, see "Translating to an AOCE Address" beginning on page 2-88 in the chapter "Messaging Service Access Modules" in this book.

The Catalogs Extension calls your kDETcmdPropertyDirtied routine when a property value has changed, requiring a view to be redrawn. If the user has changed either of the text fields in the "Fields" view of the info page, the DoPropertyDirty function shown in Listing 4-6 calls the DoUpdateAddress function to update the value of the address string. If the user has changed the address in the "String" view of the information page, the DoPropertyDirty function calls the DoUpdateNameAndZone function to update the values of the address fields.

Listing 4-8   Output subroutines for the address template code resource

```
static OSErr DoWriteData(DETCallBlockPtr callBlockPtr)
{
   OSErr err = noErr;
   RStringPtr name, zone, dName;
   Size size;
   /* Initialize the data. */
   name = nil;
   zone = nil;
   dName = nil;

   /* Get the name. */
   err = DoGetRStringPtrProperty(callBlockPtr, prMyName, &name);

   /* Get the zone. */
   if (err == noErr)
   {
      err = DoGetRStringPtrProperty(callBlockPtr, prMyZone, &zone);
   }

   /* Get the display name. */
   if (err == noErr)
   {
      err = DoGetRStringPtrProperty(callBlockPtr, kDETAspectName, &dName);
   }

   /* Write out the data. */
   if (err == noErr)
   {
      err = DoWriteNameAndZone(callBlockPtr, name, dName, zone,
                          callBlockPtr->patternOut.data);
   }
```

```
   if (err == noErr)
   {
      size = GetHandleSize((Handle) callBlockPtr->patternOut.data);
      callBlockPtr->patternOut.dataOffset = size;
      callBlockPtr->patternOut.bitOffset = 0;
   }

   DisposeIfPtr(name);
   DisposeIfPtr(zone);
   DisposeIfPtr(dName);

   return err;
}


/* ------------------------------------------------------------------------- */

static OSErr DoWriteNameAndZone(DETCallBlockPtr callBlockPtr, RStringPtr
name, RStringPtr dName, RStringPtr zone, Handle buffer)
{
   OSErr err = noErr;
   PackedDSSpecPtr pds = nil;
   Size size;

   /* Pack the data. */
   err = DoPackNameAndZone(callBlockPtr, name, dName, zone, &pds);

   /* Append the address to the data handle provided by the CE. */
   if (err == noErr)
   {
      size = GetPtrSize((Ptr) pds);

      SetHandleSize(buffer, size);
      err = MemError();

      if (err == noErr)
         BlockMove((Ptr) pds, *buffer, size);
   }

   /* Dispose of allocated stuff. */
   DisposeIfPtr(pds);

   return err;
}
```

```
/* ------------------------------------------------------------------------ */
static OSErr DoPackNameAndZone(DETCallBlockPtr callBlockPtr, RStringPtr name,
RStringPtr dName, RStringPtr zone, PackedDSSpecPtr* pds)
{
    OSErr err = noErr;
    unsigned short size;
    /* Initialize the data. */
    *pds = nil;

    /* Validate the name and zone. */
    if (!DoStringPtrIsOK(name) || !DoStringPtrIsOK(zone))
       err = paramErr;

    /* Create an address. */
    if (err == noErr)
    {
       err = MyCreateOCERecipient(pds, &size, name, dName, zone);
    }

    if ((*pds == nil) && (err == noErr))
       err = paramErr;

    return err;
}
/* ------------------------------------------------------------------------ */
/* This routine is called to make sure that the name and zone are
    not of zero length and that they have valid values. */

static Boolean DoStringPtrIsOK(RStringPtr string)
{
    Boolean good = true;
    Str255 divider;
    if (string->dataLength < 0)
       good = false;

    if (string->dataLength > 0)
    {
       if (MyValidateString(string, divider, 0) != kBadString)
          good = false;
    }

    return good;
}
```

```
/* ------------------------------------------------------------------------ */

static OSErr DoUpdateNameAndZone(DETCallBlockPtr callBlockPtr)
{
   OSErr err = noErr;
   OSErr tempErr;
   RStringPtr name, zone, address, oldAddress;
   /* Initialize the variables. */
   name = nil;
   zone = nil;
   address = nil;
   oldAddress = nil;

   err = DoGetRStringPtrProperty(callBlockPtr, prMyAddress, &address);

   if (err == noErr)
   {
      err = MyDecomposeServerlessAddressString(&name, &zone, address);
   }

   if ((err == noErr) && (name->dataLength == 0))
      err = kNullNameError;
   else if ((err == kSMPInvalidAddressString) && (name == nil))
      err = kNullNameError;

   if (err == noErr)
   {
      err = DoSetRStringProperty(callBlockPtr, prMyName, name, true);
      err = DoSetRStringProperty(callBlockPtr, prMyZone, zone, true);
      err = DoSetRStringProperty(callBlockPtr, prOldName, name, true);
   }
   else
   {
      tempErr = DoGetRStringPtrProperty(callBlockPtr, prOldAddress,
                                        &oldAddress);

      tempErr = DoSetRStringProperty(callBlockPtr, prMyAddress, oldAddress,
                                     true);
   }

   DisposeIfPtr(name);
   DisposeIfPtr(zone);
   DisposeIfPtr(oldAddress);
```

```
   DisposeIfPtr(address);

   return err;
}


/* ------------------------------------------------------------------------ */

static OSErr DoUpdateAddress(DETCallBlockPtr callBlockPtr)
{
   OSErr err = noErr;
   OSErr tempErr;
   RStringPtr name, zone, address;
   RStringPtr oldName, oldZone;
   short dirtyProperty;
   /* Initialize the variables. */
   name = nil;
   zone = nil;
   address = nil;
   oldName = nil;
   oldZone = nil;

   dirtyProperty = callBlockPtr->propertyDirtied.property;

   err = DoGetRStringPtrProperty(callBlockPtr, prMyName, &name);
   err = DoGetRStringPtrProperty(callBlockPtr, prMyZone, &zone);

   if (err == noErr)
   {
      if ((name->dataLength == 0) && (dirtyProperty == prMyName))
         err = kNullNameError;
   }

   if (err == noErr)
   {
      err = MyCreateAddressString(name, zone, &address);
   }

   if (err == noErr)
   {
      err = DoSetRStringProperty(callBlockPtr, prMyAddress, address, true);
      err = DoSetRStringProperty(callBlockPtr, prOldAddress, address, true);
      err = DoSetRStringProperty(callBlockPtr, prOldName, name, true);
   }
```

```
    else
    {
        /* Revert to the previous version. */

        tempErr = DoGetRStringPtrProperty(callBlockPtr, prOldName, &oldName);
        if (tempErr == noErr)
        {
        tempErr = DoSetRStringProperty(callBlockPtr, prMyName, oldName, true);
        }
    }

    DisposeIfPtr(oldName);
    DisposeIfPtr(oldZone);
    DisposeIfPtr(address);
    DisposeIfPtr(name);
    DisposeIfPtr(zone);

    return err;
}
```

