INSIDE MACINTOSH

AOCE Application Interfaces

Addison-Wesley Publishing Company

Reading, Massachusetts Menlo Park, California New York
Don Mills, Ontario Wokingham, England Amsterdam Bonn
Sydney Singapore Tokyo Madrid San Juan
Paris Seoul Milan Mexico City Taipei

Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

ISBN 0-201-40848-1
1 2 3 4 5 6 7 8 9-CRW-9897969594
First Printing, June 1994

Library of Congress Cataloging-in-Publication Data

Apple Computer, Inc.
Inside Macintosh : AOCE application interfaces.
p. cm.
Includes index.
ISBN 0-201-40848-1
1. Macintosh (Computer) 2. Application software. I. Title.
QA76.8.M3A668 1994
005.265—dc20 94-16207
CIP

Contents

Figures, Tables, and Listings

Chapter 1 — Introduction to the Apple Open Collaboration Environment 1-1

Chapter 2 — AOCE Utilities 2-1

Chapter 3 — Standard Mail Package 3-1

About This Book

This book, *Inside Macintosh: AOCE Application Interfaces*, describes the application programming interfaces (APIs) to PowerTalk system software and to services provided by 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 desktop interface and PowerShare servers use to implement their many features. You can use the AOCE software to enhance your application's capabilities. 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. The PowerShare collaboration servers provide mail, messaging, catalog, security, and time services.

This book shows in detail how your application can take advantage of the system software enhancements offered by the AOCE software. It provides a complete technical reference to AOCE data structures, AOCE utility routines, the Standard Mail Package, the Standard Catalog Package, AOCE templates, the Digital Signature Manager, the Interprogram Messaging Manager, the Catalog Manager, and the Authentication Manager.

You need this book if you want to incorporate AOCE features into your application or to write AOCE templates to extend the Finder's capability to display information in an AOCE catalog. If you are interested in extending the capabilities of the AOCE system software to take advantage of services offered by external databases and messaging systems, see *Inside Macintosh: AOCE Service Access Modules*.

What to Read

The PowerTalk system software and PowerShare servers add many new capabilities to the Macintosh Operating System with which you might not yet be familiar. For this reason, you should read the first chapter, "Introduction to the Apple Open Collaboration Environment," before attempting to use any of the software described in this book. That chapter 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 terms used throughout this book.

The AOCE software uses several complex, packed data structures whose exact contents are private. You must often work with unpacked forms of these

structures, compare packed data structures whose contents you cannot read, and convert between packed and unpacked forms of structures. For this reason, the AOCE software provides a variety of utility routines that you can use to pack, unpack, compare, and otherwise manipulate these data structures. The chapter "AOCE Utilities" describes these data structures and utility routines. You should read this chapter before reading any of the other chapters in this book.

The AOCE software provides several high-level programming interfaces that you can use to add PowerTalk and PowerShare capabilities to both existing and new applications.

The chapter "Standard Mail Package" tells you how to add PowerTalk mail capabilities to any application.

The chapter "Standard Catalog Package" tells you how to add catalog-browsing and searching services to any application.

If you want to extend the ability of the Finder to display information in AOCE catalogs, you can write a set of resource files called AOCE templates that describe the data to be displayed and the format in which it is shown. AOCE templates can include code resources that respond to user actions and manipulate data. The chapter "AOCE Templates" describes the template resources in detail and shows sample templates to help you get started writing your own.

A user can use the PowerTalk system software to add a digital signature to any file in the Finder or to sign any letter that has a PowerTalk mailer attached. In addition, if you want to allow a user to add a digital signature to your application's documents or to any portion of a document, you can use the information in the chapter "Digital Signature Manager" to add this capability to your application.

In addition to these high-level programming interfaces, the AOCE software provides three low-level managers that you can use to implement messaging, catalog, and authentication features in your application. These interfaces are intended for use by experienced Macintosh programmers who have a good knowledge of Macintosh system software. Whereas the chapters that describe the high-level APIs all include sample code and programming hints, the chapters on the low-level managers provide less of this sort of information. As with all the chapters, they do provide a complete reference to all of the data structures and functions provided by these managers.

You can use the AOCE Interprogram Messaging Manager, described in the chapter "Interprogram Messaging Manager," to send messages between processes or applications without user intervention. This chapter also may be of interest to anyone using the Standard Mail Package or writing a messaging service access module (MSAM). An MSAM is an interface between an external mail or messaging system and the PowerTalk system software.

The chapter "Catalog Manager" describes functions you can use to get information about AOCE catalogs and to manipulate the data in catalogs. You

can use this information to provide catalog-related functions beyond those provided by the Standard Catalog Package. This chapter is required reading for anyone writing a catalog service access module (CSAM). A CSAM is an interface between an external catalog or database and the PowerTalk system software.

The chapter "Authentication Manager" describes the functions provided by the AOCE authentication service. Some of these functions require you to use PowerShare collaboration servers. Other functions described in this chapter allow you to implement your own authentication system.

There is one appendix, "PowerTalk Built-in Templates," which describes some of the details of the AOCE templates that are built into the PowerTalk system software. You can use this information to gain access to the information in these templates or to provide additional templates that work with and extend the built-in templates.

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

Format of a Typical Chapter

Almost all chapters in this book follow a standard structure. For example, the chapter "Standard Mail Package" contains these sections:

- "About the Standard Mail Package." This section provides an overview of the features provided by the Standard Mail Package.
- "Using the Standard Mail Package." This section describes the tasks you can accomplish using the Standard Mail Package. It describes how to use the most common routines, gives related user interface information, provides code samples, and supplies additional information.
- "Standard Mail Package Reference." This section provides a complete reference to the Standard Mail Package by describing the data structures and functions that it uses. Each function description also follows 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.
- "Summary of Standard Mail Package." This section provides the Standard Mail Package's C interface, as well as the Pascal interface, for the constants, data structures, functions, and result codes associated with the Standard Mail Package. It also includes some assembly-language interface information.

Some chapters include additional main sections that introduce new concepts or discuss certain concepts in detail. For example, in the chapter "Digital Signature Manager," the section "About Public-Key Certificates" describes the

public-key certificates used by the Digital Signature Manager to verify the identity of a signer. In the chapter "Interprogram Messaging Manager," the section "Addressing IPM Messages" describes the address format used by the Interprogram Messaging Manager.

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

Four 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 2-6.) 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 5-29.) 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 3-64.) s

S **WARNING**
A warning like this indicates a potential problem that you should be aware of as you design your software. Failure to head such a warning could result in a system crash or loss of data. (An example appears on page 5-197.) 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 column at the far left 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 in put 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.

This book occasionally uses *SurfWriter* and *SurfDB* as the names of applications for illustrative purposes; these are not actual products of Apple Computer, Inc. In addition, the name *River Change Systems* is used to represent a fictitious company.

For More Information

APDA is Apple's worldwide source for hundreds of development tools, technical resources, training products, and information for anyone interested in developing applications on Apple platforms. Customers receive the *APDA Tools Catalog* featuring all current versions of Apple development tools and the most popular third-party development tools. 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	1-800-282-2732 (United States) 1-800-637-0029 (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

CHAPTER 1

Introduction to the Apple Open Collaboration Environment

Contents

This chapter describes the Apple Open Collaboration Environment and the Macintosh system software components that compose the AOCE APIs. It provides an introduction to the capabilities of the AOCE software and some of the ways you can use AOCE technology to enhance your application or to solve specific problems in workflow and collaboration among people and computers.

Whereas the other chapters of this book are intended for programmers and application developers with a working knowledge of the C programming language, this chapter is also of interest to anyone who wants to gain a deeper understanding of the capabilities and uses of the AOCE technology and the PowerTalk system software. To read this chapter you should be familiar with the fundamentals of programming for the Macintosh computer and the basic concepts of interapplication communications. For introductions to these topics, see *Inside Macintosh: Overview* and *Technical Introduction to the Macintosh Family*. It will also be helpful if you have spent some time using the PowerTalk software on your own computer.

This chapter begins with a brief statement about the relationship of the AOCE software to the rest of the Macintosh Operating System. It then presents several scenarios describing possible uses of the AOCE technology to solve collaboration and workflow problems for a mythical company. The third section of this chapter describes each of the components of the AOCE system software in some detail. The final section introduces some basic concepts used throughout the remaining chapters in this book.

About AOCE System Software

The AOCE system software adds to the capabilities of the Macintosh Operating System and of the Finder. To the Macintosh Operating System, the AOCE software adds a transport-independent messaging service that applications can use for mail and for interapplication communications. Unlike the Program-to-Program Communications (PPC) Toolbox, the AOCE Interprogram Messaging (IPM) service does not require that both the sending and receiving applications be simultaneously connected to a network. In fact, IPM does not require that the communication ends be connected to a network at all: it operates over modems and can work over any other message-transport mechanism for which a developer cares to write an interface.

Although collaboration among users and applications implies the existence of some messaging service, the AOCE system software supports collaboration in other ways as well: it provides catalogs, which store not only addresses but any sort of data you wish to put in them; it authenticates the identities of the sender, routers, and receivers of messages; and it allows users and applications to guarantee the identity of the sender of a message and the integrity of the data in a message by affixing a digital signature to the data.

The AOCE software extends the capabilities of the Finder by adding a universal mailbox capable of holding every sort of electronic mail received by the user: E-mail, voicemail, faxes, and so forth. It also provides a familiar folder-based interface that allows users to

browse the contents of AOCE catalogs. Developers can extend both the types of mail that the mailbox can receive and the types of catalog data that the Finder can display.

The following sections illustrate these uses of the AOCE software and describe the components of the AOCE software in more detail.

Some Uses for AOCE Software

To understand some of the ways you can use AOCE software to improve productivity and workflow efficiency, consider a hypothetical situation. Suppose that the president and CEO of the River Change Systems company wants to improve the efficiency of her company's operations. She uses her favorite word processor, SurfWriter, to write a memo on her Macintosh computer to all of her department heads soliciting suggestions. Rather than printing the memo and using the company mail service to deliver it, she simply adds a PowerTalk mailer to the memo (Figure 1-1), turning it into a PowerTalk letter. She then drags the addresses for her department heads from her personal catalog on her Macintosh desktop, drops them into the mailer, and chooses Send from the Mail menu to send the letter over her company's AppleTalk network. Not all of the department heads at the River Change Systems company have SurfWriter on their Macintosh computers, but those who don't can read the memo using the AppleMail letter application that is provided with the PowerTalk system software.

Figure 1-1 Letter containing an AOCE mailer

Here are some of the ideas that the department heads come up with.

The Company Store Catalog

The River Change Systems company has over 12,000 employees worldwide, and they make extensive use of the company store, buying gift and office items for themselves and friends. The company store stocks hundreds of items and prints a full-color catalog four times a year as well as numerous notices and announcements of sales and special events. Overseas employees of the company sometimes have to wait weeks for the delivery of their catalogs, occasionally missing out on limited-stock items. Although the store grosses nearly $500,000 a year, it barely breaks even because of its low profit margins.

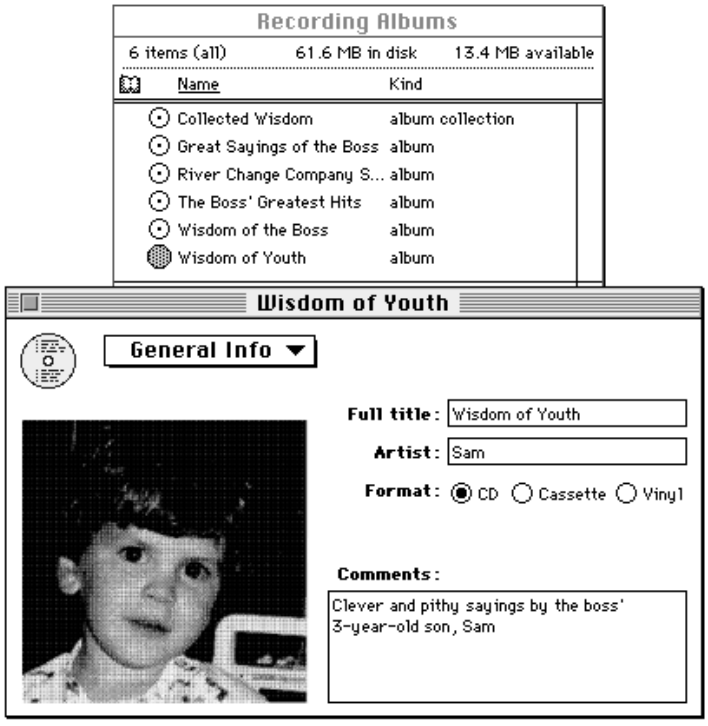
The manager of the company store suggests a PowerTalk-based system that will replace the company store's printed catalog and integrate it with the store's inventory-tracking system. This is how the system works:

The paper catalog is replaced by a server-based AOCE catalog. The basis for the catalog is the store's existing inventory database, which was created using the SurfDB database application. A catalog service access module (CSAM) in each user's computer interfaces the SurfDB database server to that user's PowerTalk system. Each item in the catalog

appears as an icon in a folder on the employee's desktop when he or she logs on to the PowerTalk system. Overseas employees, who are connected via microwave links to the company's extended AppleTalk network, have the same access to the store's catalog as anyone else. Even employees working at home or using Macintosh PowerBook computers while traveling can dial in to the network by using the AppleTalk Remote Access application.

When an employee double-clicks an item in the catalog, a window appears that contains a stack of PowerTalk information pages displaying information about the item, including a picture (Figure 1-2). Because the catalog draws its information from the store's inventory database, the catalog includes information about whether the item is in stock, how many are available, whether it is backordered if not currently available, and—depending on the type of item—such data as what sizes and colors are in stock. One of the information pages in the stack is an order form.

Figure 1-2 Catalog-item information page



When employees want to order items from the catalog, they fill out the order form on the screen. To fill in the name and address fields, they can drag the information from a personal catalog or a PowerTalk information card on their desktop and drop it on the order form. They can do the same thing to add an address of a third party for delivery of a gift. For payment, employees can authorize deductions from their paychecks or can use credit cards. In either case, they must affix a digital signature to the order form to guarantee the authenticity of the order.

When the employee clicks the Send button on the order form, the code resource associated with the information page sends the signed order form to the company store and updates the database to indicate the sale. It also sends a message to the company's accounting database informing it of the sale, the amount due, and the form of payment. In the case of a payroll deduction, the accounting database automatically reduces the employee's paycheck by the indicated amount and marks the account as paid. If the employee used a credit card, the database sends a message to the credit card bank requesting payment.

From the time the employee sends the order, no human interaction is necessary to complete the financial transaction. Once the payment has been approved, the accounting database application sends a PowerTalk message to the company store warehouse, where the order, shipping list, and mailing label are automatically printed out.

In addition to this catalog-based system, the company store uses PowerTalk to send announcements of sales and special events to all the employees. The company store marketing director uses SurfWriter to create the announcement, adds a PowerTalk mailer, and sends it to several group addresses that include all the employees of the company. Each announcement is sent in both the SurfWriter native format and the AppleMail image format, so each employee can read the letter whether or not that employee has the SurfWriter application.

Purchasing

The manager of the purchasing department suggests the use of PowerTalk to automate the routing of purchasing requests. An employee wishing to make a purchase opens the company's forms application and chooses New Form from the File menu. This opens a catalog-selection dialog box that lets the user select the proper form from a PowerTalk catalog. The application displays the form, and the employee fills it in and adds a digital signature.