Miscellaneous Subroutines

**The subroutines in Listing 4-9 are called by one or more of the functions in the
preceding sections.**

Listing 4-9   Miscellaneous subroutines used by the address template code resource

```
/* Set value of property prInited. */
static void DoSetInited(DETCallBlockPtr callBlockPtr, Boolean inited)
{
    OSErr err = noErr;
    err = DoSetBooleanProperty(callBlockPtr, prInited, inited, false);
}

/* ------------------------------------------------------------------------ */
/* Get value of property prInited. */
static Boolean DoIsInited(DETCallBlockPtr callBlockPtr)
{
    OSErr err = noErr;
    Boolean inited = false;
    err = DoGetBooleanProperty(callBlockPtr, prInited, &inited);
```

```
   if (err != noErr)
      inited = false;

   return inited;
}
/* ------------------------------------------------------------------------ */
/* Error handler */
static void DoHandleError(DETCallBlockPtr callBlockPtr, OSErr err)
{
   if ((err == noErr) || (err == kDETDidNotHandle))
      return;
/* Call kDETcmdAboutToTalk callback routine. */
   AboutToTalk(callBlockPtr);

/* Display one of the error messages in the template. */
   switch (err)
   {
      case kNullNameError:
         DisplayErrorMessage(err, kSurfInfoPageAspect, 2);
         break;

      default:
         DisplayErrorMessage(err, kSurfInfoPageAspect, 1);
         break;
   }
}

/* ------------------------------------------------------------------------ */
/* Call kDETcmdSetPropertyRString callback routine. */
pascal OSErr DoSetRStringProperty(DETCallBlockPtr callBlockPtr,
                     short property,
                     RStringPtr newValue,
                     Boolean markAsChanged)
   {
   OSErr err;
   DETCallBackBlock cbb;

   cbb.setPropertyRString.reqFunction = kDETcmdSetPropertyRString;
   cbb.setPropertyRString.property = property;
   cbb.setPropertyRString.target.selector = kDETSelf;
   cbb.setPropertyRString.newValue = newValue;

   err = CallBackDET(callBlockPtr, &cbb);
```

```
    if ((err == noErr) && markAsChanged)
        {
        err = DoSetPropertyChanged(callBlockPtr, property, true);
        }

    return err;
    }

/* ------------------------------------------------------------------------- */
/* Call kDETcmdSetPropertyNumber callback routine. */
pascal OSErr DoSetNumProperty(DETCallBlockPtr callBlockPtr,
                              short property,
                              unsigned long newValue,
                              Boolean markAsChanged)
    {
    OSErr err;
    DETCallBackBlock cbb;

    cbb.setPropertyRString.reqFunction = kDETcmdSetPropertyNumber;
    cbb.setPropertyRString.property = property;
    cbb.setPropertyRString.target.selector = kDETSelf;
    cbb.setPropertyRString.newValue = newValue;

    err = CallBackDET(callBlockPtr, &cbb);

    if ((err == noErr) && markAsChanged)
        {
        err = DoSetPropertyChanged(callBlockPtr, property, true);
        }

    return err;
    }

/* ------------------------------------------------------------------------- */
/* Call kDETcmdSetPropertyNumber callback routine with a value of 0 or 1. */
pascal OSErr DoSetBooleanProperty(DETCallBlockPtr callBlockPtr,
                        short property,
                        Boolean value,
                        Boolean markChanged)
{
    OSErr err = noErr;

    err = DoSetNumProperty(callBlockPtr, property, value, markChanged);
```

```
      return err;
}

/* ----------------------------------------------------------------------- */
/* Call kDETcmdSetPropertyChanged callback routine. */
pascal OSErr DoSetPropertyChanged(DETCallBlockPtr callBlockPtr,
                        short property,
                        Boolean propertyChanged)
{
   OSErr err;
   DETCallBackBlock cbb;

   cbb.SetPropertyChanged.reqFunction = kDETcmdSetPropertyChanged;
   cbb.SetPropertyChanged.property = property;
   cbb.SetPropertyChanged.target.selector = kDETSelf;
   cbb.SetPropertyChanged.propertyChanged = propertyChanged;

   err = CallBackDET(callBlockPtr, &cbb);
   return err;
}

/* ----------------------------------------------------------------------- */
/* Call kDETcmdGetPropertyRString callback routine. */
pascal OSErr DoGetRStringProperty(DETCallBlockPtr callBlockPtr,
                           short property,
                           RString ***str)
{
   OSErr err;
   DETCallBackBlock cbb;

   cbb.getPropertyRString.reqFunction = kDETcmdGetPropertyRString;
   cbb.getPropertyRString.property = property;
   cbb.getPropertyRString.target.selector = kDETSelf;

   err = CallBackDET(callBlockPtr, &cbb);
   *str = cbb.getPropertyRString.propertyValue;

   return err;
```

```
/* ------------------------------------------------------------------------- */
/* Call kDETcmdGetPropertyNumber callback routine. */
pascal OSErr DoGetNumProperty(DETCallBlockPtr callBlockPtr,
                              short property,
                              long *value)
{
   OSErr err;
   DETCallBackBlock cbb;

   cbb.getPropertyNumber.reqFunction = kDETcmdGetPropertyNumber;
   cbb.getPropertyNumber.property = property;
   cbb.getPropertyNumber.target.selector = kDETSelf;

   err = CallBackDET(callBlockPtr, &cbb);

   *value = cbb.getPropertyNumber.propertyValue;

   return err;
}


/* ------------------------------------------------------------------------- */
/* Call kDETcmdGetPropertyNumber callback routine and return 0 or 1. */
pascal OSErr DoGetBooleanProperty(DETCallBlockPtr callBlockPtr,
                        short property,
                        Boolean* value)
{
   OSErr err = noErr;
   long number;

   err = DoGetNumProperty(callBlockPtr, property, &number);
   *value = (number == 1);

   return err;
}


/* ------------------------------------------------------------------------- */
/* Call kDETcmdGetPropertyRString callback routine and convert handle to
   pointer. */
pascal OSErr DoGetRStringPtrProperty(DETCallBlockPtr callBlockPtr,
                                short property,
                                RStringPtr* str)
{
   OSErr err = noErr;
```

```
    RStringHandle stringH = nil;
    RStringPtr stringP = nil;

    err = DoGetRStringProperty(callBlockPtr, property, &stringH);

    if (err == noErr)
    {
        err = DoRStringHandleToPtr(stringH, &stringP);
    }

    DisposeHandle((Handle) stringH);
    *str = stringP;

    return err;
}

/* ------------------------------------------------------------------------ */
/* Convert an RString handle to a pointer. */
pascal OSErr DoRStringHandleToPtr(RStringHandle stringH,
                        RStringPtr* string)
{
    OSErr err = noErr;
    RStringPtr stringP = nil;

    stringP = (RStringPtr) NewPtr(sizeof(ProtoRString) +
                            (*stringH)->dataLength);

    HLock((Handle) stringH);
    err = OCECopyRString(*stringH, stringP, (*stringH)->dataLength);
    HUnlock((Handle) stringH);

    *string = stringP;
    return err;
}
```

SAM Setup Reference

This section lists and describes the contents of the PowerTalk Setup catalog and the attributes in records of the following types:

n   Setup

n   MSAM

n   CSAM

n   Mail Service

n   Catalog

n   Combined

Following the descriptions of these records and attributes, this section describes all of the properties and resources that you must include in your setup template.

The PowerTalk Setup Catalog

The information in the PowerTalk Setup catalog completely describes the AOCE services available on the Macintosh computer on which the Setup catalog is located. The records in the Setup catalog contain information about the installed CSAMs, the catalogs associated with those CSAMs, the installed personal MSAMs, and the messaging services associated with those MSAMs. Each CSAM and personal MSAM is represented by a record, and each personal MSAM mail slot and CSAM catalog is represented by a record (a single Combined record can represent both a mail slot and a catalog). In addition, there is a special Setup record that ties everything together.

When a user installs the PowerTalk software on his or her Macintosh, PowerTalk software creates the Setup catalog. The Setup catalog initially contains a single record called the **Setup record.** As the user adds catalog or messaging services, additional records are needed in the Setup catalog to specify these services.

Many of the records in the Setup catalog refer to other records in the Setup catalog by means of a **record reference.** A record reference is an attribute whose value consists of a packed record ID, of which only the creation ID is used to identify a given record. Except for parent MSAM record attributes, which your setup template creates, the PowerTalk Key Chain manages record references; you shouldn't need to examine or manipulate them.

Setup templates create or modify many of the records in the Setup catalog. The sections that follow describe the types of records in the Setup catalog and the contents of those records. The sections "Adding Catalog and Mail Services" beginning on page 4-5 and "Modifying an Existing Service" on page 4-30 explain how you actually create and modify the records.

Table 4-1 lists the standard types of records contained in the Setup catalog, provides the corresponding record type index where applicable, and notes who creates a record of that type. Record type indexes are described in the chapter "AOCE Utilities" in the book *Inside Macintosh: AOCE Application Interfaces.*

Table 4-1 Setup-catalog record types

Type of record	Record type index	Created by
Setup	`kSetupRecTypeNum`	AOCE
MSAM	`kMSAMRecTypeNum`	Setup template (by calling the `DirAddRecord` function if this record does not already exist)
CSAM	`kDSAMRecTypeNum`	Setup template (by calling the `DirAddDSAM` function)
Mail Service (also known as a *slot record*)	N/A	Key Chain; main aspect template provided by MSAM-only setup templates
Catalog	N/A	Key Chain; main aspect template provided by MSAM-only and CSAM-only setup templates
Combined	N/A	Key Chain; main aspect template provided by combined MSAM and CSAM setup templates

The Setup Record

There is a single Setup record in the Setup catalog. It contains record references to all of the records in the PowerTalk Setup catalog that represent slots, catalogs, and other items that show up in the PowerTalk Key Chain. It is identified by the record type index constant `kSetupRecTypeNum`. The Key Chain sets up and maintains the Setup record; you do not manipulate it or read it.

The MSAM Record

An MSAM record represents a personal MSAM. The setup template for a given MSAM creates this record. The record name is the same as the name of the file containing the personal MSAM at the time you create the record. (Once the MSAM has been created and configured, the user can change the filename without affecting the MSAM record). An MSAM record is identified by the record type index constant `kMSAMRecTypeNum`. Table 4-2 shows the attributes for an MSAM record.

Table 4-2   Attributes of an MSAM record

Attribute type and index	Data type	Description	Written by
AOCE version `kVersionAttrTypeNum`			
	`long`	AOCE version number.	Setup template
Gateway file ID `kGatewayFileIDAttrTypeNum`			
	`long`	The file ID of the personal MSAM file. To activate a mail slot, you need to find the slot's Mail Service record. To do so, you compare your file's file ID with the file ID stored in the gateway file ID attribute in each MSAM record. (If no Mail Service record exists for this file, you must create one.)	Setup template
Mail service `kMailServiceAttrTypeNum`			
	`PackedRecordID`	A record reference to a Mail Service record that represents a slot belonging to this personal MSAM. An MSAM record contains one mail service attribute for each slot associated with the MSAM.	Setup template

An MSAM record may also contain MSAM-specific information in attributes added by the personal MSAM, its setup template, or both.

The CSAM Record

A CSAM record represents a CSAM. A setup template creates this record by calling the `DirAddDSAM` function. The record name is the same as the name of the file that contains the CSAM. A CSAM record is identified by the record type index constant `kDSAMRecTypeNum`. Table 4-3 shows the attributes for a CSAM record.

Table 4-3   Attributes of a CSAM record

Attribute type and index	Data type	Description	Written by
CSAM alias kDSAMFileAliasAttrTypeNum			
	Private to AOCE	An alias to the CSAM file, created and used by AOCE software.	Setup template calls DirAddDSAM function
Catalog kDirectoryAttrTypeNum			
	PackedRecordID	A record reference to a Catalog record that represents a catalog available through this CSAM. A CSAM record contains one catalog attribute for each catalog associated with the CSAM.	Setup template calls DirAddDSAMDirectory function

A CSAM record may also contain CSAM-specific information added by the CSAM, its setup template, or both.

The Mail Service Record

A Mail Service record (also known as a *slot record*) contains information about a mail slot. The Key Chain creates this record, using a main aspect template provided by your setup template, when your setup template adds a mail service only. When your setup template adds a combined mail and catalog service, the mail slot information is contained in a Combined record rather than a Mail Service record.

Your main aspect template for the Mail Service record must specify the record type "aoce Mail Service*xxxx*" where *xxxx* is the address extension type of the messaging system to which your MSAM provides access. You can use the constant kMailServiceRecTypeBody to do so; for example, the following fragment of an aspect template creates a record of type "aoce Mail ServiceWAVE":

```
#define kServiceRecordType kMailServiceRecTypeBody "WAVE"

resource 'deta' (kSurfWriterAspect, purgeable)
{
   0, dropCheckConflicts, isMainAspect
};

resource 'rstr' (kSurfWriterAspect + kDETRecordType, purgeable)
{
   kServiceRecordType
};
```

Table 4-4 shows the attributes for a Mail Service record.

Table 4-4   Attributes of a Mail Service record

Attribute type and index	Data type	Description	Written by
AOCE version kVersionAttrTypeNum			
	long	AOCE version number.	Key Chain
Associated catalog kAssoDirectoryAttrTypeNum			
	PackedRecordID	A record reference to the record that represents the catalog with which this slot is associated. AOCE needs the Catalog record to route messages; the Setup catalog must contain the Catalog record before your MSAM can be properly set up.	Key Chain
Parent MSAM kParentMSAMAttrTypeNum			
	PackedRecordID	A record reference to the record that represents the MSAM to which this slot belongs.	Setup template
Slot ID kSlotIDAttrTypeNum			
	SlotID	The slot ID for this mail slot.	MSAM
Standard slot information kStdSlotInfoAttrTypeNum			
	MailStandardSlotInfoAttribute		
		A structure that contains information about the slot, such as when to log on to the external messaging system.	Setup template

The Mail Service record may also contain other slot-specific information added by the personal MSAM, its setup template, or both.

The Catalog Record

A Catalog record represents a catalog to which the user has access. The Key Chain creates this record, using a main aspect template provided by your setup template, when your setup template adds a mail service only or a catalog service only. When your setup template adds a combined mail and catalog service, the catalog information is contained in a Combined record rather than a Mail Service record, as described in the next section.

Your main aspect template for the Catalog record must specify the record type "aoce Directory*xxxx*". If the catalog is associated with a mail slot, *xxxx* is the address extension type of the external messaging system to which the slot's MSAM provides access. If the catalog is not associated with a slot, *xxxx* is the `signature` field of the catalog discriminator (`DirDiscriminator` structure).

You can use the constant `kDirectoryRecTypeBody` to assign the record type; for example, the following fragment of an aspect template creates a record of type "aoce DirectoryWAVE":

```
#define kServiceRecordType kDirectoryRecTypeBody "WAVE"

resource 'deta' (kSurfWriterAspect, purgeable)
{
    0, dropCheckConflicts, isMainAspect
};

resource 'rstr' (kSurfWriterAspect + kDETRecordType, purgeable)
{
    kServiceRecordType
};
```

Table 4-5 shows the attributes for a Catalog record. Note that some attributes are used in Catalog records only for stand-alone MSAMs, some are used in Catalog records only for stand-alone CSAMs, and some are used both for MSAMs and CSAMs. When you install a combined MSAM and CSAM, you provide a template for a Combined record rather than for a Catalog record (see Table 4-6 on page 4-70).

Table 4-5   Attributes of a Catalog record

Attribute type and index	Data type	Description	Written by	Used for
Version `kVersionAttrTypeNum`				
	`long`	Version number of the Key Chain at the time the record was created	Key Chain	CSAM
Associated mail service `kAssoMailServiceAttrTypeNum`				
	`PackedRecordID`	Record reference to the Mail Service record associated with this catalog	Key Chain	MSAM
Parent CSAM `kParentDSAMAttrTypeNum`				
	`PackedRecordID`	Record reference to CSAM record for this catalog	Setup template calls `DirAddDSAMDirectory` function	
				CSAM

continued

Table 4-5   Attributes of a Catalog record (continued)

Attribute type and index	Data type	Description	Written by	Used for
Discriminator `kDiscriminatorAttrTypeNum`				
	`DirDiscriminator`	This catalog's discriminator value	Setup template (CSAM template calls `DirAddDSAMDirectory` function)	
				MSAM CSAM
Capability flags `kSFlagsAttrTypeNum`				
	`long`	This catalog's capability flags	Setup template calls `DirAddDSAMDirectory` function	
				CSAM
Comment `kCommentAttrTypeNum`				
	`RString`	Comment for your use	Setup template	MSAM CSAM
Real name `kRealNameAttrTypeNum`				
	`RString`	Name of this catalog for your use	Setup template	MSAM CSAM
User's record ID `kDirUserRIDAttrTypeNum`				
	`RecordID`	User's record ID	Setup template calls `OCESetupAddDirectoryInfo` function	
				MSAM CSAM
Native name `kDirNativeNameAttrTypeNum`				
	`RString`	User's name or account name in the external catalog; for your use	Setup template	MSAM CSAM
User's key `kDirUserKeyAttrTypeNum`				
	Private to AOCE	User's encrypted password	Setup template calls `OCESetupAddDirectoryInfo` function	
				MSAM

continued

Table 4-5   Attributes of a Catalog record (continued)

Attribute type and index	Data type	Description	Written by	Used for
Private data `kPrivateDataAttrTypeNum`				
	Binary data of any length (to maximum size of attribute)	Data for your use; for example, information about address formats	Setup template	MSAM CSAM

The Combined Record

A Combined record represents a mail slot and a catalog in a single record. The Key Chain creates this record, using a main aspect template provided by your setup template, when your setup template adds a combined MSAM and CSAM.

Your main aspect template for the Combined record must specify the record type "aoce Combined*xxxx*" where *xxxx* is the address extension type of the external messaging system to which the slot's MSAM provides access.