When the employee clicks a Send button at the bottom of the form, the application automatically looks up the AppleTalk address of the employee's manager in a PowerShare catalog and sends the form to that manager. The form appears as a forms application file in the manager's PowerTalk mailbox. The manager reads the form, approves or disapproves it, adds a digital signature, and clicks Send. At each routing step the forms application sends dated messages in AppleMail letter format to the requestor and to the purchasing department indicating that a request has been made, who made it, who has last forwarded it, and to whom it has most recently been sent. If at any point the request is denied, the forms application sends a letter, with a copy of the form attached, informing the original requestor that the request has been denied, when it was denied, and who denied it.

When the fully approved form arrives at purchasing, the requested item can be purchased. If the vendor is also using PowerTalk, the purchasing department can forward the purchase order over the network or by modem and arrange for the electronic transfer of funds from one bank to the other. The entire transaction can be completed without printing any paper. What's more, a digital signature guaranteeing the originator and integrity of the request has been added at each step. If the request is

delayed at any point in the routing, the purchasing department can check the latest routing report to determine who is holding it up and for how long.

Sales

The large sales staff of the River Change Systems company have been losing sales when they miss messages either because they are on the road or because a salesperson neglects to check all of the different types of electronic messages that can come in each day: voicemail, faxes, AppleLink, and internal E-mail. In addition, sales staff need some way to check inventory and enter orders while on the road.

By adding PowerTalk to their systems, each salesperson can gather all of his or her messages in a single mailbox, which the salesperson can check whether in the office or on the road. In addition, by using a PowerTalk CSAM to access the company's inventory database, the salesperson can check inventory and place orders at any time. Each salesperson can also use a personal PowerTalk catalog stored on his or her notebook computer to keep track of accounts and to store addresses, phone numbers, and personal information about clients.

The Components of the AOCE Software

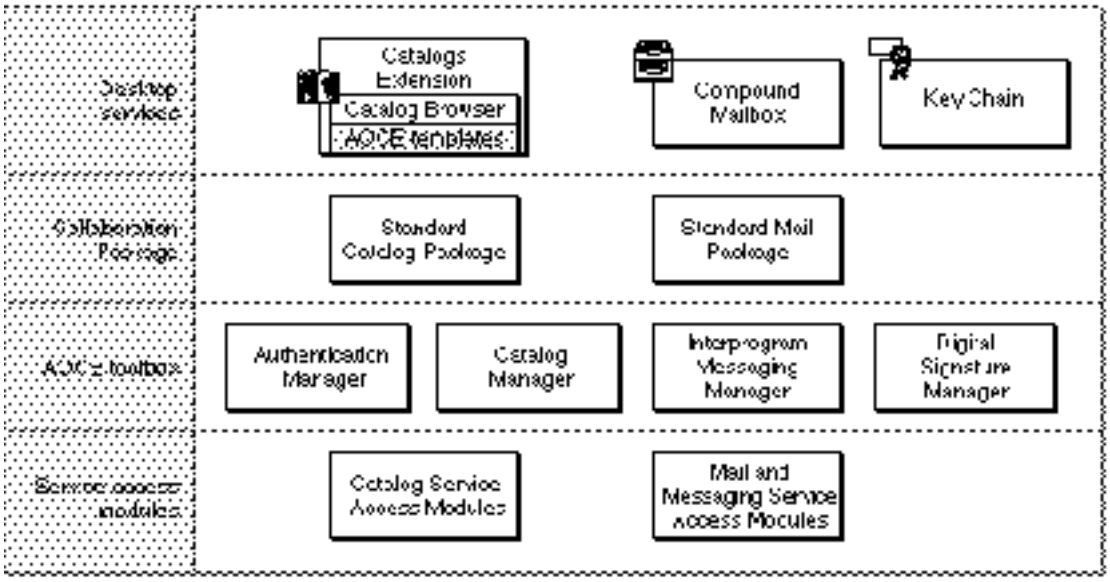
Rather than suggesting more uses for the AOCE technology, at this point it might be more useful to describe the components that constitute the AOCE software and to show how each of the features or functions in the foregoing examples is implemented with AOCE APIs.

At the heart of AOCE technology are three basic services: messaging, authentication, and catalogs. These are enhanced by an independent module, Digital Signatures, plus an extensible version of the Finder. The AOCE software provides APIs for each of these components: the Interprogram Messaging Manager, the Authentication Manager, the Catalog Manager, the Digital Signatures Manager, and the AOCE template mechanism for extending the Finder. In addition, there are two high-level AOCE APIs that make it very easy to add mail and catalog services to existing applications: the Standard Mail Package and the Standard Catalog Package.

To allow the AOCE system software to work with external databases and messaging systems, the AOCE technology also includes interfaces for catalog service access modules (CSAMs) and mail and messaging service access modules (MSAMs).

Figure 1-3 shows all of the components of the AOCE software.

Figure 1-3 The components of the AOCE software



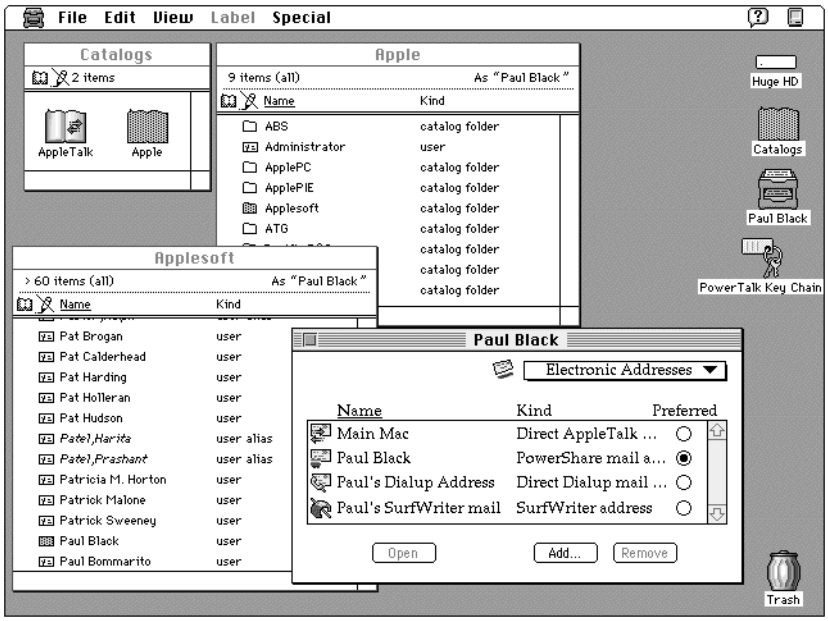
Desktop Services

The PowerTalk system software includes extensions to the desktop services offered by System 7. The user adds and configures mail and catalog services through the PowerTalk Key Chain. If you provide a CSAM or MSAM, you will also have to provide resources for use by the Key Chain as described in the chapter “Service Access Module Setup” in *Inside Macintosh: AOCE Service Access Modules*. All of the forms of electronic mail coming in from the messaging systems that the user has added to his or her Key Chain appear in the compound mailbox, which appears on the user’s desktop as the Mailbox icon. There is no application interface to the mailbox. To learn how to interface an external mail or messaging system to the PowerTalk messaging system, see *Inside Macintosh: AOCE Service Access Modules*.

The AOCE Catalogs Extension (CE) to the Finder includes the Catalog Browser and AOCE templates. The **Catalog Browser** is a Finder extension that allows a user to search through an AOCE catalog by opening folders on the desktop. An **AOCE catalog** is a hierarchically arranged store of data. AOCE catalogs include PowerTalk server-based catalogs, personal catalogs, and external databases interfaced to the AOCE catalog system through a catalog service access module (CSAM). Personal catalogs are stored as hierarchical file system (HFS) files on the user’s computer. AOCE catalogs are described in more detail in “Catalogs, Records, and Attributes” on page 1-14.

Viewed through the catalog browser, an AOCE catalog appears to be a folder, analogous to a volume in HFS, that contains *catalog folders*, analogous to HFS folders, and *records*, analogous to HFS files. Each record can contain one or more blocks of data, called *attribute values*. Figure 1-4 shows a desktop with catalogs, catalog folders, and record icons displayed.

Figure 1-4 The Catalogs Extension to the Finder in use on a desktop



Just as with HFS files, the user can open an AOCE catalog record to view its contents. Unlike HFS, however, which calls on the application that created the file to open it, it's the Finder itself that opens a catalog record. When the user opens a record on the desktop, the Catalogs Extension (CE) to the Finder displays a special window called an *information page window*. The information page window for a record might contain a list of attribute values; if so, the user can open each attribute value in the list to see what it contains, displayed in another information page window.

An information page window contains one or more **information pages**, each of which contains information about a record or about one of its attributes. This information can include a list of attributes in a record, information derived from attributes, and controls such as buttons and checkboxes. Information pages can also contain editable text and pictures. When the user makes changes to the information in an information page and closes the information page, the CE makes corresponding changes to the attribute values in the record. Figure 1-2 on page 1-6 shows an information page window.

Because there are no restrictions on the type of data that you can put in a record, the AOCE software includes a mechanism for designing and implementing new information pages to display new types of data (or for displaying old types of data in new ways). To create new information pages and to interface them with AOCE catalog records and attributes, you write resource files known as **AOCE templates**.

In the example in "The Company Store Catalog" beginning on page 1-5, AOCE templates are used extensively by the company store catalog to display the information in the store's database. On the user's desktop, the store's database appears as an AOCE catalog folder, containing a number of catalog folders with labels such as "Rubber ducks," "Other bath toys," and "Recording albums." Each such folder contains one

AOCE catalog record for each item in the company store catalog; for example, the "Rubber ducks" folder contains a little over 40 records, each containing information about a different variety of rubber duck.

When an employee opens a catalog-item record, an information page displays information that is stored as attributes in the record: a picture of the item, its price, the number in stock, and so forth. The order-form information page contains text-entry fields and controls that let the employee order the item. For each of these information pages, the company's programmers have supplied a set of AOCE templates to define the contents and appearance of the information page and to implement the functions that it performs.

For detailed information about AOCE templates, see the chapter "AOCE Templates" in this book. For more information about the structure of AOCE catalogs, see the chapter "Catalog Manager" in this book.

Collaboration Package

The AOCE Collaboration Package consists of two high-level APIs: the Standard Mail Package and the Standard Catalog Package. The Standard Mail Package provides two interfaces: the send-letter routines and the mailer routines.

Standard Mail Package

The send-letter routines send a file created by an application to a user's mailbox. In the example in "Purchasing" beginning on page 1-7, the forms application uses the send-letter routines to send purchase-order status reports to the originator of the purchase order and to the purchasing department.

The mailer routines allow you to add a mailer to any of your application's documents. A mailer appears as a new region at the top of your document's window (or you can put one in its own window, if you wish) that provides addressing and subject fields. The user can also add a digital signature to a letter containing a mailer. Adding a mailer to a document turns the document into a letter. The AOCE software delivers a letter you mail this way into the recipients' PowerTalk mailboxes. When the recipient double-clicks the letter, the Finder opens the application that was used to create it. If the recipient does not have that application, and if you chose to send an image version or a standard AppleMail format version of the document, the user can still open the document using the AppleMail application.

In the example, the president of the River Change Systems company uses a SurfWriter application document with a PowerTalk mailer to send a memo to all department heads, and the company store uses this application to send announcements to employees. In both cases, the letter includes an image or AppleMail format version so that recipients who do not have SurfWriter can still open it.

Standard Catalog Package

The Standard Catalog Package makes it easy for you to provide dialog boxes that let the user browse and search PowerTalk catalogs from within your application. In the example in "Purchasing" on page 1-7, the forms application provides a dialog box to the user to allow the selection of a form from a PowerTalk catalog that contains forms. Because you can restrict the dialog box to display only records of specific types, the catalog can contain many types of records in addition to those containing forms, but the forms application presents only the selection of forms to the user.

Digital Signature Manager

The Digital Signature Manager allows you to add a digital signature to a file or any portion of a file. A **digital signature** is an encrypted number that is associated with a particular set of data. The digital signature guarantees the identity of the individual or entity (for example, the company) that signed the data, and it ensures the integrity of the data. A digital signature cannot be forged, and the signed data can not be altered without invalidating the signature.

Users can use a mailer to add a digital signature to a document in an application that uses the Standard Mail Package to add mailers to its documents. They can also use the DigiSign utility program to sign any file. You can use the Digital Signature Manager to allow users to sign any data. In the example in "The Company Store Catalog" beginning on page 1-5, the AOCE template that provides data for the order-form information page calls the Digital Signature Manager to sign the order form to authorize a credit card purchase or payroll deduction. In "Purchasing" on page 1-7, the forms application requires the person submitting, approving, or disapproving the purchase order to add a digital signature. Because the signature is being added from within the application, the application itself must call the Digital Signature Manager.

Collaboration Toolbox

The fundamental services of AOCE are handled by three system software managers: the Authentication Manager, the Catalog Manager, and the Interprogram Messaging (IPM) Manager. Although you can call each of these managers independently to make use of its services, they work together to support a variety of server-based and workstation-based collaboration services. Together with the Digital Signature Manager, these managers are referred to in this book as the **AOCE toolbox**. The Authentication, Catalog, and IPM Managers are sometimes referred to as the **Collaboration toolbox**.

Authentication Manager

The Authentication Manager provides services to the other Collaboration toolbox managers and to the PowerShare servers. The Authentication Manager allows each end of a communications connection to determine that the other end is who it claims to be. Every time a user provides a password to log on to PowerTalk or to connect to a

PowerShare collaboration server, the Authentication Manager provides credentials to the AOCE software that verify the identity of the user.

Very few applications need to call the Authentication Manager directly. However, if you want to authenticate the connection ends of an AppleTalk Secure Data Stream Protocol (ASDSP) connection or if you want to authenticate some other type of connection outside of the AOCE messaging service, you can use Authentication Manager functions to do so. In any case, it is important that you understand the concepts of authentication and the credentials (known as *identities*) returned to your program by the Authentication Manager; these concepts are discussed in "Authentication and Authentication Identities" beginning on page 1-17. ASDSP is described in the chapter on ADSP in *Inside Macintosh: Networking*.

Catalog Manager

The AOCE Catalog Manager provides an interface to AOCE catalogs (see "Desktop Services" on page 1-9 for a general description of AOCE catalogs). The AOCE Catalog Manager lets you

- get information about AOCE catalogs and catalog nodes
- create, open, and close personal catalogs
- manage the organization of an AOCE catalog
- manage the contents of an AOCE catalog
- get information about access controls for a catalog and the contents of a catalog

The AOCE Finder extension allows users to browse catalogs, and the Standard Catalog Package lets you display dialog boxes that let the user browse catalogs and search for specific records. If you want to get information from a catalog or make changes to a catalog directly from your application or without user interaction, you can use the functions provided by the Catalog Manager.

In the example "Purchasing" on page 1-7, the forms application uses the Catalog Manager to find in a PowerShare catalog the name and address of the next person to whom the form should be routed. No user interaction is therefore necessary to route the form.

Interprogram Messaging Manager

The Interprogram Messaging (IPM) Manager provides a messaging service that is used by the AOCE software and that you can use for your own purposes. The IPM Manager works with servers (such as the PowerShare mail servers) and without servers. It can send messages over an AppleTalk network, over other networks (through mail and messaging service access modules, or MSAMs), and even over telephone lines through modems. The IPM Manager provides *store-and-forward messaging*. In store-and-forward messaging, the receiver does not have to be available to receive the message at the time it is sent; the IPM Manager stores the message and sends it on when the receiver is available.