You can use the constant `kCombinedRecTypeBody` to assign the record type; for example, the following fragment of an aspect template creates a record of type "aoce CombinedWAVE":

```
#define kServiceRecordType kCombinedRecTypeBody "WAVE"


resource 'deta' (kSurfWriterAspect, purgeable)
{
   0, dropCheckConflicts, isMainAspect
};


resource 'rstr' (kSurfWriterAspect + kDETRecordType, purgeable)
{
   kServiceRecordType
};
```

Table 4-6 shows the attributes for a Combined record.

Table 4-6   Attributes of a Combined record

Attribute type and index	Data type	Description	Written by
Version `kVersionAttrTypeNum`			
	`long`	Version number of the Key Chain at the time the record was created.	Key Chain

Table 4-6   Attributes of a Combined record (continued)

Attribute type and index	Data type	Description	Written by
Associated catalog `kAssoDirectoryAttrTypeNum`			
	`PackedRecordID`	A record reference to the record that represents the catalog with which this slot is associated. For Combined records, this attribute points back to the Combined record itself.	
			Key Chain
Associated mail service `kAssoMailServiceAttrTypeNum`			
	`PackedRecordID`	Record reference to the Mail Service record associated with this catalog. For Combined records, this attribute points back to the Combined record itself.	
			Key Chain
Parent CSAM `kParentDSAMAttrTypeNum`			
	`PackedRecordID`	Record reference to the CSAM record for this catalog.	Setup template calls `DirAddDSAMDirectory` function
Parent MSAM `kParentMSAMAttrTypeNum`			
	`PackedRecordID`	A record reference to the record that represents the MSAM to which this slot belongs.	Setup template
Slot ID `kSlotIDAttrTypeNum`			
	`SlotID`	The slot ID for this mail slot.	MSAM
Standard slot information `kStdSlotInfoAttrTypeNum`			
	`MailStandardSlotInfoAttribute`	A structure that contains information about the slot, such as when to log on to the external messaging system.	Setup template

Table 4-6   Attributes of a Combined record (continued)

Attribute type and index	Data type	Description	Written by
Discriminator `kDiscriminatorAttrTypeNum`			
	`DirDiscriminator`	This catalog's discriminator value.	Setup template calls `DirAddDSAMDirectory` function
Capability flags `kSFlagsAttrTypeNum`			
	`long`	This catalog's capability flags.	Setup template calls `DirAddDSAMDirectory` function
Comment `kCommentAttrTypeNum`			
	`RString`	Displayable comment about this catalog; for example, the time the catalog was installed.	Setup template
Real name `kRealNameAttrTypeNum`			
	`RString`	Name of this catalog for your use.	Setup template
User's record ID `kDirUserRIDAttrTypeNum`			
	`RecordID`	User's record ID.	Setup template calls `OCESetupAddDirectoryInfo` function
Native name `kDirNativeNameAttrTypeNum`			
	`RString`	User's name or account name in the external catalog; for your use.	Setup template
User's key `kDirUserKeyAttrTypeNum`			
	Private to AOCE	User's encrypted password.	Setup template calls `OCESetupAddDirectoryInfo` function
Private data `kPrivateDataAttrTypeNum`			
	Binary data of any length (to maximum size of attribute)	Data for your use; for example, information about address formats.	Setup template

The Setup Template Resources

Every CSAM and personal MSAM must include a setup template in the resource fork of the SAM file. The setup template provides the human interface that allows a user to add or remove the SAM and the services that it supports. In response to user input, the template creates and modifies records in the Setup catalog.

A setup template consists of an aspect template and at least one information page template. To learn how to write an AOCE template, read the chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces*. This section covers only those topics that are specific to setup templates.

Table 4-7 shows the resources required for the setup aspect template. Only those resources that are unique to setup templates or that must have specific values in setup templates are described here. For descriptions of the other required resources and for a complete list of all the resources you can use in aspect templates, see the chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces*.

Table 4-7   Required resources for setup aspect templates

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
`'deta'`	0	Identifies template as main aspect and provides a base resource ID.
`'rstr'`	kDETTemplateName	Name of template.
`'rstr'`	kDETRecordType	Type of record to which the template applies.
`'rstr'`	kDETAspectName	The name displayed in the Key Chain in the Service field.
`'rstr'`	kSAMAspectKind	The kind of service as shown in the Kind field of the Key Chain.
`'detn'`	kSAMAspectCannotDelete	A property that detemines whether the user can delete the slot or catalog set up by this aspect.
`'rstr'`	kSAMAspectUserName	The string the Key Chain displays in the Name field.
`'sami'`	kSAMAspectSlotCreationInfo	The information needed by the Key Chain to create and delete slots and catalogs.
Icon suite	kDETAspectMainBitmap	Suite of icons.
`'rstr'`	kDETAspectKind	The kind of record as shown in the Get Info dialog box. Neither the code resource nor the user can change this value.
`'rstr'`	kDETAspectWhatIs	Help-balloon string for objects of the type described by this aspect when they appear in the Key Chain.
`'detc'`	kDETAspectCode	A code resource.

IMPORTANT

Because of these additional required resources, your own properties should start at offset `kSAMFirstDevProperty` rather than at `kFirstDevProperty`. s

The rest of this section describes the resources required for setup aspect templates. For a complete description of the resources you can use in any AOCE aspect template, see the chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces*.

Aspect Signature Resource

You must supply a main aspect template for Mail Service records, Catalog records, and Combined records required for your setup template (see Table 4-1 on page 4-64). The aspect signature resource provides the base resource ID for the aspect template and specifies that the template is a main aspect template. Because users cannot drag records out of the Key Chain or drop them in, the drop-related fields of the aspect signature resource are not significant. The signature resource for an aspect template is of type `'deta'`.

```
resource 'deta' (kSurfWriterAspect, purgeable)
{
   0, dropCheckConflicts, isMainAspect
};
```

kDETTemplateName

The template name resource is required of all templates. It has a resource ID with an offset of `kDETTemplateName` from the template's base (signature) resource ID.

```
resource 'rstr' (kSurfWriterAspect+kDETTemplateName, purgeable) {
   "kAspectName"
};
```

Your information page templates must use this name to refer to the aspect template that provides their property values.

kDETRecordType

The record-type resource specifies the record type to which the aspect template applies. The record-type resource has a resource ID with an offset of `kDETRecordType` from the template's base resource ID.

```
#define kServiceRecordType kMailServiceRecTypeBody "WAVE"

resource 'rstr' (kSurfWriterAspect + kDETRecordType, purgeable)
{
    kServiceRecordType
};
```

A main aspect template for a Mail Service record must specify the record type "aoce Mail
Service*xxxx*" where *xxxx* is the address extension type of the messaging system to which
your MSAM provides access. A main aspect template for a Catalog record must specify
the record type "aoce Directory*xxxx*". If the catalog is associated with a mail slot, *xxxx* is
the extension type of the external messaging system to which the slot's MSAM provides
access. If the catalog is not associated with a slot, *xxxx* is the `signature` field of the
catalog discriminator (`DirDiscriminator` structure). A main aspect template for a
Combined record must specify the record type "aoce Combined*xxxx*" where *xxxx* is the
extension type of the external messaging system to which the slot's MSAM provides
access. You can use the following constants when assigning a record type:

```
#define  kDirectoryRecTypeBody      "aoce Directory"
#define  kMailServiceRecTypeBody    "aoce Mail Service"
#define  kCombinedRecTypeBody       "aoce Combined"
```

kDETAspectName

A setup aspect template should specify a default name for the CE to display in the
Service field in the Key Chain. To provide this name, use an `RString` resource with
an offset of `kDETAspectName` from the template's base resource ID.