Whereas you can use the Standard Mail Package to add mail capabilities to your application and to send any file as an attachment to a letter, you can use the IPM Manager to send mail or non-mail messages. The distinction between mail and non-mail messages is that mail involves the transmission of a message that is intended to be read by a person, whereas non-mail messages are intended to be used by an application.

In the example "The Company Store Catalog" beginning on page 1-5, when the employee clicks the Send button on the order form, the code resource associated with the information page uses the IPM Manager to send the signed order form to the company store and to send a message to the company's accounting database informing it of the sale, the amount due, and the form of payment. The signed order form is intended to be read by people and so is sent as electronic mail, but the message to the accounting database contains commands intended to be used only by the database application and so is sent as a non-mail message.

Service Access Modules

Developers can provide software modules, called service access modules (SAMs) that link the AOCE Collaboration toolbox to external mail and messaging services or databases. Two types of mail and messaging SAMs can be written: server-based MSAMs and personal MSAMs. A server-based MSAM resides on the same computer as a PowerShare mail server and makes an external mail or messaging system available to all users of that PowerShare server. A personal MSAM resides on an individual user's computer and makes an external mail or messaging system available through the AOCE software for only that user. In either case, the user sends and receives mail from the external system through the PowerTalk mailbox, and applications that send or receive AOCE messages can send and receive these messages via the external messaging system.

All catalog service access modules (CSAMs) are in the form of device drivers located on the user's computer. A CSAM allows a non-AOCE database to appear to users and applications to be an AOCE catalog. In the example "The Company Store Catalog" beginning on page 1-5, each user of the company catalog has a CSAM that interfaces the SurfDB database server to that user's PowerTalk system.

AOCE Concepts

This section describes some of the basic concepts and structures that appear frequently in this book. All of these topics are discussed in more detail in the succeeding chapters in this book, but they are all crucial to an understanding of how the AOCE software works and how you can use it, and you might find an early introduction to these topics helpful.

Catalogs, Records, and Attributes

As discussed in "Desktop Services" beginning on page 1-9, an AOCE catalog is a hierarchically arranged store of data. The root level of a catalog appears on the desktop

as a folder with the name of the catalog. The root-level catalog folder can contain any number of folders (also known as catalog nodes or **dNodes**) and records, and each contained folder can itself contain folders and records. (Note that in some AOCE catalogs, such as personal catalogs, the root-level folder contains only records, not other folders.) The catalog records contain the data and take the place in the catalog hierarchy occupied by files in the hierarchical file system (HFS). Accordingly, records cannot contain folders or other records.

The data in an AOCE catalog record is stored as **attribute values,** which cannot exceed 65,536 bytes in size. Each attribute value has an **attribute value tag,** which specifies the format of the data in the attribute value. Attribute values are grouped according to **attribute type.** When you add an attribute value to a record, the Catalog Manager assigns it an **attribute creation ID,** which is unique within the record. The combination of an attribute type, attribute value, attribute value tag, and attribute creation ID is referred to as an **attribute.**

Although an attribute value is the smallest unit of data storable in a catalog, an AOCE template can parse the data in an attribute value and display any portion of that data in a manner meaningful to the user. When the user edits the data in an information page and closes the information page, the CE uses the data-parsing pattern in the template to determine how to update the attribute value and saves the new value in the record.

You can call Catalog Manager functions to create and delete records and to add, modify, and delete attributes within records. You can call Standard Catalog Package functions to provide dialog boxes that allow users to browse and search catalogs. You can write AOCE templates that tell the Finder how to display the contents of records and attributes.

Messaging and Message Queues

The AOCE Interprogram Messaging Manager provides store-and-forward messaging; that is, when you send a message, the IPM Manager stores the message until it is able to forward it to its destination. Although PowerShare mail servers and other AOCE-compatible messaging servers can store messages on the server computer so that it is never necessary for the sending computer and receiving computer to be online simultaneously, the IPM Manager also provides a store-and-forward messaging service for messages not sent through servers. To achieve this end, the IPM Manager maintains an output queue for messages on the computer of the sending application and an input queue on the computer of the receiving application.

Suppose, for example, that you used a PowerTalk mailer to send a SurfWriter application document to someone at an AppleTalk address. If the recipient's computer was not connected to the network at the time you sent the message, the IPM Manager would place the message in your output queue. The IPM Manager would then check AppleTalk periodically to determine whether the recipient computer was present on the network. When the recipient's computer comes online, the IPM Manager in the sender's computer forwards the letter, and the IPM Manager in the recipient's computer places it in the input queue. Because in this example the message is a letter, it appears in the recipient's mailbox. Note that it is not necessary for the application used to create the message—

SurfWriter in this case—to be running on either computer at the time the IPM Manager forwards the letter from the sender's output queue to the recipient's input queue.

The IPM Manager provides default input queues for mail and for non-mail messages. In addition, your application can create additional input queues on the local computer for its own purposes. Managing such queues is somewhat complicated, however, and the AOCE software provides no protocol for determining the name of a nonstandard input queue; thus, you should create your own queues only when you have a clear need to do so.

The administrator of an AOCE messaging server can create additional input queues on the server computer and typically creates one queue for each user who has an account on that server.

There is no facility for creating additional output queues; the IPM Manager opens and maintains a single output queue on each user's computer for all mail and messaging purposes.

Because several applications might be using the same default input mail and messaging queues for different purposes, the IPM Manager provides the ability to filter the message list by a variety of criteria when you enumerate a queue.

Queues are described in detail in the chapter "Interprogram Messaging Manager" in this book.

Addressing Mail and Messages

Each AOCE message or letter must be delivered to a specific input queue. In many cases, the AOCE software takes care of addressing the message or letter for you, as when the user drags an address from a catalog and drops it in a mailer. In other cases, an AOCE function returns an address to which you can send a message or letter. For example, you can use the Standard Catalog Package to let the user select a user record. The Standard Catalog Package returns to you the catalog specifier (`DSSpec` structure) that identifies the record the user selects. You can then provide that catalog specifier to the IPM Manager as the address for a message. The IPM Manager extracts the name and address of the default messaging queue for that user from the specified record and routes the message accordingly.

This last example illustrates the use of *indirect addressing*. Indirect addressing lets you specify the entity (the user or application, for example) to which you want the message sent without specifying— or even having to know—the actual queue name or the location of the queue. You do this by specifying a catalog record (and, optionally, the attribute within that record) that contains the address. (If you use the user record type and do not specify the attribute, the IPM Manager uses the default mail or messaging queue, as in the preceding example.)

You can also use indirect addressing to send a letter or message to a group. In the simplest use of group addresses, the record you specify as an address is a group record containing the record identifiers of the user records of the members of the group. The AOCE software takes care of resolving the addresses and routing the message. You can

define your own record type in which to store group members' addresses and resolve group addresses yourself, but that requires a great deal more work on your part.

You can also use *direct addressing*, specifying the actual queue name plus the AppleTalk address, modem string, or other exact information needed by the IPM Manager to deliver the message. To use direct addressing, you must have prior knowledge of the queue name and location of the recipient, or you must develop your own protocol, outside of the AOCE APIs, for determining this information.

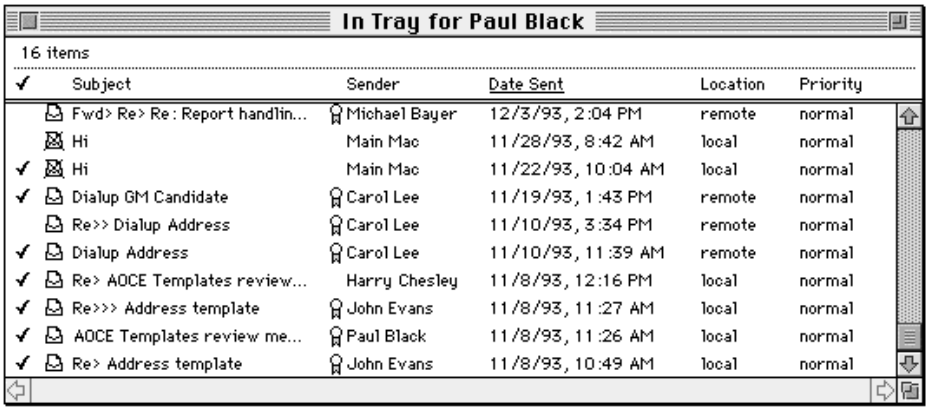
Addressing is discussed in detail in the chapter "Interprogram Messaging Manager" in this book.

Authentication and Authentication Identities

The AOCE Authentication Manager works together with the PowerShare collaboration servers to verify the identity of the requestor of a messaging or catalog service. Each AOCE message header includes an authentication field that you can use to determine whether the message or letter has been authenticated. If the value of this field is `true`, you can have a very high level of confidence that the name in the sender field is genuine.

In order for a message (or letter) to be authenticated, the sender of the message and every PowerShare server used to route the message has to have provided a valid password when logging on to the AOCE system. In addition, the recipient of the message has to provide a password before opening the mailbox or reading messages from an input queue. In the case of mail, the user can determine whether a letter is authenticated by looking for a seal next to the name of the sender in the In Tray. In Figure 1-5, for example, the letter titled "Dialup Address" is authenticated, whereas the letter titled "Re> AOCE Templates review..." is not.

Figure 1-5 An In Tray with certified (that is, authenticated) and uncertified letters



Note that only messages sent entirely through PowerShare servers can be authenticated by the AOCE system; messages sent through MSAM servers or by way of a serverless connection (over telephone lines, for example) cannot be authenticated by the AOCE

software. However, you can use Authentication Manager functions to implement your own authentication system for such messages. Note also that authentication guarantees only the identity of the sender, not the integrity of the data in the message itself. To guarantee that the content of a message has not been altered, use the Digital Signature Manager (see “Digital Signature Manager” on page 1-12).

When an entity (user, server, or application) provides a valid name and password to the Authentication Manager, the Authentication Manager returns a number known as an **authentication identity**. Each entity requesting a service from the IPM Manager, Catalog Manager, Standard Mail Package, or Standard Catalog Package then has to provide the authentication identity each time it requests a messaging, mail, or catalog service.

The IPM Manager uses the authentication identity to authenticate a message or letter. The Standard Mail Package goes a step further, using the authentication identity to determine the sender of a letter and to fill in the From field in the mailer. The Catalog Manager and Standard Catalog Package use the authentication identity for a different purpose entirely; they use the identity to determine whether the requestor of a catalog service has the authority to make a specific request. For example, the user who “owns” a user record (that is, the user to whom that record was assigned by the catalog administrator) normally has the authority to change personal data within that record. Your application, by providing that user’s authentication identity to the Catalog Manager, could then alter attribute values in that user record on behalf of that user. You could not, however, alter data in some other person’s user record, because the Catalog Manager would not grant you access to do so based on the authentication identity you supplied.

The Catalog Manager provides access controls for catalogs, dNodes, records, and attribute types. Access controls are described in detail in the chapter “Catalog Manager” in this book.

If you have used the PowerTalk software, you are probably already familiar with the concept of the PowerTalk Key Chain. Each time the user adds a service to the Key Chain, the AOCE software saves in a special personal catalog an encrypted form of the name and password the user provides. In this book this special catalog is referred to as the *PowerTalk Setup catalog*. The AOCE Authentication Manager takes the user’s name and password and creates a number called the **local identity**. In most cases, it is the local identity that you provide to an AOCE toolbox function as the authentication identity when requesting a service.

Suppose a user has accounts on two different PowerShare collaboration servers. The user adds the name and password for each of these accounts to the PowerTalk Key Chain. When the user turns on the computer in the morning and opens the PowerTalk mailbox, the PowerTalk software requests the user’s access code, which is the password that provides access to the Setup catalog. The user’s PowerTalk software then uses the information in the Setup catalog to obtain authentication identities, referred to in this book as **specific identities**, to obtain services from the PowerShare servers listed in the Key Chain.

When you need to provide an authentication identity to an AOCE function, you can provide either a local identity or a specific identity. When you provide the local identity,

the AOCE software uses the information in the Setup catalog to obtain the specific identity it needs to carry out the request. In most cases, you should provide the local identity to any AOCE function that requires an authentication identity. You should provide a specific identity only when it is absolutely necessary to do so, such as when you are using a background interapplication messaging service that is not visible to the user (and that has therefore not been added to the Key Chain), or when the user is not the owner of the computer and does not know the owner's access code.

The `SDPPromptForID` function (described in the chapter "Standard Catalog Package" in this book) displays a dialog box requesting a name and password and returns an authentication identity (either local or specific). If the user has already logged on, you can use the `AuthGetLocalIdentity` function (see the chapter "Authentication Manager" in this book) to obtain the local identity.

CHAPTER 2

AOCE Utilities

Contents

This chapter describes those data structures and utility functions that are used throughout the Apple Open Collaborative Environment (AOCE) but are not specific to any one particular manager or package. It describes the data structures you need to be familiar with to use the AOCE toolbox functions and shows you how to use the AOCE utility functions to manipulate these data structures in various ways.

You should read this chapter if you will be using the Standard Mail or Standard Catalog Packages to add AOCE services to your application, are developing a stand-alone mail or communications package that will use AOCE services, or if you are developing lower-level AOCE entities such as catalog services access modules.

Before reading this chapter you should have at least a general understanding of the Apple Open Collaborative Environment. At the minimum, you should have read the chapter "Introduction to AOCE," earlier in this book, which explains the organization and use of the various AOCE managers and services.

About the AOCE Utilities

The AOCE toolbox contains over 60 utility functions that are designed to provide you with easy methods for performing various tasks using the AOCE data structures. Here are some of the services that the AOCE utility functions provide:

- converting data structures to their packed forms (packing)
- converting data structures from their packed forms (unpacking)
- checking data structures to verify that they are in the proper format and contain valid data for their particular type
- comparing data structures for equality
- copying the contents of one data structure to another
- converting variables from one data type to another
- determining the size of data structures
- determining whether a given data structure is null or empty

Unless otherwise noted, all of the AOCE utility functions described in this chapter can be called at interrupt level and do not allocate any memory.

AOCE Data Structures of Maximum and Minimum Size

Some of the AOCE data structures are defined as maximum- or minimum-sized structures. A maximum-sized structure is one that, upon creation, contains enough storage to hold the maximum amount of data possible for that particular type of data structure. An example of a maximum-sized AOCE structure is the `RString` structure shown here and defined on page 2-20.

```
struct RString
{
   RStringHeader
   Byte  body[kRStringMaxBytes];
};
```

When you create a new `RString` structure, it contains enough memory to hold 256 bytes of data in its `body` field, plus the number of bytes necessary for the `RStringHeader` field. You never need to allocate any additional memory for the structure.

By contrast, a minimum-sized structure is one that, upon creation, contains only the minimum necessary storage. The minimum storage varies according to the type of data structure. An example of a minimum-sized structure is the `ProtoRString` structure shown here and defined on page 2-22.

```
struct ProtoRString
{
   RStringHeader
};
```

As you can see, the `ProtoRString` structure differs from the `RString` structure in that it does not contain a `body` field. Therefore, when you create a `ProtoRString` structure for the first time, it contains only enough memory to hold the information in its `RStringHeader` field. If you want to store any additional data in the `ProtoRString` structure, you will have to allocate the memory. See the section "Allocating AOCE Strings of Nonstandard Sizes" on page 2-16 for details on how to allocate additional memory for a `ProtoRString` structure.

The advantage of using minimum-sized AOCE data structures is that you can allocate structures of any size and can save memory by allocating structures that are exactly the size you need. The disadvantage of using minimum-sized AOCE data structures is that you will have to remember to allocate additional storage for the structure as you need it, and you will have to write more code to allocate each structure.

After declaring a variable as a minimum-sized AOCE structure, you may sometimes find that you need to allocate it as a maximum-sized structure. See the section "Allocating a RecordID Structure of Maximum Size" on page 2-16 for more information.

Using the AOCE Utilities

This section describes how you can use various AOCE utility functions and data structures in your own code. Many of the AOCE utility functions have similar characteristics and can be grouped according to the type of operations they perform. This section explains most of the major groups of AOCE utility functions and provides you with background knowledge that may help you understand how to use these functions.

Determining Whether the Collaboration Toolbox Is Available

Before calling any of the AOCE Utility functions, you 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” of *Inside Macintosh: Operating System Utilities*.

Packing and Unpacking the AOCE Data Structures

Several of the AOCE data structures contain fields that are themselves structures, and these may in turn contain other nested structures. It is sometimes useful to compact, or “flatten” a complex data structure into a sequence of bytes in order to perform an operation more efficiently. This process is known as **packing** the data structure. Similarly, the process of reconstructing a data structure from a sequence of bytes is known as **unpacking** the data structure.

Many of the AOCE functions pass packed structures. Because the packed forms of these structures are private, you can’t read or write them unless you use the utility routines to pack and unpack them.

Another reason for using the packed form of a data structure is to simplify I/O related tasks, such as writing the information contained in a data structure to a file, or sending the data to a serial port. In its packed form, the data is usually just a stream of bytes, which is much easier to work with in I/O operations.

The AOCE toolbox simplifies the processes of packing and unpacking by providing unpacked and packed forms of many of its data structures, as well as the utility functions to convert between the two forms. All of the AOCE packing functions begin with the letters `OCEPack` followed by the name of the data structure they pack, and all of the AOCE unpacking functions begin with the letters `OCEUnpack` followed by the name of the data structure they unpack. For example, the AOCE packing function that packs `RecordID` structures is named `OCEPackRecordID`.

Table 2-1 shows the AOCE data structures that have packed forms, along with the functions used to convert between the packed and unpacked forms.

Table 2-1 AOCE packed data structures and functions used to pack and unpack them

Unpacked data structure	Packed data structure	Packing/unpacking functions
`RString`	`PackedPathName`	`OCEUnpackPathName`
		`OCEPackPathName`
`RLI`	`PackedRLI`	`OCEPackRLI`
		`OCEUnpackRLI`
		`OCEPackedRLIPartsSize`
		`OCEPackRLIParts`
`RecordID`	`PackedRecordID`	`OCEPackRecordID`
		`OCEUnpackRecordID`
`DSSpec`	`PackedDSSpec`	`OCEPackDSSpec`
		`OCEUnpackDSSpec`

Note
The `Rstring` structure is shown in Table 2-1 as the unpacked form of the `PackedPathName` structure. This is actually a special case because the unpacked form of the `PackedPathName` structure is an array of `RString` structures. See the description of the `PackedPathName` structure on page 2-29 for more information. To create a `PackedPathName` structure, you need to supply an array of `RString` structures to the `OCEPackPathName` function (page 2-60). u

Unpacking Catalog Specifications

The catalog services specification data structure, of data type `DSSpec`, is central to accessing information within PowerTalk. Unpacking a `PackedDSSpec` structure is the process of converting the sequence of bytes in a `PackedDSSpec` structure into the structure of a `DSSpec`. In its packed form, the `DSSpec` structure contains other data structures that are also packed, so you must unpack each component as well as the `PackedDSSpec` structure itself.

Listing 2-1 shows how to unpack a `DSSpec` structure completely into its component parts, including its nested packed structures.

1. First, allocate a `DSSpec` structure (`DSSpecDumpRecord`) in which to store the contents of the `PackedDSSpec` structure when you unpack it. Also declare Boolean variables to record whether the various parts of the structure are valid.

2. Call the Standard Catalog Package function `SDPGetPanelSelectionSize` to obtain the size of the `PackedDSSpec` structure and then allocate memory for it. Call `SDPGetPanelSelection` to retrieve the `PackedDSSpec` structure.
3. Call the `UnpackPackedDSSpec` function to unpack the `PackedDSSpec` structure. Pass the function a pointer to the `PackedDSSpec` structure to be unpacked and a pointer to the `DSSpec` structure to hold its component parts.
4. Call the `DoDisplayDSSpecDumpRecord` function (which is not shown here) to use the information that you have retrieved from the `PackedDSSpec` structure; for example, to display the contents of a record that a user has selected.
5. It is possible that the `PackedDSSpec` structure you obtained from the `SDPGetPanelSelection` routine contains corrupted data. Therefore, you should check the integrity of the `PackedDSSpec` structure and of each of the nested packed structures that it contains before unpacking them. The `DoUnpackPackedDSSpec` function calls a series of AOCE utility functions to verify the integrity of the packed structures and to unpack them. The validation functions are nested in conditional statements. If any of the structures is invalid, the code prints an error message specifying which structure was corrupted. (The error messages are in the `else` statements at the end of Listing 2-1.)
   - The `OCEValidPackedDSSpec` and `OCEUnpackDSSpec` functions verify and unpack the packed `DSSpec` structure itself. The `OCEGetDSSpecInfo` function returns the type of the `DSSpec` structure.
   - The `OCEValidPackedRLI` and `OCEUnpackRLI` functions verify and unpack the `packedRLI` structure contained in the unpacked `DSSpec` structure.
   - The unpacked `RLI` structure contains a `PackedPathName` structure that you must unpack. However, before unpacking it, you call `OCEDNodeNameCount` to obtain the presumed number of pathnames. Then you allocate a vector to hold the `RString` structures that make up the pathname list. Finally, you call `OCEUnpackPathName` to unpack the `PackedPathName` buffer. If the presumed number of pathnames matches the actual number returned by `OCEUnpackPathName`, you are done.

Listing 2-1 Unpacking a DSSpec structure

```
/* In the example, the following external functions are defined:
   DoNOTE(message)           Write the message to the error log.
   DoFailOSErr(status, msg)  If status is not noErr, begin error recovery.
   DoFailNIL(ptr)            If ptr is nil, begin error recovery. This is
                            generally an unexpected, serious, error.

 The argument PackedDSSpec is stored in a private structure,
 DSSpecDumpRecord. Members of this structure contain pointers to the
 packed DSSpec.*/

typedef struct DSSpecDumpRecord {
  DSSpec          theDSSpec;         /* Unpacked DSSpec */
```

```
  RecordID        recordID;          /* Its record ID structure */
  RLI             theDSSpecRLI;      /* Its unpacked Record Location Info */
  OSType          specType;          /* The type of this DSSpec */
  unsigned short  nodeNameCount;     /* Presumed number of pathnames */
  unsigned short  trueNodeNameCount; /* Actual number of pathnames */
  RStringPtr      *partsVector;      /* -> vector of pathname RStrings */

/* These Boolean variables record the status of the DSSpec. They are true
if    the associated part of the structure is present and in good
condition. */

  Boolean     isValidDSSpec;         /* OCEValidPackedDSSpec succeeds */
  Boolean     isNonNullRLI;          /* RLI is present in this DSSpec */
  Boolean     isValidPackedRLI;      /* OCEValidPackedRLI succeeds */
  Boolean     isValidPackedPathName; /* OCEValidPackedPathName succeeds */
  Boolean     isValidUnpackedCount;  /* Unpacked count == presumed count */
} DSSpecDumpRecord, *DSSpecDumpPtr;

void
DoUnpackSDPPanelSelection(
        register DocumentPtr            dbp,
        SDPPanelHandle                  thePanel
    )
{
        OSErr                           status;
        PackedDSSpecPtr                 packedDSSpec;
        unsigned short                  packedDSSpecSize;
        DSSpecDumpRecord                dumpRecord;


/* Allocate memory for the DSSpec and get it from the Standard
   Directory Manager. */

        status = SDPGetPanelSelectionSize(thePanel, &DpackedDSSpecSize);
        DoFailOSErr(status, "\pSDPGetPanelSelectionSize");
        packedDSSpec = (PackedDSSpecPtr) NewPtrClear(packedDSSpecSize);
        DoFailNIL(packedDSSpec );
        status = SDPGetPanelSelection(thePanel, packedDSSpec);
        DoFailOSErr(status, "\pSDPGetPanelSelection");
        DoUnpackPackedDSSpec(packedDSSpec, &dumpRecord);
        DoDisplayDSSpecDumpRecord(&dumpRecord); /* Not shown */
        if (dumpRecord.partsVector != NULL)
            DisposePtr((Ptr) dumpRecord.partsVector);
        if (packedDSSpec != NULL)
```

```
            DisposePtr((Ptr) packedDSSpec);
}

void
DoUnpackPackedDSSpec(
        PackedDSSpecPtr                 packedDSSpec
        register DSSpecDumpPtr          theDSSpecDumpPtr
    )
{
#define SPEC (*theDSSpecDumpPtr)

        ClearMemory(&SPEC, sizeof SPEC);
        SPEC.isValidDSSpec = OCEValidPackedDSSpec(packedDSSpec);
        if (SPEC.isValidDSSpec) {
            OCEUnpackDSSpec(packedDSSpec, &SPEC.theDSSpec, &SPEC.recordID);
            SPEC.specType = OCEGetDSSpecInfo(&SPEC.theDSSpec);
            SPEC.isNonNullRLI = (SPEC.recordID.rli != NULL);
            if (SPEC.isNonNullRLI) {
                SPEC.isValidPackedRLI = OCEValidPackedRLI(SPEC.recordID.rli);
                if (SPEC.isValidPackedRLI) {
                    OCEUnpackRLI(SPEC.recordID.rli, &SPEC.theDSSpecRLI);
                    SPEC.isValidPackedPathName =
                        OCEValidPackedPathName(SPEC.theDSSpecRLI.path);

                     /* SPEC.isValidPackedPathName is false if you click
                        on a printer or CPU in the AppleTalk directory. */

                    if (SPEC.isValidPackedPathName) {
                        SPEC.nodeNameCount =
                           OCEDNodeNameCount(SPEC.theDSSpecRLI.path);

                         /* Allocate a vector to hold the RStrings that make
                            up the pathname list. Then unpack the pathname
                            list. */
                        SPEC.partsPtr = (RStringPtr *) NewPtrClear(
                                sizeof (RStringPtr) * SPEC.nodeNameCount
                            );
                        DoFailNIL(SPEC.partsPtr);
                        SPEC.trueNodeNameCount= OCEUnpackPathName(
                                SPEC.theDSSpecRLI.path,
                                SPEC.partsPtr,
                                SPEC.nodeNameCount
                            );
```

```
                if (SPEC.nodeNameCount == SPEC.trueNodeNameCount)
                    SPEC.isValidUnpackedCount = true;
                else {
                    NOTE("\pUnpacked Node Name Count != Node Name
                          Count");
                }
                else {
                    NOTE("\pInvalid PackedPathName");
                }
            }
            else {
                NOTE("\pInvalid Packed RLI");
            }
        }
        else {
            NOTE("\pValid DSSpec but NULL RLI");
        }
    }
    else {
        NOTE("\pInvalid Packed DSSpec");
    }
}
```

Validating the AOCE Data Structures

The AOCE toolbox provides a set of validation functions that allow you to verify the integrity of the various AOCE data structures. All of the AOCE validation functions begin with the letters "`OCEValid`" and are followed by the name of the data structure that they validate. For example, the AOCE validation function for `PackedDSSpec` structures is called `OCEValidPackedDSSpec`. Table 2-1 on page 2-6 shows the AOCE validation functions along with the data structures that each function validates. You should use the AOCE validation functions whenever you want to make sure that the AOCE data structures allocated in your program

- are valid values for that data type
- contain fields that have valid values
- are of a valid size
- contain fields of a valid size

The way the AOCE validation functions verify the integrity of a data structure depends upon the type of structure being examined. In general, however, AOCE validation functions perform the following checks:

- They determine whether the pointer to the data structure is `nil` or the data structure has a length of 0 and whether these are permissible values for this data structure.

- They determine if the data structure or any of its fields contain values that are not valid for that particular data structure.
- They determine if the value contained in any length fields of the data structure is equal to the number of bytes of data actually contained in that field.
- If the data structure contains fields that are other AOCE data structures, then the validation function passes these fields to other AOCE validation functions until all of the data structure's fields are checked. If the AOCE validation function cannot validate a field, it does not check that field but does check the rest of the data structure for validity.
- For packed data structures, the AOCE validation functions check that the packed data structure is at least as large or larger than the smallest possible packed structure of that type. This ensures that the data structure is at least large enough to hold the minimum amount of data in all of its fields.

Table 2-2 AOCE validation functions and associated data structures

Verify function name	Data structure verified
`OCEValidRString`	`RString`
`OCEValidPackedPathName`	`PackedPathName`
`OCEValidRLI`	`RLI`
`OCEValidPackedRLI`	`PackedRLI`
`OCEValidPackedRecordID`	`PackedRecordID`
`OCEValidPackedDSSpec`	`PackedDSSpec`

Listing 2-2 shows how to use the `OCEValidPackedPathName` function (page 2-62) to compare a `PackedPathName` structure for validity. This sample code calls the `OCEValidPackedPathName` function two different times to illustrate cases when the `PackedPathName` structure is valid and when it is not valid. The `MyValidatePackedPathName` function assumes the existence of a routine named `DoErrorChecking`, which handles any memory errors. For information on the `PackedPathName` structure see page 2-29.

Listing 2-2 Validating a `PackedPathName` structure

```
MyValidatePackedPathName()
{
   PackedPathName*      myPackedPathName;
   PackedPathName*      myNilPackedPathName;
   Boolean              isValid;       /* value returned by
                                          OCEValidPackedPathName/*
```

```
   /* First call OCEValidPackedPathName with a nil pointer. */
   myNilPackedPathName = nil;

   /* The AOCE toolbox does not consider nil PackedPathName
      pointers to be valid, so this call to OCEValidPackedPathName
      returns false in the isValid variable. */

   isValid = OCEValidPackedPathName(myNilPackedPathName);

   /* Allocate a PackedPathName structure. */
   myPackedPathName = (PackedPathName *)
                        NewPtr(sizeof(PackedPathName);

   DoErrorChecking();   /* make sure the PackedPathName allocation
                           didn't fail */

   myPackedPathName->dataLength = 0;/* set the length of the
                                       PackedPathName to 0 */

   /* The AOCE toolbox considers a PackedPathName with a length of
      0 to be valid, so this call to OCEValidPackedPathName
      returns true in the isValid variable. */

   isValid = OCEValidPackedPathName(myPackedPathName);
}
```

Comparing AOCE Data Structures for Equality

The AOCE toolbox provides a set of functions that allow you to compare the AOCE data structures for equality. All the AOCE equality functions begin with the letters `OCEEqual` and are followed by the type of the data structures being compared. For example, the AOCE equality function that compares two `RString` structures is called `OCEEqualRString`. The AOCE equality functions and the data structures that they compare are shown in Table 2-1 on page 2-6.