Note
For main aspect templates for records other than setup templates, the CE
automatically sets the `kDETAspectName` property to be the name of the
record. However, for records in the Key Chain, your template must
provide a resource to set this property explicitly. u

```
resource 'rstr' (kSurfWriterAspect+kDETAspectName, purgeable)
{
    "New Mail Server"
};
```

Your setup information page template can allow the user to change this name to the
name of the mail server or catalog server on which he or she has an account.

kDETAspectKind

Specify the kind of the record as it is to be displayed in a Get Info dialog box with an `RString` resource with an offset of `kDETAspectKind` from the template's base resource ID.

```
resource 'rstr' (kSurfWriterAspect+kDETAspectKind, purgeable)
{
    "SurfWriter Mail Service"
};
```

This resource is the only source for this information. Neither your code resource nor the user can change the value you specify in this resource. Unlike AOCE dNode windows, however, the Key Chain does not use the `kDETAspectKind` resource to determine what to display in the Kind field. The Key Chain uses the `kSAMAspectKind` resource (described next) for that purpose.

kSAMAspectKind

A setup aspect template must specify the name the CE should display in the Kind field in the Key Chain. To provide this name, use an `RString` resource with an offset of `kSAMAspectKind` from the template's base resource ID.

```
resource 'rstr' (kSurfWriterAspect+kSAMAspectKind, purgeable)
{
    "SurfWriter Mail Service"
};
```

This resource is the only source for this information. Your code resource can change the value you specify in this resource. You must also include a `kDETAspectKind` resource to specify the kind of the record as it is to be displayed in a Get Info dialog box.

kSAMAspectUserName

A setup aspect template should specify the default name the CE should display in the Name field in the Key Chain. To provide this name, use an `RString` resource with an offset of `kSAMAspectUserName` from the template's base resource ID.

```
resource 'rstr' (kSurfWriterAspect+kSAMAspectUserName, purgeable)
{
    "SurfWriter User"
};
```

Your setup information page template can allow the user to change this name to the account name on the mail system or catalog server. If your system does not use account names, you should use the name of the owner of the Key Chain for this property. You can obtain this name from the Setup record, where it is stored in an attribute of type "Local Name". The attribute type index for this attribute is `kLocalNameAttrTypeNum`.

kSAMAspectCannotDelete

The `kSAMAspectCannotDelete` property indicates whether the slot or catalog associated with this aspect can be deleted. A property value of 0 indicates that the slot or catalog can be deleted. Otherwise, it cannot be deleted. The default value of this property is 0. Your setup template can set the value of this property to prevent the user from deleting the slot or catalog once it has been added.

```
resource 'detn' (kSurfWriterAspect+kSAMAspectCannotDelete,
purgeable)
{
    1
};
```

kSAMAspectSlotCreationInfo

The slot creation information resource gives the Key Chain the information it needs to create and delete MSAM slots and CSAM catalogs. To provide this information, use a resource of type `'sami'` with an offset of `kSAMAspectSlotCreationInfo` from the template's base resource ID. This resource has the following Rez type definition:

```
type 'sami' {
    integer;                       // max number of catalogs/slots
    longint;                       // catalog signature, MSAM type
    byte notMSAM, servesMSAM;      // an MSAM template?
    byte notDSAM, servesDSAM;      // a CSAM template?
    rstring;                       // display when user clicks Add
    align word;
    rstring;                       // new record name
    align word;
};
```

The `integer` value is the maximum number of slots, catalogs, or combined services that your SAM can support. Set this to 0 if you can support an unlimited number of slots or catalogs.

The `longint` value identifies your type of service. For catalogs, this is the value of the `signature` field of the catalog discriminator (`DirDiscriminator` structure; see the chapter "AOCE Utilities" in *Inside Macintosh: AOCE Application Interfaces*). For slots, this is the extension type of the addresses. Addresses and address extension types are described in the chapter "Messaging Service Access Modules" in this book. For a catalog and mail service to work together, the catalog discriminator and address extension type values must be the same.

The two `byte` values specify whether your SAM is an MSAM or a CSAM. If it is a combined MSAM and CSAM, specify both `servesMSAM` and `servesDSAM` for these values.

The first `RString` value specifies the text for the dialog box that the Key Chain displays when the user clicks the Add button. This value in the `kSAMAspectSlotCreationInfo` resource replaces the `kDETAspectNewMenuName` resource used in other (non-setup) main aspect templates.

The second `RString` value specifies the name the Key Chain assigns initially to new records of this type. (The user can use the Key Chain information page to rename this record.) This value in the `kSAMAspectSlotCreationInfo` resource replaces the `kDETAspectNewEntryName` resource used in other (non-setup) main aspect templates.

Here is an example of a `kSAMAspectSlotCreationInfo` resource:

```
#define kSignature   'WAVE'
resource 'sami' (kSurfWriterAspect + kSAMAspectSlotCreationInfo,
                 purgeable)
{
    2,
    kSignature,
    servesMSAM,
    servesDSAM,
    "SurfWriter Combined Service",
    "untitled combined SurfWriter"
};
```

kDETAspectMainBitmap

Every main aspect template, including a setup template, must include an icon suite with a resource ID that has an offset of `kDETAspectMainBitmap` from the template's base resource ID. Suppose, for example, that you prepared an icon suite in a ResEdit file named SurfWriterIcons, that all of your icon resources had resource IDs of 0, and that your resource base ID was `kSurfWriterAspect`. In this case, you could use the following code to include the icon suite in your setup aspect template:

```
include "SurfWriterIcons" 'ICN#'(0) as
                'ICN#'(kMainAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'icl4'(0) as
                'icl4'(kMainAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'icl8'(0) as
                'icl8'(kMainAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'ics#'(0) as
                'ics#'(kMainAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'ics4'(0) as
                'ics4'(kMainAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'ics8'(0) as
                'ics8'(kMainAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'SICN'(0) as
                'SICN'(kMainAspect+kDETAspectMainBitmap, purgeable);
```

The icon suite must be included in the main aspect template and cannot be changed from a code resource or by the user.

kDETAspectWhatIs

Each setup main-aspect template must provide a help-balloon string. The Key Chain displays this string when the user enables Balloon Help online assistance and moves the cursor over an entry in the Key Chain to which this main aspect applies. The help-balloon string is in an RString resource with an offset kDETAspectWhatIs from the template's base resource ID.

```
resource 'rstr' (kSurfWriterAspect+kDETAspectWhatIs, purgeable) {
   "Contains information about this key, which represents a
     SurfWriter combined mail and catalog service."
};
```

kDETAspectCode

Every setup aspect template must include a code resource with an offset of kDETAspectCode from the template's base resource ID. For example, to include code that has been compiled and saved as the resource SurfWriterCode of type 'detc' with a resource ID of 0, add the following line to the setup aspect template:

```
include "SurfWriterCode" 'detc'(0) as
    'detc'(kSurfWriterAspect+kDETAspectCode, purgeable);
```

Your setup template code resource must call a variety of Collaboration toolbox functions and AOCE template callback functions to create records and attributes. See "Adding Catalog and Mail Services" beginning on page 4-5 for a description of each step involved in setting up catalog and mail services. Code resources and template callback functions are described in the chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces.*

The Address Template

Every MSAM must include an address template in the resource fork of the MSAM file. The template provides the human interface that allows a user to view, create, and edit the addresses the MSAM needs to send letters to recipients on its messaging system.

An address template consists of an aspect template and at least one information page template.

The lookup table (`'dett'` pattern) for an address must end with a pattern element of type `'Pref'`. This custom element type lets the Electronic Addresses information page set the preferred address radio buttons correctly.

The standard address information page is 259 pixels wide and 200 pixels high. It has a page-selection pop-up menu at location (8, 56, 30, 206) (top, left, bottom, right). It has a page-identifying large icon at (8, 8, 40, 40). Within the page are two radio buttons labeled "View as:", a Fields radio button and a String radio button. The string "View as:" is at location (49, 56, 63, 106). The Fields radio button is at location (48, 111, 64, 154). The String radio button is at location (48, 164, 64, 209). Between the view-as selector and the data is a dotted line, at location (72, 8, 73, 251).

Addresses with all types of tags are forwarded to the drop-send aspect by a built-in forwarder. For this reason, your address template does not need to handle drops.

For an example of an address template, see "Writing and Modifying Addresses" beginning on page 4-30.

Glossary

access controls   A set of bits that specify the types of operations a requestor is authorized to perform on a given catalog node, record, or attribute type.

address template   A set of AOCE templates that allow a user to enter address information into a User record.

AOCE   Apple Open Collaboration Environment.

AOCE catalog   A hierarchically arranged store of data in a format intelligible to the AOCE Catalog Manager. See also **external catalog, PowerShare catalog.**

AOCE messaging system   The set of PowerTalk system software and PowerShare mail servers that allows Macintosh users and processes connected over a network or via a modem to exchange information.

AOCE Setup catalog   See **PowerTalk Setup catalog.**

AOCE system software   The collection of Macintosh Operating System managers and utility functions that provide APIs for catalog, messaging, and security services. The AOCE system software includes the Standard Mail Package, the Standard Catalog Package, AOCE templates, the Interprogram Messaging Manager, the Catalog Manager, the Authentication Manager, and the Digital Signature Manager, as well as utility functions. See also **PowerTalk system software.**

AOCE template   A resource file that extends the AOCE extension to the Finder to display new types of data in catalogs or to display data in a new way. See also **aspect template, file type template, forwarder template, information page template, killer template.**

AOCE toolbox   The low-level APIs for the AOCE system software: the Authentication Manager, Catalog Manager, Interprogram Messaging Manager, and Digital Signature Manager. See also **Collaboration package, Collaboration toolbox.**

API   Application programming interface.