The actual method used to determine the equality of the data structures varies with their type. Before using any equality function, you should read its description to find out exactly how that function compares the data structures for equality. For example, the `OCEEqualPackedPathName` function (page 2-61) considers two `PackedPathName` structures to be equal if these three conditions are met: (a) one of the pointers passed into the function is `nil`, (b) the other pointer is not `nil`, and (c) the pointer that is not `nil` does point to a `PackedPathName` structure that has a length of 0. In general, each AOCE equality function acts as follows when comparing two structures for equality:

- If the data structures are packed, then the AOCE equality function unpacks them before comparing them. This has no effect on the original data structures.
- If the pointers to the data structures are both `nil`, then they are equal.

- If the data structures are not the same length, then they are not equal and no further comparisons are performed on them.
- If the data structures have fields that are other AOCE data structures, then the AOCE equality function compares these nested structures by calling the appropriate AOCE equality functions for these data structure types. This process is repeated for each nested data structure. If any of the nested structures are not equal, then the AOCE equality function returns `false`, indicating that the original data structures are not equal.

Table 2-3 AOCE equality functions and associated data structures

Equality Function Name	Data Structures Compared
`OCEEqualRString`	`RString`
`OCEEqualCreationID`	`CreationID`
`OCEEqualPackedPathName`	`PackedPathName`
`OCEEqualDirDiscriminator`	`DirDiscriminator`
`OCEEqualRLI`	`RLI`
`OCEEqualPackedRLI`	`PackedRLI`
`OCEEqualLocalRecordID`	`LocalRecordID`
`OCEEqualShortRecordID`	`ShortRecordID`
`OCEEqualRecordID`	`RecordID`
`OCEEqualPackedRecordID`	`PackedRecordID`
`OCEEqualDSSpec`	`DSSpec`
`OCEEqualPackedDSSpec`	`PackedDSSpec`

Copying AOCE Data Structures

The AOCE toolbox provides a set of functions for copying the contents of one AOCE data structure into another. You should use the AOCE copy functions whenever you want to copy the contents of one AOCE data structure into another.

None of the utility functions allocates any memory. Therefore, before you call an AOCE copy function, you need to make sure you have allocated both the source and destination structures. The AOCE copy function returns an error if the structures you allocate are too small. You should always check the value returned by an AOCE copy function to make sure that the copy took place successfully.

All of the AOCE copy functions begin with the letters `OCECopy` and are followed by the name of the data structure type that they copy. For example, the AOCE function for copying two `CreationID` structures is `OCECopyCreationID`. See Table 2-1 on page 2-6 for a list of the AOCE copy functions and the data structures that they copy.

Listing 2-3 illustrates the correct way to call an AOCE copy function. The `MyCopyingCode` function uses the `OCECopyRString` (page 2-45) utility routine to copy the `sourceRString` structure. The `sourceRString` structure is assumed to be a valid `RString` structure that has already been allocated and initialized elsewhere. The `MyCopyingCode` function also uses the Macintosh toolbox routine `MemErr` to check for memory allocation errors. In addition, the `myCopyingCode` function assumes the existence of a function named `DoErrorHandling` that handles an error if one occurs.

Listing 2-3 Calling a copy function

```
MyCopyingCode(RString*  sourceRString)
{
   /* This function assumes that the sourceRString parameter is
      a pointer to a valid RString containing data to be copied.
*/

   OSErr    myError;    /* this variable holds the value returned
                           by the OCECopyRString function */


   RString* destinationRString;  /* pointer to the RString that
                                    you want to copy the contents
                                    of sourceRString into */
   destinationRString = nil;     /* initialize the pointer to a
                                    "safe" value before
                                    continuing... */
   myError = noErr;              /* initialize error to none */

      /* Here is the correct way to call OCECopyRString. This
       code allocates the destinationRString variable to the
      correct size before calling the OCECopyRString function. */

   destinationRString = (RString *)NewPtr(sizeof(RString));
   /* Check if memory allocation failed by calling MemError
      Toolbox function. */
   if (MemError() != noErr)
   {
      /* There was an error. Call your error handler. */
      DoErrorHandling(myError);
   }
   /* Otherwise the RString was allocated properly. */
   myError = OCECopyRString(sourceRString, destinationRString);
   if (myError != noErr)
   {
```

```
      /* There was an error. Call your error handler. */
      DoErrorHandling(myError);
   }
}
```

Copying Versus Duplicating AOCE Data Structures

There is a single AOCE duplication function, `OCEDuplicateRLI`; it is used to duplicate `RLI` data structures. The difference between copying and duplicating as performed by AOCE toolbox functions is subtle but important. In this context, **copying** is taking the contents of each field in the source structure and placing them in the corresponding field of the destination structure. This process includes all nested structures as well.

However, some AOCE data structures, such as `RLI` structures, contain fields that are pointers to other nested data structures. For this reason, it is possible to change the pointers in the destination structure so that they point to the corresponding data structures in the source structure. This process of copying the pointers to data structures and not the actual data structures themselves, is called **duplicating** the data structures. This distinction between copying and duplicating applies only to the AOCE utility functions, and not to other APIs.

There are advantages and disadvantages to duplicating a data structure as opposed to copying it, and you must decide when it is appropriate to use duplication or copying in your own code. The advantage of duplicating a data structure is that it is much faster and requires less code than copying because only a pointer must be moved instead of a whole data structure.

The disadvantage of duplication is that you must keep both the source and destination structures in memory until you have finished using them. Here is the reason why: When you duplicate a structure, the pointers in the destination structure change to point to the source structure. Thus, after you duplicate a data structure, there is really only one copy of the data, but that data is pointed to by both the source and destination structures.

Table 2-4 AOCE copying and duplicating functions and associated data structures

Copying Function Name	Data Structure Copied
`OCECopyRString`	`RString`
`OCECopyCreationID`	`CreationID`
`OCECopyPackedPathName`	`PackedPathName`
`OCECopyDirDiscriminator`	`DirDiscriminator`
`OCECopyRLI`	`RLI`
`OCEDuplicateRLI`	`RLI`
`OCECopyPackedRLI`	`PackedRLI`

continued

Table 2-4 AOCE copying and duplicating functions and associated data structures (continued)

Copying Function Name	Data Structure Copied
OCECopyLocalRecordID	LocalRecordID
OCECopyShortRecordID	ShortRecordID
OCECopyRecordID	RecordID
OCECopyPackedRecordID	PackedRecordID
OCECopyPackedDSSpec	PackedDSSpec

Allocating AOCE Strings of Nonstandard Sizes

Three standard AOCE string sizes are defined for you by the RString, RString64, and RString32 structures. There are times, however, when you may wish to create an AOCE string of arbitrary size to store specialized data. Listing 2-4 shows how to accomplish this task. This example allocates an AOCE string that has a size of 23 bytes.

Listing 2-4 Allocating a string to store specialized data

```
ProtoRString *rstr;                    /* create a pointer to a
                                          ProtoRString struct */

rstr = NewPtr(23+sizeof(RStringHeader)); /* allocate memory for it
                                          including its header
                                          information */

if (rstr == nil)
{
   /* Then allocation has failed; not enough memory available.
      Put your error handling here. */
}
rstr->charSet = smRoman;            /* set script code to Roman */
rstr->length = 23;                  /* set the proper length */
```

Allocating a RecordID Structure of Maximum Size

When you allocate a new minimum-sized structure for the first time, memory is not automatically allocated for any of its fields except the header. There are times, however, when you may want to create a structure that has all of the memory for its fields allocated, thus ensuring that you have enough memory to hold a maximum-sized structure. For more information on minimum and maximum-sized AOCE structures, see “AOCE Data Structures of Maximum and Minimum Size” on page 2-3.

Listing 2-5 shows two functions, `MyAllocateMaxRID` and `MyDeallocateMaxRID`, which allocate and dispose of a maximum-sized `RecordID` structure. The `MyAllocateMaxRID` function uses the AOCE utility routine `OCESetCreationIDtoNULL` (page 2-54) to initialize the fields of a `CreationID` structure to `NULL` values. In addition, this function uses the Macintosh Toolbox routine `MemErr` to check for memory allocation errors.

Listing 2-5 Allocating and disposing of a maximum-sized `RecordID` structure

```
/* This function allocates a maximum-sized recordID structure*/

OSErr MyAllocateMaxRID(RecordID *rid)
{
   OSErr err;                        /* The error, if any, returned
                                        by AllocateMaxRID */
   PackedRLIPtr rli;                 /* Pointer to a packed RLI */
   RString *name;                    /* The record name */
   RString *type;                    /* The record type */

   rid->local.recordName = nil;      /* Initialize the record */
   rid->local.recordType = nil;      /* name, type, and rli to */
   rid->rli = nil;                   /* nil */

   /* Now allocate memory for a maximum-sized RString to hold
      the record name. */

   name = (RString*) NewPtr(sizeof(RString));

   err = MemError();
   if (err == noErr)
      {
      /* Now allocate space for the RString to hold the
         record type. */

      type = (RString*) NewPtr(sizeof(RString));

      err = MemError();
      if (err == noErr)
         {
         /* Finally, allocate the memory for the packed RLI. */

         rli = (PackedRLIPtr) NewPtr(sizeof(PackedRLI));

         err = MemError();
```

```
         if (err == noErr)
            {
            /* Now that all storage has been allocated, assign
               it to its proper location. */

            rid->local.recordName = name;
            rid->local.recordType = type;
            rid->rli = rli;

            /* Set the RLI's length field to its maximum size */

            rli->length = kRLIMaxBytes;

            /* Set the name and type RString's length fields to
               their maximum size. */

            name->length = kRStringMaxBytes;
            type->length = kRStringMaxBytes;

            /* Now initialize the creation ID by setting it to
               NULL. */

            OCESetCreationIDtoNull(&(rid->local.cid));

            }
         }
      }
   if (err != noErr)    /* if there was an error during memory */
                           allocation, dispose of the record ID
                           and return the error to the caller */
   {
      MyDeallocateMaxRID(rid);/* call function described next */
   }

   return err;
}

/* This function deallocates a record ID whose fields were
   allocated on the heap. */

void MyDeallocateMaxRID(RecordID *rid)
{
   DisposPtr((Ptr) rid->local.recordName);
```

```
   DisposPtr((Ptr) rid->local.recordType);
   DisposPtr((Ptr) rid->rli);
}
```

AOCE Utilities Reference

This section describes the data structures that are used throughout the various AOCE managers and packages and the utility functions that manipulate these data structures.

AOCE Data Structures

The data types described in this chapter are used throughout AOCE and are not confined to a particular manager or package.

AOCE String Structures

The AOCE string structures are used by AOCE functions in place of standard Pascal strings because AOCE strings can handle international character sets that may consist of 2 bytes per character and because AOCE strings include the script code for the character set of the data they contain. Standard Pascal strings use only 1 byte per character. All of the AOCE string structures consist of an `RStringHeader` field and a `body` field. The `RStringHeader` field contains information about the AOCE string, such as its character set and length, whereas the `body` field holds the actual string contents.

RStringHeader

The header is the portion of each AOCE string that defines the particular qualities that apply to the string's contents. Each header contains the field `charSet`, which is used to specify the character set, or script code, corresponding to the script you should use to interpret the AOCE string. A script code represents a writing system for a human language, such as Roman, Kanji, or Arabic, and the `charSet` field is the same as the script code used by the Script Manager to specify a particular script. See *Inside Macintosh: Text* for more information about script codes and international character sets, as well as for a listing of defined script code constants.

The header is defined as follows:

```
#define RStringHeader      \
   CharacterSet charSet;   \
   unsigned short dataLength;

   typedef short CharacterSet;
```

Field descriptions

`charSet`	The character set that applies to the text contained in the `RString`.
`datalength`	The length, in bytes, of the `body` field of the `RString` structure, not including the header. Note that for 2-byte character sets, such as Kanji, the number of characters in the `RString` structure is half the number of bytes in the `body` field.

RString

The `RString` structure is the basis for most strings in AOCE, as well as for other AOCE data types such as the `DirectoryName`, `AttributeType`, and `NetworkSpec` structures. The maximum number of bytes in an `RString` structure is defined by the constant `kRStringMaxBytes`, and the maximum number of characters in an `RString` structure is defined by the constant `kRStringMaxChars`.

Because the `RString` structure is of maximum size, it is already large enough to hold any other valid `RString` structure when you allocate it. For a minimum-sized AOCE string structure, see the `ProtoRString` type on page 2-22. The `RString` structure is defined as follows:

```
struct RString
{
   RStringHeader
   Byte  body[kRStringMaxBytes];
};

typedef struct RString RString;
```

Field descriptions

`RStringHeader`	A header (described on page 2-19) which defines the character set information that applies to the text of the `RString` structure and specifies the length, in bytes, of the data in the `body` field of the `RString` structure.
`body`	An array containing the actual `RString` structure's characters. The array has a length of `kRStringMaxBytes` number of bytes and contains as many bytes of data as specified by the `dataLength` field of the header. The constant `kRStringMaxBytes` is equal to 256 bytes.

RString64

The `RString64` structure is identical to an `RString` structure, except that its maximum size is smaller. The `RString64` length is defined by the constant `kRString64Size`.

```
struct RString64
{
   RStringHeader
   Byte     body[kRString64Size];
};

typedef struct RString64 RString64;
```

Field descriptions

`RStringHeader` A header (described on page 2-19) which defines the character set information that applies to the text of the `RString64` structure and specifies the length, in bytes, of the data in the `body` field of the `RString64` structure.

`body` An array containing the actual `RString64` structure's characters. The array has a length of `kRString64Size` number of bytes and contains as many bytes of data as specified by the `dataLength` field of the header. The constant `kRString64Size` is equal to 64 bytes.

RString32

The `RString32` structure is identical to an `RString` structure, except that its maximum size is smaller. The `RString32` structure's length is defined by the constant `kRString32Size`.

```
struct RString32
{
   RStringHeader
   Byte     body[kRString32Size];
};

typedef struct RString32 RString32;
```

Field descriptions

`RStringHeader` A header (described on page 2-19) which defines the character set information that applies to the text of the `RString32` structure and specifies the length, in bytes, of the data in the `body` field of the `RString32` structure.

`body` An array containing the actual `RString32` structure's characters. The array has a length of `kRString32Size` number of bytes and contains as many bytes of data as specified by the `dataLength` field of the header. The constant `kRString32Size` is equal to 32 bytes.

ProtoRString

The ProtoRString is the only AOCE string structure of minimum size; it initially has no space allocated for the string contents. You should use a ProtoRString structure whenever you need to create an AOCE string of variable length.

```
struct ProtoRString
{
   RStringHeader
   /* Define the body of the ProtoRstring here. */
};

typedef struct ProtoRString ProtoRString;
```

Field descriptions

RStringHeader — A header (described on page 2-19) which defines the character set information that applies to the text of the RString structure and specifies the length, in bytes, of the data in the body field of the RString structure.

Note

The ProtoRString structure does not have a defined body field as do the other AOCE string structures. It is up to you to add a body field for the ProtoRString structure. See the section "Allocating AOCE Strings of Nonstandard Sizes" on page 2-16 for an example of how to do this. u

DirectoryName

A DirectoryName structure consists of a character set code, a length containing the number of bytes in the body field, and the data in the body field. A DirectoryName structure is identical to an RString structure, except that its maximum length is defined by the constant kDirectoryNameMaxBytes and its body field holds the name of a catalog (it is called a DirectoryName structure for historical reasons). You can typecast any DirectoryName structure to an RString structure and use the RString utility functions on it. The RString utility functions are described starting on page 2-45.

```
struct DirectoryName
{
   RStringHeader
   Byte     body[kDirectoryNameMaxBytes];
};

typedef struct DirectoryName DirectoryName;
```

Field descriptions

`RStringHeader` A header (described on page 2-19) which defines the character set information that applies to the text of the `RString` structure and specifies the length, in bytes, of the data in the `body` field of the `RString` structure.

`body` An array of characters that contains the name of a catalog. This array can contain up to `kDirectoryNameMaxBytes` number of bytes and contains as many bytes of data as specified by the `dataLength` field of the header. The constant `kDirectoryNameMaxBytes` is equal to 32 bytes.

NetworkSpec

A `NetworkSpec` structure consists of a character set code, a length containing the number of bytes of data, and the data itself. A `NetworkSpec` structure is identical to an `RString` structure, except that its maximum length is defined by the constant `kNetworkSpecMaxBytes` and its `body` field is used to hold the name of a network. You can typecast any `NetworkSpec` structure to an `RString` structure and use any of the `RString` utility functions on it. The `RString` utility functions are described starting on page 2-45.

For an example of how some functions use the `NetworkSpec` structure, see the `DirGetLocalNetworkSpec` and `DirGetDNodeInfo` functions in the chapter "Catalog Manager" in this book.

```
struct NetworkSpec
{
   RStringHeader
   Byte  body[kNetworkSpecMaxBytes];
};

typedef struct NetworkSpec NetworkSpec;
```

The `RStringHeader`, described on page 2-19, defines the character set information that applies to the text of the `RString` structure and specifies the length, in bytes, of the `body` field of the `RString` structure.

Field descriptions

`RStringHeader` A header (described on page 2-19) which defines the character set information that applies to the text of the `RString` structure and specifies the length, in bytes, of the data in the `body` field of the `RString` structure.

`body` An array of characters that contains the name of a network. This array can contain up to `kNetworkSpecMaxBytes` number of bytes and contains as many bytes of data as specified by the `dataLength` field of the header. The constant `kNetworkSpecMaxBytes` is equal to 32 bytes.

RStringKind

Some of the AOCE utility functions require a parameter of type `RStringKind` in addition to an AOCE string parameter. Based on the value of the parameter of type `RStringKind`, the routine determines how it will handle the `RString` structure. The `OCERelRString` (page 2-48), `OCEEqualRString` (page 2-50), and `OCEValidRString` (page 2-51) functions use the `RStringKind` data type. When you call one of these functions, you need to decide what value of the `RStringKind` type to use.

```
enum
{
   kOCEDirName =           0,
   kOCERecordOrDNodeName = 1,
   kOCERecordType =        2,
   kOCENetworkSpec =       3,
   kOCEAttrType =          4,
   kOCEGenericSensitive =  5,
   kOCEGenericInsensitive =6
};

typedef unsigned short RStringKind;
```

Field descriptions

`kOCEDirName` The AOCE string is a `DirectoryName` structure containing a catalog name. For more information about the `DirectoryName` structure see page 2-22.

`kOCERecordOrDNodeName`
The AOCE string is a `recordName` structure containing a record name or a catalog node name. See the `LocalRecordId` structure on page 2-27 for the definition of the `recordName` structure.

`kOCERecordType`
The AOCE string is a `recordType` structure containing a record type. See the `LocalRecordId` structure on page 2-27 for more information on the `recordType` structure.

`kOCENetworkSpec`
The AOCE string is a `NetworkSpec` structure containing a network specification. See page 2-23 for more information on the `NetworkSpec` structure.

`kOCEAttrType` The AOCE string is an `AttributeType` structure containing an attribute type. For more information on the `AttributeType` structure see page 2-39.

`kOCEGenericSensitive`
The AOCE string is a generic AOCE string type that you should use when you want an AOCE utility routine to be both case-sensitive and sensitive to diacritical marks in its treatment of an `RString` structure (c ≠ C ≠ ç). Use this type for your own AOCE strings that will not be seen by a user.

`kOCEGenericInsensitive`
The AOCE string is a generic AOCE string type that you should use when you want an AOCE utility routine to be neither case-sensitive nor sensitive to diacritical marks in its treatment of an `RString` structure (c = C = ç). Use this type for your own AOCE strings that will be seen by a user.

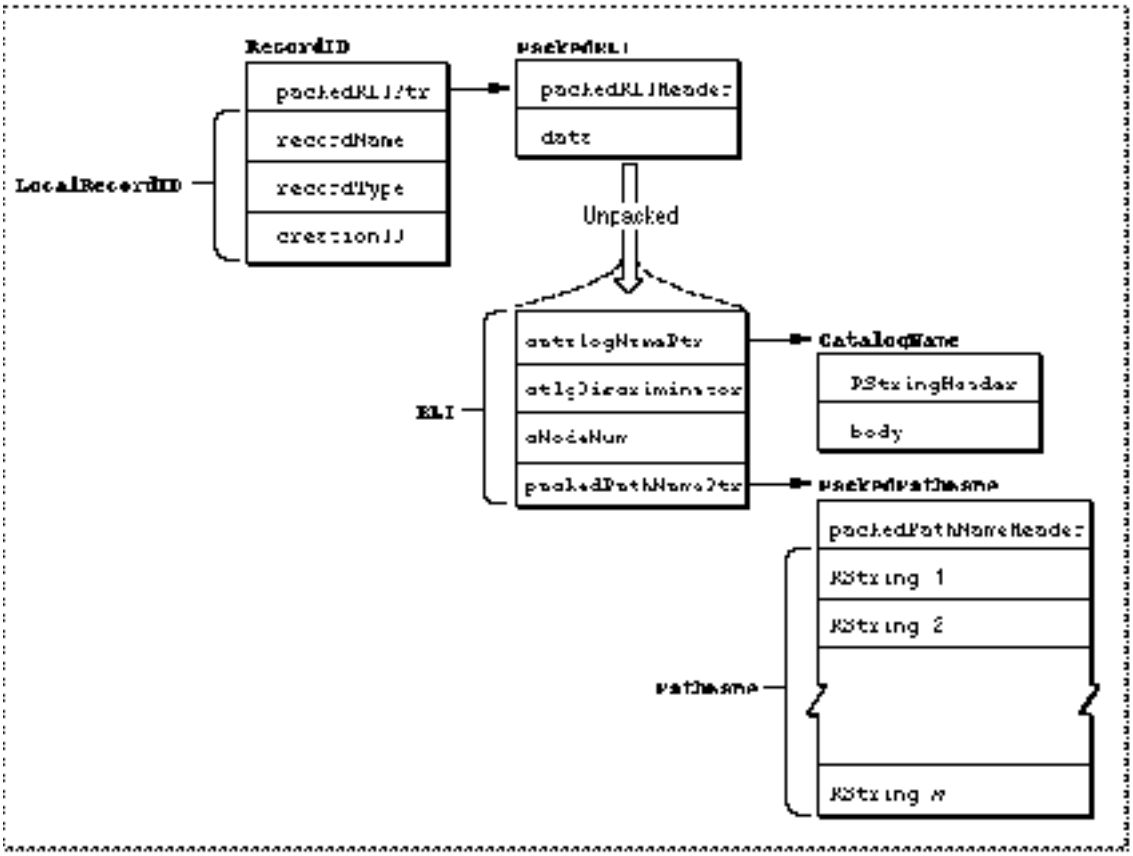
Note

You should use the `kOCEGenericSensitive` and `kOCEGenericInsensitive` `RStringKind` values when you use AOCE strings to hold data other than a catalog node name or the five derivative AOCE string structures (`DirectoryName`, `AttributeType`, `NetworkSpec`, `recordName`, and `recordType`). Do not use the `kOCEGenericSensitive` and `kOCEGenericInsensitive` `RStringKind` types with `DirectoryName`, `recordName`, `recordType`, `NetworkSpec`, or `AttributeType` structures or with catalog node names because this may cause the AOCE string to be treated incorrectly by the function you are calling. u

Record Identifier Structures

A record identifier structure uniquely identifies a record in an AOCE catalog. It consists of the name and discriminator value of the catalog, the catalog node number or the path information for the catalog node in which the record is located, and the record's name, type, and creation identifier. A record identifier is defined by the `RecordID` structure. Because the `RecordID` structure is composed of substructures (see Figure 2-1), many of which contain components of their own, the component structures of the `RecordID` structure are described first in this section.

Figure 2-1 The Record identifier structure



CreationID

The record creation identifier is defined by the `CreationID` structure and is used to uniquely identify a record within a PowerShare catalog or in a personal catalog. Some catalogs may not support the `CreationID` structure; they may rely on the uniqueness of a record's name and type to specify each record instead. The `CreationID` structure is a component of the `LocalRecordID` structure (page 2-27).

The fields of the `CreationID` structure are private to a catalog; you never need to know how to put data into a `CreationID` structure or how the data is represented inside the `CreationID` structure. Once you have allocated space for a new `CreationID` structure, you simply pass it into a function such as `DirAddRecord`, which fills the `CreationID` structure with the proper data for you. You then pass the `CreationID` structure along to other functions that require it, such as the `DirDeleteRecord` function. For more information on the `DirAddRecord` and `DirDeleteRecord` functions see the chapter "Catalog Manager" in this book.

AOCE defines two types of `CreationID` structures: the `CreationID` structure and the `AttributeCreationID` structure. These structures are identical but have different names to help distinguish the way in which they are used by various AOCE managers and functions. The `CreationID` structure is sometimes called the record `CreationID` structure to reinforce the idea that it is being used for a record, and not an attribute.

```
struct CreationID
{
   unsigned long  source;  /* private to a catalog.*/
   unsigned long  seq;     /* private to a catalog*/
};

typedef struct CreationID CreationID;

typedef CreationID AttributeCreationID;
```

LocalRecordID

A local record identifier uniquely identifies a record within a catalog. It contains the record's creation identifier, described in the previous section, and the record's name and type. The name and type can uniquely identify a record in an external catalog that does not support creation identifiers. The local record identifier is defined by the `LocalRecordID` structure.

The creation identifier field of the local record identifier is maintained by the catalog that contains the `LocalRecordID` structure. Whenever a record is created in a catalog that supports creation identifiers, the catalog assigns the record a new creation identifier that is unique within the catalog. This procedure prevents duplicate creation identifiers within the same catalog. Within a catalog that does not support creation identifiers, it is not possible to have two records with the same name and type, because the catalog uses the record's name and type to define a particular record uniquely.

The `LocalRecordID` structure is a component of the `RecordID` structure described on page 2-34, and is also a component of the `DirEnumSpec` structure, described in the chapter "Standard Catalog Package" in this book. See the `DirFindValue` function in the chapter "Catalog Manager" in this book for an example of a function that uses the `LocalRecordID` structure.

```
struct LocalRecordID
{
   CreationID  cid;              /* creation ID of the record */
   RStringPtr  recordName;       /* name of the record */
   RStringPtr  recordType;       /* type of record */
};
```

```
typedef struct LocalRecordID LocalRecordID;
typedef LocalRecordID *LocalRecordIDPtr;
```

Field descriptions

`cid`	The creation identifier of the record. If the creation identifier is not `NULL`, this number is unique within a catalog.
`recordName`	The name of the record. The name is not necessarily unique within a catalog.
`recordType`	The type of entity that the record represents. For example, the record could be of type `User`, `Group`, `LaserWriter`, and so forth. For a list of standard record types, see the `OCERecordTypeIndex` structure described next. The record type is not necessarily unique within a catalog.

OCERecordTypeIndex

The `OCERecordTypeIndex` is an enumerated list of the standard AOCE record types. You should use this list whenever you need to obtain a record type that has been defined by Apple Computer, Inc. All lowercase four-character combinations are reserved by Apple Computer, Inc., as well as all uppercase and lowercase combinations of the sequence 'AOCE'. To get a specific record type, call the `OCEGetIndRecordType` function and pass it the proper index constant from the `OCERecordTypeIndex` enumerated list. The `OCEGetIndRecordType` function returns a pointer to an `RString` structure that contains the proper record type corresponding to the index entry you supplied. See page 2-85 for the complete description of the `OCEGetIndRecordType` function.

```
enum /* OCERecordTypeIndex */
{
   kUserRecTypeNum =          1,    /* "User" */
   kGroupRecTypeNum =         2,    /* "Group" */
   kMnMRecTypeNum =           3,    /* "AppleMail™ M&M" */
   kMnMForwarderRecTypeNum =  4,    /* "AppleMail™ Fwdr" */
   kNetworkSpecRecTypeNum =   5,    /* "NetworkSpec" */
   kADAPServerRecTypeNum =    6,    /* "PowerShare Server" */
   kADAPDNodeRecTypeNum =     7,    /* "PowerShare DNode" */
   kADAPDNodeRepRecTypeNum =  8,    /* "PowerShare DNode Rep" */
   kServerSetupRecTypeNum =   9,    /* "Server Setup" */
   kDirectoryRecTypeNum =     10,   /* "Catalog" */
   kDNodeRecTypeNum =         11,   /* "DNode" */
   kSetupRecTypeNum =         12,   /* "Setup" */
   kMSAMRecTypeNum =          13,   /* "MSAM" */
   kDSAMRecTypeNum =          14,   /* "CSAM" */
   kAttributeValueRecTypeNum =15,   /* "Attribute Value" */
```

```
   kBusinessCardRecTypeNum =  16,   /* "Business Card" */
   kMailServiceRecTypeNum =   17,   /* "Mail Service" */
   kCombinedRecTypeNum =      18,   /* "Combined" */
   kOtherServiceRecTypeNum =  19,   /* "Other Service" */
   kAFPServiceRecTypeNum =    20    /* "Other Service afps" */
};

typedef unsigned short OCERecordTypeIndex;
```

In addition to the `OCERecordTypeIndex` values defined above, there are three more record type definitions:

```
#define kFirstOCERecTypeNum kUserRecTypeNum
            /* first standard AOCE record type */

#define kLastOCERecTypeNum kAFPServiceRecTypeNum
            /* last standard AOCE record type */

#define kNumOCERecTypes (kLastOCERecTypeNum -
                         kFirstOCERecTypeNum + 1)/* total number of
                                                    standard AOCE
                                                    record types */
```

You can use these three constants to enumerate all the standard AOCE record types.

PackedPathName

The `PackedPathName` structure contains the names of all of the catalog nodes in the path from the catalog node in which a record resides, to the root catalog node in the AOCE catalog tree. A `PackedPathName` structure is an array of `RString` structures, with each component `RString` structure containing the name of a catalog node on the path. You create a `PackedPathName` structure from an array of `RString` structures by using the `OCEPackPathName` function (page 2-60). You can also unpack a `PackedPathName` structure into its `RString` component parts by using the `OCEUnpackPathName` function (page 2-58). The maximum size of an entire packed pathname is defined by the constant `kPathNameMaxBytes`.

The `PackedPathName` structure's format is private, so you must always use the `OCEPackPathName` and `OCEUnpackPathName` functions to pack and unpack these structures. Do not assume you know the format of `PackedPathName` structures.