AppleMail format   See **standard interchange format.**

AppleTalk Secure Data Stream Protocol (ASDSP)   A networking protocol that provides reliable transmission of an encrypted stream of bytes between two authenticated entities on an AppleTalk internet.

approval file   A file you receive from a signature-authorization-issuing agency. You use this file to activate your signer file.

approval request   A notarized (or otherwise authorized) request to issue a public-key certificate. The approval request includes what is intended to be the public key of the certificate's owner.

approved signer file   See **signer file.**

approving agency   See **certificate issuer.**

ASDSP   See **AppleTalk Secure Data Stream Protocol.**

aspect   A structure in memory that contains properties provided by an aspect template. An aspect might also contain code provided by the code resource in an aspect template.

aspect template   An AOCE template that specifies how attributes in a record are to be parsed into properties for display in an information page. An aspect template can also specify certain constant property values and can contain a code resource that translates between property types and implements features in information pages. See also **information page template.**

attribute   The smallest unit of data in an AOCE catalog; the data within a record is organized into attributes. Each attribute has a type indicating the type of data, a tag indicating the format of the data, a creation ID, and data (the attribute value).

attribute creation ID A number assigned by a catalog that uniquely identifies an attribute value within a record. It persists for as long as the attribute value exists and is never reused. Not all catalogs support attribute creation IDs. See also **pseudo-persistent attribute creation ID.**

attribute tag See **attribute value tag.**

attribute type The type of data in an attribute; for example, telephone number or picture. A record can contain more than one attribute type, and there can be more than one attribute value of the same attribute type in a record.

attribute value The data in an attribute.

attribute value tag The format of the data in an attribute value.

authentication Verification of the identification of an entity on a network or of one end of a communication link.

authentication identity See **identity.**

Authentication Manager The part of the Macintosh Operating System that authenticates users of AOCE messaging and catalog services and provides authentication services to applications.

authentication server A secure network-based server that holds the client keys of users and services and generates credentials that allow users to do mutual authentication.

bcc recipient A "blind courtesy copy" recipient of a letter. Bcc recipients are not listed in copies of the letter recieved by To and cc recipients. See also original recipient.

block creator A four-character sequence that indicates which application created a message block; analogous to a file's creator in HFS.

block type A code that indicates the format of the data contained within a message block.

callback routine (1) An application-defined routine called by the Operating System. When you call certain functions, you provide a pointer to a callback routine, and the function installs your routine in memory. Then when a certain event occurs, the Operating System calls your callback routine. See also **completion routine.** (2) A function provided by the CE to provide a

service for aspect code resources. When the CE calls your code resource, your code resource can call the CE's callback routines.

catalog See **AOCE catalog.**

Catalog Browser A Finder extension that allows a user to search through an AOCE catalog by opening folders on the desktop.

catalog discriminator A name and reference number that uniquely identifies a catalog.

Catalog Manager The part of the Macintosh Operating System that manages the organization, reading, and writing of data in AOCE catalogs.

catalog node See **dNode.**

catalog service access module (CSAM) A code module, implemented as a device driver, that makes an external catalog available within an AOCE system by supporting the Catalog Manager API.

catalog service function A CSAM-defined function that responds to requests for AOCE catalog services from clients of the Catalog Manager.

Catalogs Extension An extension to the Finder that makes it possible for the Finder to display the contents of AOCE catalogs and for the user to edit the contents of records.

cc recipient A "courtesy copy" or secondary recipient of a letter. See also original recipient.

CE See **Catalogs Extension.**

certificate See **public-key certificate.**

certificate issuer The organization that authorized, or issued, a particular public-key certificate. Each certificate is digitally signed by its issuer.

certificate owner The person or organization to which a particular public-key certificate has been issued. Each certificate contains the public key of its owner.

certificate request See **approval request.**

certificate set A chain of public-key certificates that, combined with a digital signature, make up a full signature. A certificate set consists of the public-key certificate of the signer (owner),

digitally signed by the organization that issued the certificate; plus the certificate of the issuing organization, signed by the organization that issued that certificate; and so on, until the last signature is that of the prime issuing organization. The certificate set provides the signer's public key for decryption of the signer's signatures and ensures the validity of that public key.

certification authority   See **certificate issuer.**

chain of certificates   See **certificate set.**

client key   A key that is known only to a specific entity and to the authentication server.

Collaboration package   The high-level APIs for the AOCE system software collaboration managers: the Standard Mail Package and the Standard Catalog Package. See also **Collaboration toolbox.**

Collaboration toolbox   The low-level APIs for the AOCE system software collaboration managers: the Authentication Manager, Catalog Manager, and Interprogram Messaging Manager. See also **AOCE toolbox, Collaboration package.**

completion routine   A callback routine you can specify when you execute a function asynchronously. When the function completes execution, it calls your completion routine.

conditional view   A view in an information page that is displayed only if certain conditions are met in the aspect associated with that information page.

content block   A message block that contains the body of a letter in standard interchange format.

content enclosure   An enclosure that contains a letter's content. It may be the sole content in a letter or be accompanied by content in a content block, an image block, or both. See also **regular enclosure.**

context   A data structure used by some Digital Signature Manager routines to hold information and the results of calculations needed when processing data. See also queue context.

copying   As used by AOCE utility routines: the process of taking the contents of each field in a source structure and placing them in the

corresponding field of a destination structure. This process includes all nested structures as well. Compare **duplicating.**

creation ID   See **attribute creation ID, record creation ID.**

credentials   Encrypted information provided by a server and sent by an initiator to a recipient as part of the authentication process. The credentials contain the session key and the initiator's identification.

CSAM   See **catalog service access module.**

current block   The message block last added to a message.

decrypt   To restore encrypted data to its previous, legible (unscrambled) state. In most cryptographic systems, decryption is performed by mathematically manipulating the data with a large number called a *key.*

delivery indication   Information within a report that indicates the successful delivery of a specific message to a specific recipient.

DES   Data Encryption Standard. A standard algorithm for data encryption.

DES encryption   A form of secret-key encryption used by the Digital Signature Manager solely for keeping users' private keys secure. See also **secret-key cryptography.**

digest   A number, 16 bytes long, that is calculated from the contents of a given set of data. A digest is like a sophisticated checksum; it is almost impossible for two data sets of any size with any difference to yield the same digest value.

digital signature   A data structure associated with a document or other set of data. The digital signature uniquely identifies the person or organization that is signing, or authorizing the contents of, the data and ensures the integrity of the signed data. It is a digest of the data to which the signature applies, encrypted with the private key of the signer. A digital signature can be verified by decrypting with the signer's public key. Same as encrypted digest. See also **full signature.**

Digital Signature Manager The part of the Macintosh Operating System that manages digital signatures and certificates.

distinguished name The complete identifier of the owner or issuer of a certificate. A distinguished name includes elements such as common name, organization, street address, and country.

dNode A container within an AOCE catalog that contains records, other dNodes, or both.

dNode number A number assigned by a catalog that uniquely identifies a catalog node within that catalog. Not all catalogs support dNode numbers. See also **pathname.**

dNode window A Finder window that displays the dNodes and records contained in a dNode.

duplicating As used by AOCE utility routines: the process of copying the pointers to data structures and not the actual data structures themselves. Compare **copying.**

enclosure A file or folder sent along with a letter, like an attachment to a conventional hard-copy letter. See also **content enclosure**, **regular enclosure.**

encrypt To hide data by putting it into a scrambled (illegible) state, in such a way that its original state can be restored later. In most cryptographic systems, encryption is performed by mathematically manipulating the data with a large number called a key.

encrypted digest See **digital signature.**

encryption key See **key.**

extension type A four-character value that identifies a type of messaging system that uses a specific addressing convention; for example, an AppleLink system or an X.400 system.

external catalog A catalog or database accessible to AOCE-enabled applications through the Catalog Manager API. For a user to have access to an external catalog, the user's AOCE system must include a CSAM for that catalog service.

external messaging system Any non-AOCE messaging system.

external service A service that is not provided automatically with PowerTalk system software and PowerShare servers.

file type template An AOCE template that extends the list of file types that may contain an AOCE template. During system startup, the Catalogs Extension searches for AOCE templates in files whose types are on the list.

focus box See **focus rectangle.**

focus rectangle A heavy border around a panel or around the content portion of a window. This border indicates to the user that the area it encloses is active and that any subsequent key-down event pertains to that portion of the window. Also called focus box.

foreign dNode A dNode in a PowerShare catalog used by AOCE system software to route messages to an external messaging system through a server MSAM.

Forwarder record A catalog record that contains identifying information about a server MSAM.

forwarder template An AOCE template that allows existing aspect templates and information page templates to be used for new types of records and attributes.

From recipient The sender of a message. See also original recipient.

full digital signature See **full signature.**

full signature A digital signature plus the certificate set of the signer. The Digital Signature Manager creates and verifies full signatures. Same as full digital signature.

identity A number used as shorthand for the name and key or name and password of a user or service. **See also local identity, specific identity.**

image block A message block containing a graphic representation of a letter's content. It may be the sole content in a letter or be accompanied by content in a content block, a content enclosure, or both. The format of data in an image block is sometimes referred to as *snapshot format.*

incoming message A message coming into an AOCE system from an external messaging system.

incoming queue   A queue belonging to a mail slot into which a personal MSAM puts letters coming into an AOCE system from an external system.

information page   A formatted display of data and controls, similar in appearance to a dialog box, showing information about an AOCE catalog record or a portion of a record. See also **information page template.**

information page template   An AOCE template that defines the layout and contents of an information page, using the properties in a specific aspect.

information page window   A window that contains one or more information pages. If the window contains more than one information page, only one information page is displayed at a time. In that case, the window contains a pop-up menu with a list of the information pages available.

initiator   The originator of the authentication process.

intermediary   A representative of a user or service that uses a proxy to obtain credentials for mutual authentication and then performs some function for the user or service represented.

Interprogram Messaging Manager (IPM)   The part of the Macintosh Operating System that manages the creation, sending, and receiving of messages. IPM messages conform to a specific structure and can be transmitted over an AppleTalk network or any other communication link. The Interprogram Messaging Manager provides store-and-forward messaging services for Macintosh computers.

issuer   See **certificate issuer.**

issuing organization   See **certificate issuer.**

key   A number used by an encryption algorithm to encrypt or decrypt data.

Key Chain   See **PowerTalk Key Chain.**

Key Chain Access Code   The master password providing access to a PowerTalk Key Chain.

killer template   An AOCE template that disables other AOCE templates. A killer template can disable any type of AOCE template except another killer template.

large-catalog mode   A set of algorithms used by certain components of a PowerTalk system when retrieving information from large catalogs and displaying that information to the user.

letter   A type of message consisting of a defined set of message blocks. A letter is intended to be read by a person. See also **mailer, non-letter message.**

letter attribute   A piece of information about a letter stored in the letter header or the letter's message summary. Letter attributes include information such as the sender, the subject, the time the letter was sent, and so forth. Not to be confused with **attribute.**

letter header block   A message block found in every letter. It contains recipient information and letter attributes.

local identity   A number used as shorthand for the name and password of the principal user of a particular computer. **A local identity gives the user access to all the services for which names and passwords are stored in the PowerTalk Setup catalog. See also specific identity.**

lookup table   A resource in an aspect template that parses attribute values into properties and properties into attribute values. A lookup table contains an entry for each type of attribute value to be translated into and from properties.

mail   A term used to refer collectively to letters.

mailer   A region added to a document window that transforms the document into a letter. The mailer enables the user to enter addresses and subject information, enclose other files and folders in the letter, and add a digital signature to the letter.

mailer set   All of the mailers belonging to a forwarded letter.

mail slot   A personal MSAM slot that serves to transfer letters. See also **slot.**

main aspect   An aspect that contains the properties the CE needs to fill in the data for an item in a sublist. Compare **main view aspect.**

main aspect template   A template for a main aspect.

main enclosure   See **content enclosure.**

main view aspect   An aspect that provides the properties for all the views in the main portion of an information page; that is, all of the information page except for the items in a sublist. Compare **main aspect.**

Master Key password   The password of the principal user of a computer. This password unlocks the local identity and provides access to the services represented in the PowerTalk Setup catalog.

message   The basic unit of communication defined by the Interprogram Messaging Manager. The term *message* is used as an inclusive term to refer both to letters and non-letter messages. See also **letter, non-letter message.**

message block   A component of a message consisting of a sequence of any number of bytes whose format is governed by the block creator and block type.

message creator   A four-character sequence that indicates which application created a message; analogous to a file's creator in HFS.

message family   A set of messages grouped according to similar characteristics. Messages of the same family conform to the syntax of a defined set of message block types and their associated semantics.

message header   That part of a message that contains control information about the message such as the message creator and message type, the total length of the message, the time it was submitted, addressing information, and so forth.

message mark   A marker, used by the IPM Manager, that points to the current location within a message that is being created.

message queue   A set of messages maintained by the IPM Manager on a recipient's disk or the disk of a message server.

message summary   A set of data used by the Finder to display an incoming letter to a user.

message type   A code that indicates the semantics of the message, the block types the message should contain, and the relationships among the various blocks in the message.

messaging service access module (MSAM)   A foreground or background application that makes an external messaging system accessible from within an AOCE system. It translates and transfers letters, non-letter messages, or both between an AOCE system and an external messaging system. See also **personal MSAM, server MSAM.**

messaging slot   A personal MSAM slot that serves to transfer non-letter messages. See also **slot.**

messaging system   A combination of hardware and software that gives users or processes the ability to exchange messages.

MSAM   See **messaging service access module.**

mutual authentication   Authentication of both ends of a communication link accomplished by exchanging a series of encrypted challenges and replies.

nested letter   A complete letter included whole within another letter.

nested message   Any type of message included whole within another message.

nesting level   An indication of how many messages are nested within a given message. For example, a letter that contains one nested letter has a nesting level of 1, and a letter that contains no nested letters has a nesting level of 0.

non-delivery indication   Information within a report that indicates unsuccessful attempts to deliver a specific message to a specific recipient.

non-letter message   A message sent from one application or process to another, not intended to be read by people. Compare **letter.**

online mode   A mode of operation available only to personal MSAMs in which the MSAM actively manages letters in a user's AOCE mailbox and in the user's accounts on external messaging systems, reflecting changes in one to the other, keeping both ends synchronized to the degree possible.

original recipient   Any of four specific types of recipient that can be specified by the sender of a message: To, From, cc, or bcc. An original recipient may be a group address. A non-letter message can include only From and To recipients. See also **resolved recipient.**

outgoing message   A message that is leaving an AOCE system to go to an external messaging system.

outgoing queue   A queue from which an MSAM reads messages that it must deliver to an external messaging system.

owner   See **certificate owner.**

packing   The process of compacting or "flattening" a complex data structure into a sequence of bytes. Compare **unpacking.**

parse function   A CSAM-defined function that responds to requests for AOCE parse services from clients of the Catalog Manager.

partial pathname   In an AOCE catalog, a value that uniquely identifies a catalog by specifying a dNode number and continuing with the name of each dNode under that one to the dNode in question.

password   In digital signatures, a set of characters used as a key to encrypt and decrypt a certificate owner's private key.

password encryption   See **DES encryption.**

pathname   In an AOCE catalog, a string that uniquely identifies a catalog node by specifying the name of each catalog node in the catalog starting from the first node under the root node and including each intervening node to the node in question. See also **dNode number.**

personal catalog   An AOCE catalog created and managed by the Catalog Manager. A personal catalog is an HFS file located on a user's local disk. A personal catalog can store any records that can be kept in a PowerShare catalog and is often used to store frequently used information from such a catalog.

personal MSAM   An MSAM that transfers messages between the user's Macintosh and specific user accounts on an external messaging system. A personal MSAM runs on a user's Macintosh. Compare **server MSAM.**

physical queue   The actual data of a message queue residing on a disk. A physical queue can have any number of associated virtual queues. See also **virtual queue.**

PMSAM   See **personal MSAM.**

PowerShare catalog   An AOCE server-based catalog provided by Apple Computer, Inc. See also **external catalog.**

PowerShare server   A server installed on an AppleTalk network to provide catalog services to any number of entities on that network. A PowerShare server can also identify and authenticate users to ensure that only authorized people or agents gain access to the catalog information.

PowerTalk Key Chain   The PowerTalk software that sets up and maintains a user's PowerTalk Setup catalog.