The `PackedPathName` structure is a component of the record location information structure (page 2-32). In addition, the AOCE Catalog Manager uses the packed pathname structure in various functions such as `DirMapDNodeToPathName` and `DirMapPathNameToDNode`. For information on these functions, see the chapter "Catalog Manager" in this book.

```
struct PackedPathName
{
   unsigned short  dataLength;       /* number of bytes in data
                                        field */
   Byte         data[kPathNameMaxBytes - sizeof (unsigned short)];
};

typedef struct PackedPathName PackedPathName;
```

Field descriptions

`dataLength` The number of bytes in the data field. This does not include the bytes in the dataLength field itself.

`data` A packed array containing the names of all of the catalog nodes in the path from the catalog node in which the record resides, to the catalog root node. Each of the names in the array is an `RString` structure.

ProtoPackedPathName

The `ProtoPackedPathName` structure is a minimum-sized structure. It is equivalent to a `PackedPathName` structure without a data field. You should use this data type whenever you need to create a `PackedPathName` structure of variable length.

```
struct ProtoPackedPathName {
   unsigned short dataLength;
/* Followed by data */
};

typedef struct ProtoPackedPathName ProtoPackedPathName;
```

Field descriptions

`dataLength` The length of the data field of the PackedPathName structure.

Note

You must create the data portion of the `ProtoPackedPathName` structure yourself. Since this is a minimum-sized structure, it initially has no `data` field, and hence no memory is allocated for any contents. See the section "Allocating AOCE Strings of Nonstandard Sizes" on page 2-16 for an example of how to allocate memory for a minimum-sized structure. u

DirDiscriminator

A catalog discriminator is defined by a `DirDiscriminator` structure and is used to differentiate between two or more catalogs that have the same name, as the combination of a catalog name and a `DirDiscriminator` structure uniquely identify a catalog. The `DirDiscriminator` structure contains two fields which are set by the catalog. An application does not need to set or change these fields. If you are creating a catalog server access module, you need to read the chapter "Catalog Service Access Modules" in *Inside Macintosh: AOCE Service Access Modules* for information on how to modify the fields of a `DirDiscriminator` structure.

In addition to being a component of the record location information structure, described next, the `DirDiscriminator` structure is used by several of the AOCE Catalog Manager functions. You also use a `DirDiscriminator` structure when you provide callback functions to such functions as `DirEnumerateDirectoriesParse` and `DirNetSearchADAPDirectoriesParse`. See the chapter "Catalog Manager" in this book for more information on these two functions.

```
struct DirDiscriminator {
   OCEDirectoryKind  signature;     /* type of a catalog */
   unsigned long     misc;          /* private to catalog */
};

typedef struct DirDiscriminator DirDiscriminator;
```

Field descriptions

`signature` Defined by the catalog provider. It may be, but is not required to be, the same as the application's signature. Apple Computer, Inc. has defined the following values for this field. Developers of catalog service access modules may define additional values.

`kDirAllKinds = 0`

`kDirADAPKind = 'adap'`

`kDirPersonalDirectoryKind = 'pdir'`

`kDirDSAMKind = 'dsam'`

`misc` Defined by the catalog provider. A catalog service access module may use it to distinguish between different catalogs that it supports. See the chapter "Catalog Service Access Modules" in *Inside Macintosh: AOCE Service Access Modules* for more information on this field.

RLI

The record location information structure identifies the catalog and catalog node in which a record resides. The record location information is defined by the `RLI` data type. The `RLI` structure is the unpacked form of the `PackedRLI` data structure, described next.

```
typedef unsigned long   DNodeNum;

struct RLI {
   DirectoryNamePtr     directoryName;
   DirDiscriminator     discriminator;
   DNodeNum             dNodeNumber;
   PackedPathNamePtr    path;
};

typedef struct RLI RLI;
typedef RLI *RLIPtr;
```

Field descriptions

`directoryName` A pointer to the name of the catalog in which the record resides. The maximum number of bytes in a catalog name is defined by the constant `kDirectoryNameMaxBytes`.

`discriminator` A value that allows you to distinguish between two or more catalogs that have the same name.

`dNodeNumber` A value that uniquely identifies the catalog node in which the record resides. Set this field to 0 or to `kNULLDNodeNumber` if you are using the `path` field to identify the catalog node.

`path` A pointer to a buffer that contains the names of all of the catalog nodes on the path from the catalog node in which the record resides, to the catalog root node. You should set this field to `nil` if you are using the `dNodeNumber` field to identify the catalog node.

The `directoryName` and `discriminator` fields of the `RLI` structure specify the catalog. The last two fields of the `RLI` structure, the `dNodeNumber` and `path` fields, specify a catalog node within the catalog specified by the `directoryName` and `discriminator` fields. For PowerShare catalogs, you must specify the catalog node by either a catalog node number or by a pathname, but not both.

Some catalogs may allow you to specify a catalog node using a partial pathname. A partial pathname is a combination of values in the `dNodeNumber` and `path` fields. To assure compatibility with all catalogs, you need to call the `DirGetDirectoryInfo` function to find out if the catalog supports the use of partial pathnames before providing a partial pathname to the catalog. If a catalog supports partial pathnames, you must set both the `dNodeNumber` and `path` fields to meaningful values, because both fields are used. If this is the case, and your application does not support partial pathnames, you should set either the `dNodeNumber` field to 0 or the `path` field to `nil`.

PackedRLI

The record location information in its packed form is defined by the PackedRLI data type. Use the OCEPackRLI function (page 2-71) to create a PackedRLI structure from an RLI structure or its component parts. Use the OCEUnpackRLI function (page 2-72) to unpack a PackedRLI structure into its component parts. The order of the data within a PackedRLI structure is private, so you must use the utility functions when creating and unpackingPackedRLI structures. This is the only way to be sure that the data will be in the correct format.

In addition to being a component of the RecordID data structure, described on page 2-34, the PackedRLI structure is used by several of the AOCE Catalog Manager functions.

```
#define kRLIMaxBytes (sizeof (RString) + \
                      sizeof (DirDiscriminator) + \
                      sizeof (DNodeNum) + kPathNameMaxBytes)
```

The constant kRLIMaxBytes is the maximum number of bytes that can be stored in the data field of a PackedRLI structure. This is large enough to hold the sum of RString, DirDiscriminator, and DNodeNum structures plus a maximum-length pathname.

```
struct PackedRLI {
   unsigned short dataLength;          /* length of data field */
   Byte           data[kRLIMaxBytes];  /* packed record
                                          location info */
};

typedef struct PackedRLI PackedRLI;
typedef PackedRLI *PacedPLIPtr;
```

Field descriptions

dataLength	The number of bytes in the data field of the PackedRLI structure. It does not include the number of bytes in the dataLength parameter itself.
data	A packed array of characters that contains the catalog name, the catalog discriminator, and the catalog node number or a pathname.

ProtoPackedRLI

The ProtoPackedRLI structure is a minimum-sized structure. It is equivalent to a PackedRLI structure without a data field. You should use this data type whenever you need to create a PackedRLI structure of variable length.

```
struct ProtoPackedRLI {
   unsigned short dataLength;          /* length of data */
/* Followed by data */
};

typedef struct ProtoPackedRLI ProtoPackedRLI;
typedef ProtoPackedRLI *ProtoPackedRLIPtr;
```

Field descriptions

`dataLength` The length of the `data` field of the `PackedRLI` structure.

Note

You must create the data portion of the `ProtoPackedRLI` structure yourself. Because this is a minimum-sized structure, it initially has no `data` field, and thus no memory is allocated for any contents. See the section "Allocating AOCE Strings of Nonstandard Sizes" on page 2-16 for an example of allocating memory for a minimum-sized structure. u

RecordID

Each record in an AOCE catalog is described by a `RecordID` structure. A `RecordID` structure consists of two parts: a local record identifier and a packed record location information structure. The local record identifier uniquely defines the record within its catalog. The packed record location information structure identifies the catalog and catalog node in which the record resides.

```
struct RecordID {
   PackedRLIPtr      rli;     /* identifies record's catalog
                                 and dNode */
   LocalRecordID     local;   /* identifies record within
                                 its dNode */
};

typedef struct RecordID RecordID;
typedef RecordID *RecordIDPtr;
```

Field descriptions

`rli` A pointer to a `PackedRLI` structure that identifies the catalog and the specific catalog node in which the record resides.

`local` A `LocalRecordID` structure that uniquely identifies the record within its catalog.

PackedRecordID

A packed record identifier is the packed form of a `RecordID` structure and is defined by the `PackedRecordID` structure. The packed form of the `RecordID` structure is useful when you wish to store data or transmit it because the `PackedRecordID` structure is a single block of data, rather than a structure containing pointers into other structures as the `RecordID` structure is. You use the `OCEPackRecordID` function (page 2-90) to create a `PackedRecordID` structure from a `RecordID` structure, and you use the `OCEUnpackRecordID` function (page 2-91) to convert a `PackedRecordID` structure into an unpacked `RecordID` structure.

```
#define kPackedRecordIDMaxBytes (kPathNameMaxBytes + \
         sizeof (DNodeNum) + sizeof (DirDiscriminator) + \
         sizeof (CreationID) + (3 * sizeof (RString)))
```

The constant `kPackedRecordIDMaxBytes` defines the maximum number of bytes that can be stored in the `data` field of a `PackedRecordID` structure.

```
struct PackedRecordID {
   unsigned short dataLength; /* length of data field
                                 in PackedRecordID */
   Byte     data[kPackedRecordIDMaxBytes];/* packed record ID */
};

typedef struct PackedRecordID PackedRecordID;
```

Field descriptions

`dataLength` The size of the `data` field of the `PackedRecordID` structure. It does not include the length of the `dataLength` parameter itself.

`data` An array containing the `RecordID` data.

ShortRecordID

A short record identifier structure is similar to a record identifier, except that it does not contain the `recordName` and `recordType` fields. For more information on record location information structures see page 2-32.

```
struct ShortRecordID
{
   PackedRLIPtr rli;
   CreationID cid;
};

typedef struct ShortRecordID ShortRecordID;
```

Field descriptions

`rli`	A pointer to a packed record location information structure.
`cid`	A pointer to a creation identifier structure.

Catalog Services Specification

The catalog services specification structures are used throughout AOCE for performing various tasks such as getting and setting access controls for records, obtaining the individual members of a group record that the user has selected, computing the size of a record currently selected by the user, specifying message addresses, and so forth. The catalog services specification is defined by the `DSSpec` structure and its packed form by the `PackedDSspec` structure. Other forms of the `DSSpec` structure include the `OCERecipient` and the packed form, `OCEPackedRecipient`, which are defined in the chapter "Interprogram Messaging Manager" in this book.

In addition to the above uses, you can also use the catalog services specification to hold your own types of data that may not have a specified size. In this case, use the `ProtoPackedDSspec` structure.

DSSpec

The catalog services specification structure is defined by the `DSSpec` data type. A `DSSpec` structure contains a pointer to a `RecordID` structure, plus additional information such as an extension type, extension size, and extension value. When you supply a `DSSpec` structure to a routine, you must provide a pointer to a record identifier in its `entitySpecifier` field. The other fields are optional, depending upon what data the `DSSpec` structure is being used to hold. For example, if the `DSSpec` structure has no extension, then it can represent either the root of all catalogs, a single catalog, a catalog node, or a record. If the `DSSpec` structure has an extension, then the `extensionType`, `extensionSize`, and `extensionValue` fields must contain valid values for the particular extension type. For more information on extension types and their allowable values, see the `OCEValidDSSpec` function on page 2-102 and the `OCEGetDSSpecInfo` function on page 2-103.

One of the uses for the `DSSpec` structure is to specify access controls for a catalog node, record, or attribute type that supports access controls. The way that you accomplish this for PowerShare catalogs, for example, is to obtain a `DSSpec` structure by calling the `OCEGetAccessControlDSSpec` function. This function returns a pointer to a `DSSpec` structure based on the information you supply when you call the function. You can then use the `DSSpec` structure with access control functions such as `DirGetDNodeAccessControlGet`. For information on access control functions, see the section "Getting Access Controls" in the chapter "Catalog Manager" in this book.

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

typedef struct DSSpec DSSpec;
typedef DSSpec *DSSpecPtr;
```

Field descriptions

`entitySpecifier`
A pointer to a `RecordID` structure that contains the record information pertaining to the `DSSpec`. If the extension type is not `'entn'`, the contents of this field determine whether the `DSSpec` structure represents a catalog, a catalog node, a record, or the root of all catalogs.

`extensionType` The extension type of the `DSSpec` structure, if any. If the extension type is `'entn'` then the `DSSpec` has an extension. To determine whether a `DSSpec` structure has an extension type or not, you call the `OCEGetDSSpecInfo` function (page 2-103).

`extensionSize` The size, in bytes, of the extension (if any).

`extensionValue`
A pointer to the data of the extension.

PackedDSSpec

The `PackedDSSpec` structure is the packed form of the `DSSpec` structure. The `PackedDSSpec` structures are used by AOCE in various functions. For example, the `SDPGetPanelSelection` function uses a `PackedDSSpec` structure to indicate the record that the user has selected. Another use of the `PackedDSSpec` structure is as a component of an `Attribute` structure. If an attribute value has a `tag` field set to the value `typePackedDSSpec`, then the attribute contains data of type `PackedDSSpec`.

You can use the functions `OCEUnpackDSSpec` (page 2-98) and `OCEPackDSSpec` (page 2-97) to convert between the packed and unpacked forms of the `DSSpec` structure.

Note

The `PackedDSSpec` is not a maximum-sized structure. When you allocate a `PackedDSSpec` structure it will hold any valid packed `RecordID` structure, but not necessarily any additional extension data. u

```
#define kPackedDSSpecMaxBytes(sizeof (PackedRecordID) + \
   sizeof (OSType) + sizeof (unsigned short))
```

The constant kPackedDSSpecMaxBytes is the maximum size in bytes that can be stored in the data field of a PackedDSSpec structure.

```
struct PackedDSSpec {
      unsigned short    dataLength;/* length of data field */
      Byte              data[kPackedDSSpecMaxBytes];
};
```

Field descriptions

dataLength — The length of the data field of the PackedDSSpec structure. This does not include the bytes in the dataLength field itself.

data — An array containing the actual contents of the PackedDSSpec. The size of the data array is equal to kPackedDSSpecMaxBytes bytes.

```
typedef struct PackedDSSpec PackedDSSpec;
```

ProtoPackedDSSpec

The ProtoPackedDSSpec structure is a minimum-sized structure. It is equivalent to a PackedDSSpec structure without a data field. You should use this data type whenever you need to create a variable length packed DSSpec structure.