PowerTalk Setup catalog   A special personal catalog that contains information about the mail and messaging services, catalog services, and other services available to the owner of the computer. See also **local identity.**

PowerTalk system software   Apple Computer's implementation of the AOCE system software for use on Macintosh computers. The PowerTalk system software includes desktop services as well as all of the services of the AOCE system software managers.

private key   One of a pair of keys needed for private-key cryptography. Every user has a private key kept by the user and known only to the user.

property   An individual, self-contained piece of information, such as a number or a string. A property is defined in an aspect template and stored in an aspect in memory.

property command   Any command handled by your AOCE template code resource's `kDETcmdPropertyCommand` routine. The CE calls your code resource with the `kDETcmdPropertyCommand` routine selector when the user clicks a button or checkbox in your information page, when the user selects an item in a pop-up menu in your information page, and in a few other circumstances.

property number   A reference number assigned to a property by an aspect template. The property number uniquely identifies that property within that aspect.

property type   A constant associated with a property that specifies the nature of the data in the property value. For example, a property type can be a number, a string, or a custom type defined by a developer.

property value   The data associated with a property.

proxy   A privilege provided by a user or service to an intermediary. The proxy allows the intermediary to be authenticated as the user or service for a limited period of time.

pseudonym   An alternative name for a record in a Catalog Manager routine.

pseudo-persistent attribute creation ID   A number that uniquely identifies an attribute value within a record. It persists from the time the CSAM is opened at system startup until system shutdown. See also **attribute creation ID.**

public key   One of a pair of keys needed for public-key cryptography. Every user has a public key, which can be distributed to other users.

public-key certificate   A document that contains, among other information, the name and public key of a user. The user is the owner of the certificate. See also **signed certificate, certificate set.**

public-key cryptography   A system of cryptography in which every user has two keys to encrypt and decrypt data: a public key and a private key. Data encrypted with a user's public key can be decrypted only with that same user's private key. Likewise, data encrypted with a user's private key can be decrypted only with that user's public key.

quasi-batch mode   A mode of operation available only to personal MSAMs in which the MSAM complies with the minimum requirements of online mode. See also **online mode.**

queue context   A grouping of virtual message queues. When you close a queue context, you simultaneously close all of the queues associated with that context. See also **virtual queue.**

recipient   (1) The end of a communications link that receives credentials and a challenge from the initiator. The recipient must respond correctly to establish an authenticated connection. (2) An addressee on an AOCE message. See also **original recipient, resolved recipient.**

record   The fundamental container for data storage in an AOCE catalog; analogous to a file in the HFS hierarchy. A record can contain any number of attributes.

record alias   A record that enables you to store information about another record. For example, an alias could store in its attribute value the record location information for the original record.

record creation ID   A number that uniquely identifies a record within a catalog. Not all catalogs support record creation IDs.

record ID   The identity of a record, comprising the record name, record type, record creation ID, and record location information. See also **record creation ID**, **record type.**

record reference   An attribute that identifies a specific catalog record.

record type   A value that indicates the type of entity represented by a record—for example, LaserWriter, User, or Group.

regular enclosure   Any message enclosure that is not a content enclosure. See also **content enclosure**, **enclosure.**

report   A message with a defined set of message blocks used to send delivery and non-delivery indications to the sender of the message.

resolved recipient   A recipient to which an MSAM must deliver a message. See also **original recipient.**

RSA   RSA Data Security, Inc., a prime issuing organization for public-key certificates.

SAM   See **service access module.**

secret-key cryptography   A system of cryptography in which a single key is used to both encrypt and decrypt data. All who wish to share information must share the same key and keep it secret from all others.

server   A program or process that provides some service to other processes on a network.

server MSAM   An MSAM that transfers messages for multiple users on the AppleTalk network to which it is connected. It transfers messages between a PowerShare mail server and an external messaging system. A server MSAM must run on the same Macintosh as a PowerShare mail server. Compare **personal MSAM.**

service access module   A software component that provides a PowerTalk user with access to external mail and messaging services or catalog services.

session key   A key provided by an authentication server to be used by both the initiator and the recipient for mutual authentication. The session key remains valid for a limited time period.

Setup catalog   See **PowerTalk Setup catalog.**

Setup record   A record in the PowerTalk Setup catalog containing record references to all records in the PowerTalk Setup catalog that represent slots, catalogs, and other items.

setup template   A set of AOCE templates that allow a user to install and configure a service access module.

sign   As used by the Digital Signature Manager: To create a digital signature and affix it to a document or other piece of data. By signing, the signer authorizes the content of the data, protects it from alteration, and asserts his or her identity as the signer.

signature   See **digital signature.**

signature resource   A resource in an AOCE template that specifies the type of the template and the base ID number for the template. Other standard template resources have ID numbers equal to the signature resource's ID number plus some offset value.

signed certificate   A public-key certificate that has been digitally signed by its issuer. Like any digital signature, the signature on a certificate ensures the integrity of the certificate (including its public key) and proves the identity of the signer (the issuer of the certificate).

signed digest   See **encrypted digest.**

signer   The individual or organization that signs a document or other piece of data. To create a signature, a signer must be the owner of a public-key certificate.

signer file   A file used by a signer to create a digital signature. It consists of the signer's encrypted private key and the signer's certificate set.

Simple Mail Transfer Protocol (SMTP)   A protocol for the exchange of electronic mail. Computers connected to the Internet often use this protocol.

slot   A collection of information about one account on an external messaging system. The information includes whatever is necessary to allow an MSAM to access the account and retrieve and send messages. See also **mail slot, messaging slot.**

SMSAM   See **server MSAM.**

snapshot format   See **image block.**

specific identity   A number used as shorthand for the name and key of an alternate user on a computer to provide access to a specific catalog or mail service. See also **local identity.**

stand-alone attribute   A record that contains only one attribute, extracted from another record. Although technically a record, the AOCE software treats a stand-alone attribute like an attribute in most circumstances. The record type of a stand-alone attribute begins with the value of the constant `kAttributeValueRecTypeBody`.

Standard Catalog Package   The part of the Macintosh Operating System that manages find and browse panels for AOCE catalogs.

standard content   See **standard interchange format.**

standard interchange format   A set of data formats that consists of plain text, styled text, sound (AIFF), images (PICT), and QuickTime movies (`'MooV'`).

Standard Mail Package   The part of the Macintosh Operating System that manages mailers and makes it easy for applications to create and send letters.

standard mode   A mode of operation available to server MSAMs and to personal MSAMs that deal with non-letter messages. An MSAM operating in standard mode hands off an incoming message to an AOCE system. It is the AOCE system, not the MSAM operating in standard mode, that is responsible for delivering the message to the ultimate destination.

store-and-forward gateway   A link between different messaging systems, sometimes bridging different physical media, providing temporary data storage, and, where necessary, address translation.

store-and-forward messaging   A method of delivering messages that provides for temporary storage and forwarding of a message from one location to another, sometimes through several intermediate store-and-forward gateways or servers.

store-and-forward server   A server that provides store-and-forward messaging services. PowerShare servers are store-and-forward servers.

sublist   A list of attributes that appears as a distinct subset of the items displayed in an information page window, or a list of records that appears in a dNode window.

tag   See **attribute value tag.**

TCP/IP   Transmission Control Protocol/Internet Protocol. The major transport protocol and the network layer protocol typically used in communicating messages over the Internet.

template   See **AOCE template.**

To recipient   A principal recipient of a message. See also original recipient.

unapproved signer file   A file created by the MacSigner application when it creates an approval request. The unapproved signer file contains a DES-encrypted number that is intended to be the user's private key.

universal coordinated time (UTC)   The same as Greenwich Mean Time (GMT); the standard time as established by the Royal Observatory at Greenwich, England.

unpacking   The process of reconstructing a data structure from a sequence of bytes. Compare **packing.**

User record   A catalog record representing an entity that has an account on an AOCE messaging or catalog server. A User record contains electronic addresses and biographical information about the entity that can be read by users of the system, as well as information about the entity's access privileges and password for use by the AOCE software.

UTC   See **universal coordinated time.**

verify   To establish the authenticity of a digital signature. Verification consists of determining that the signed document has not changed since it was signed and affirming that the public key used to decrypt the signature is valid.

view   An item or field in an information page displaying one or more property values.

view list   A data structure that specifies individual views on an information page. Each item in the list includes the graphic rectangle containing the view, the number of the property that provides the information to be displayed, the type of view, and information specific to that view type.

virtual queue   A view of a physical message queue through which an application can open, close, and list messages. More than one virtual queue can be associated with a single physical queue. See also **physical queue.**

Index