```
struct ProtoPackedDSSpec {
   unsigned short    dataLength;/* length of data field */
/* Followed by data */
};

typedef struct ProtoPackedDSSpec ProtoPackedDSSpec;
typedef ProtoPackedDSSpec *ProtoPackedDSSpecPtr;
```

Field descriptions

dataLength — The length of the data field of the PackedDSSpec structure.

Note

You must create the data portion of the ProtoPackedDSSpec structure yourself. Since this is a minimum-sized structure, it initially has no data field and hence no memory is allocated for any contents. u

Attribute Structures

The attribute structures are used in AOCE to provide access to a record's contents, as well as to determine what type of data is stored in a record. The three main attribute structures are Attribute, AttributeType, and AttributeValue. The Attribute structure contains AttributeValue and AttributeType structures as components.

The AttributeValue structure is described on page 2-42. The AttributeType structure is a derivative of the RString structure (page 2-20) and is described on page 2-39.

Attributes

In AOCE, all information in a record is stored as attribute values of the record. An attribute can hold any type of data, and it is defined by the Attribute structure. Each Attribute structure contains an AttributeType, AttributeCreationID, and AttributeValue component. Certain types of attributes have been reserved by Apple Computer, Inc., but you can create other types as needed. The Attribute structure provides you with all the information you need to manipulate an attribute value. Because an attribute value may contain vastly different types of data depending upon its type, it is vital that you determine the type of attribute before attempting to manipulate or use its value.

Because the Attribute structure is composed of several substructures such as AttributeValue, which may contain structures of their own, the Attribute structure is described last in this section, after its component structures.

AttributeType

An attribute type is a component of the Attribute structure and is used to indicate what kind of information is stored in the value field of an Attribute structure. For a complete description of the Attribute and AttributeValue structures, see page 2-44 and page 2-42 respectively. You can define your own attribute types or use a standard attribute type. For a list of standard attribute types and their data formats see the description of OCEAttributeTypeIndex, next.

An attribute type consists of a character set code, a length containing the number of bytes in the body field, and the data in the body field. An AttributeType structure is identical to an RString structure, except that its maximum length is defined by the constant kAttributeTypeMaxBytes and its body field specifies the type of a given attribute. Attribute types must be larger than 0 bytes; AOCE does not allow NULL attribute types. You can typecast any AttributeType structure to an RString structure and use the RString utility functions on it. The RString utility functions are described in "AOCE String Functions" beginning on page 2-45.

In addition to being a component of an Attribute structure, the AttributeType structure is used by several of the AOCE Catalog Manager functions. In particular, the callback functions you create for the DirLookupParse and DirEnumerateAttributeTypesGet functions take an attribute type as an input. See the chapter "Catalog Manager" in this book for more information on these functions.

An attribute type is defined as follows:

```
struct AttributeType
{
   RStringHeader
   Byte  body[kAttributeTypeMaxBytes];
};

typedef struct AttributeType AttributeType;
typedef AttributeType *AttributeTypePtr;
```

The `RStringHeader`, described on page 2-19, defines the character set information that applies to the text of the `RString` structure and specifies the length, in bytes, of the `body` field of the `RString` structure.

Field descriptions

`body` An array of characters that contains the name of an attribute type. The maximum length of an attribute type is defined by the constant `kAttributeTypeMaxBytes`, and is equal to 32 bytes.

OCEAttributeTypeIndex

You should use the attribute type index whenever you need to obtain a standard attribute type. To do this, you call the `OCEGetIndAttributeType` function (page 2-94) with the proper value from the `OCEAttributeTypeIndex` list. The `OCEGetIndAttributeType` function returns a pointer to an `RString` structure containing the standard attribute type based on the index value you supplied.

All lowercase four-character combinations are reserved by Apple Computer, Inc., as are all uppercase and lowercase combinations of the sequence 'AOCE'.

```
#define  kMemberAttrTypeNum         1001  /* "Member" */
#define  kAdminsAttrTypeNum         1002  /* "Administrators" */
#define  kMailSlotsAttrTypeNum      1003  /* "mailslots" */
#define  kPrefMailAttrTypeNum       1004  /* "pref mailslot" */
#define  kAddressAttrTypeNum        1005  /* "Address" */
#define  kPictureAttrTypeNum        1006  /* "Picture" */
#define  kAuthKeyAttrTypeNum        1007  /* "auth key" */
#define  kTelephoneAttrTypeNum      1008  /* "Telephone" */
#define  kNBPNameAttrTypeNum        1009  /* "NBP Name" */
#define  kQMappingAttrTypeNum       1010  /* "ForwarderQMap" */
#define  kDialupSlotAttrTypeNum     1011  /* "DialupSlotInfo" */
#define  kHomeNetAttrTypeNum        1012  /* "Home Internet" */
#define  kCoResAttrTypeNum          1013  /* "Co-resident M&M" */
#define  kFwdrLocalAttrTypeNum      1014  /* "FwdrLocalRecord" */
```

```
#define  kConnectAttrTypeNum         1015  /* "Connected To" */
#define  kForeignAttrTypeNum         1016  /* "Foreign RLIs" */
#define  kOwnersAttrTypeNum          1017  /* "Owners" */
#define  kReadListAttrTypeNum        1018  /* "ReadList" */
#define  kWriteListAttrTypeNum       1019  /* "WriteList" */
#define  kDescriptorAttrTypeNum      1020  /* "Descriptor" */
#define  kCertificateAttrTypeNu      1021  /* "Certificate" */
#define  kMsgQsAttrTypeNum           1022  /* "MessageQs" */
#define  kPrefMsgQAttrTypeNum        1023  /* "PrefMessageQ" */
#define  kMasterPFAttrTypeNum        1024  /* "MasterPF" */
#define  kMasterNetSpecAttrTypeNum   1025  /* "MasterNetSpec" */
#define  kServersOfAttrTypeNum       1026  /* "Servers Of" */
#define  kParentCIDAttrTypeNum       1027  /* "Parent CID" */
#define  kNetworkSpecAttrTypeNum     1028  /* "NetworkSpec" */
#define  kLocationAttrTypeNum        1029  /* "Location" */
#define  kTimeSvrTypeAttrTypeNum     1030  /* "TimeServer Type" */
#define  kUpdateTimerAttrTypeNum     1031  /* "Update Timer" */
#define  kShadowsOfAttrTypeNum       1032  /* "Shadows Of" */
#define  kShadowServerAttrTypeNum    1033  /* "Shadow Server" */
#define  kTBSetupAttrTypeNum         1034  /* "TB Setup" */
#define  kMailSetupAttrTypeNum       1035  /* "Mail Setup" */
#define  kSlotIDAttrTypeNum          1036  /* "SlotID" */
#define  kGatewayFileIDAttrTypeNum   1037  /* "Gateway FileID" */
#define  kMailServiceAttrTypeNum     1038  /* "Mail Service" */
#define  kStdSlotInfoAttrTypeNum     1039  /* "Std Slot Info" */
#define  kAssoDirectoryAttrTypeNum   1040  /* "Asso. Catalog" */
#define  kDirectoryAttrTypeNum       1041  /* "Catalog" */
#define  kDirectoriesAttrTypeNum     1042  /* "Catalogs" */
#define  kSFlagsAttrTypeNum          1043  /* "SFlags" */
#define  kLocalNameAttrTypeNum       1044  /* "Local Name" */
#define  kLocalKeyAttrTypeNum        1045  /* "Local Key" */
#define  kDirUserRIDAttrTypeNum      1046  /* "Dir User RID" */
#define  kDirUserKeyAttrTypeNum      1047  /* "Dir User Key" */
#define  kDirNativeNameAttrTypeNum   1048  /* "Dir Native Name" */
#define  kCommentAttrTypeNum         1049  /* "Comment" */
#define  kRealNameAttrTypeNum        1050  /* "Real Name" */
#define  kPrivateDataAttrTypeNum     1051  /* "Private Data" */
#define  kDirTypeAttrTypeNum         1052  /* "Catalog Type" */
#define  kDSAMFileAliasAttrTypeNum   1053  /* "CSAM File Alias" */
#define  kCanAddressToAttrTypeNum    1054  /* "Can Address To" */
#define  kDiscriminatorAttrTypeNum   1055  /* "Discriminator" */
#define  kAliasAttrTypeNum           1056  /* "Alias" */
#define  kParentMSAMAttrTypeNum      1057  /* "Parent MSAM" */
```

```
#define  kParentDSAMAttrTypeNum     1058  /* "Parent CSAM" */
#define  kSlotAttrTypeNum           1059  /* "Slot" */
#define  kAssoMailServiceAttrTypeNum1060  /* "Asso. Mail
                                             Service" */
#define  kFakeAttrTypeNum           1061  /* "Fake" */
#define  kInheritSysAdminAttrTypeNum1062  /* "Inherit
                                              SysAdministrators" */
#define  kPreferredPDAttrTypeNum    1063  /* "Preferred PD" */
#define  kLastLoginAttrTypeNum      1064  /* "Last Login" */
#define  kMailerAOMStateAttrTypeNum 1065  /* "Mailer AOM State" */
#define  kMailerSendOptionsAttrTypeNum \
                                    1066  /* "Mailer Send
                                             Options" */
#define  kJoinedAttrTypeNum         1067  /* "Joined" */
#define  kUnconfiguredAttrTypeNum   1068  /* "Unconfigured" */
#define  kVersionAttrTypeNum        1069  /* "Version" */
#define  kLocationNamesAttrTypeNum  1070  /* "Location Names" */
#define  kActiveAttrTypeNum         1071  /* "Active" */
#define  kDeleteRequestedAttrTypeNum
                                    1072  /* "Delete Requested" */
#define  kGatewayTypeAttrTypeNum    1073  /* "Gateway Type" */
```

In addition, Apple Computer, Inc., has defined three other attribute type constants to simplify the task of enumerating the standard attribute types.

```
typedef unsigned short OCEAttributeTypeIndex;

#define kFirstOCEAttrTypeNum kMemberAttrTypeNum
/* the first standard attribute type */

#define kLastOCEAttrTypeNum kGatewayTypeAttrTypeNum
/* the last standard attribute type */

#define kNumOCEAttrTypes (kLastOCEAttrTypeNum -
                          kFirstOCEAttrTypeNum + 1)
                          /* the total number of attributes */
```

AttributeValue

The `AttributeValue` structure consists of a `tag` field that indicates the format of the attribute value, a `datalength` field specifying the number of bytes contained in the attribute value, and a pointer to the attribute value data itself. Apple Computer, Inc. has reserved tags for attribute values that consist of `RString` and `PackedDSSpec`

structures, as well as for an unspecified sequence of bytes. You can also define your own tags to specify the attribute value formats that you have created.

```
typedef DescType AttributeTag;/* same type used in AppleEvents */

enum {
   typeRString       =  'rstr',
   typePackedDSSpec  =  'dspc',
   typeBinary        =  'bnry'
};
```

Constant descriptions

`typeRString` The attribute value is an `RString` structure.

`typePackedDSSpec` The attribute value is a `PackedDSSpec` structure.

`typeBinary` The attribute value is a sequence of bytes not defined by a formal structure.

```
struct AttributeValue {
   AttributeTag   tag;        /* format of attribute value */
   unsigned long  dataLength; /* # of bytes in attribute value */
   Ptr            bytes;      /* points to attribute value data */
};

typedef struct AttributeValue AttributeValue;
typedef AttributeValue *AttributeValuePtr;
```

Field descriptions

`tag` A value that indicates the format of the attribute value contained in the `bytes` field. If the `tag` field is set to `'rstr'`, the attribute value is considered to be an `RString` type.

If the attribute value is an `RString` structure, then the maximum size of the `body` field of the `RString` structure is (`kAttrValueMaxBytes - sizeof(ProtoRString)`) bytes.

If the attribute value is a `DSSpec` structure, then the maximum amount of data that can be stored in the `DSSpec` structure is (`kAttrValueMaxBytes - sizeof(ProtoPackedDSSpec)`) bytes.

The `tag` field can also contain a value defined by you that specifies the format of the attribute value.

Apple's PowerShare catalogs and personal catalogs restrict attribute values to a maximum size of `kAttrValueMaxBytes` bytes. If the `tag` field is set to `'dspc'`, the attribute value is a `PackedDSSpec` type.

`dataLength`	The number of bytes in the buffer pointed to by the `bytes` field. If the `tag` field is equal to `'rstr'` or `'dspc'`, then this length also includes the size of the `dataLength` field of the `DSSpec` structure or the `RStringHeader` of the `RString` structure.
`bytes`	A pointer to a buffer that contains the attribute value. You must provide this buffer. The constant `kAttrValueMaxBytes` defines the maximum size of any attribute value.

Attribute

The `Attribute` structure completely defines an attribute value by specifying its attribute type, attribute creation identifier, attribute tag, and the attribute value.

```
typedef CreationID   AttributeCreationID;

struct Attribute {
   AttributeType        attributeType; /* type of the attribute */
   AttributeCreationID  cid;           /* the creationID of the
                                          attribute */
   AttributeValue       value;         /* the attribute value */
};

typedef struct Attribute Attribute;
```

Field descriptions

`attributeType`	The attribute type. Apple Computer, Inc. has reserved all attribute types that are four-letter lowercase combinations, as well as any uppercase and lowercase combination of the letters 'AOCE'. A complete list of reserved attribute types can be found on page 2-40.
`cid`	The attribute creation identifier that uniquely defines the attribute value within the record. The `AttributeCreationID` structure has the same definition as the `CreationID` structure (see page 2-26).
`value`	The data for the attribute.

AOCE Utility Functions

The AOCE utility functions make it easier to manipulate the AOCE data structures. These functions perform various tasks such as comparison, duplication, creation, and conversion of structures. To call any of the functions described here from assembly language, you need to perform the following actions:

1. Leave space on the stack for the function result, if any.
2. Push the parameters on the stack using Pascal calling convention. This means that `parameter1` is pushed first, `parameter2` is pushed second, and so forth.

3. Place the routine selector in register D0.
4. Call the `__OCEUtils` trap macro.

AOCE String Functions

The AOCE string functions described in this section facilitate the creation, duplication, and conversion of AOCE strings.

OCECopyRString

The `OCECopyRString` function copies one AOCE string into another AOCE string.

```
pascal OSErr OCECopyRString (const RString *str1, RString *str2,
                    unsigned short str2Length);
```

`str1` A pointer to the source AOCE string that you want to copy from. You must provide this structure.

`str2` A pointer to the destination AOCE string that you want to copy to. You must provide this structure.

`str2Length`
The length of the destination AOCE string, not including the header information.

DESCRIPTION

The `OCECopyRString` function copies the contents of the source AOCE string into the destination AOCE string. If the destination string is not large enough to hold the contents of the source string, then the `OCECopyRString` function returns a memory-full error. You obtain the proper size needed for the destination AOCE string from the value contained in the `RStringHeader` field of the source AOCE string. Once you obtain this value, you can then use it to allocate a destination AOCE string of the proper size.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0308

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	Not enough memory to copy the source string into the destination string, or the destination string is not large enough to hold the source string

SEE ALSO

The RString structure is described on page 2-20.

OCECToRString

The OCECToRString function converts a C string into an AOCE string.

```
pascal void OCECToRString (const char *cStr, CharacterSet charSet,
                           RString *rStr,
                           unsigned short rStrLength);
```

cStr — A pointer to the C string you want to convert.

charSet — The script code that the OCECToRString function uses for the RString structure's header.

rStr — A pointer to an RString structure. You must allocate this.

rStrLength — The length, in bytes, of the body field of the RString structure, not including the length of the header information. If the C string is longer than the AOCE string, then only the number of bytes equal to the value of the rStrLength parameter are copied from the C string into the AOCE string.

DESCRIPTION

Given a C string and a RString structure that you supply, the OCECToRString function converts the C string into the RString structure. The OCECToRString function uses the charSet and rStrLength parameters to create the RStringHeader field of the new RString structure.

SPECIAL CONSIDERATIONS

If the C string is longer than the AOCE string, then only the number of bytes equal to the value of the rStrLength parameter are copied from the C string into the AOCE string.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0339

SEE ALSO

The RString structure is described on page 2-20.

For information on converting an RString structure to or from a Pascal string, see the functions OCEPToRString (next) and OCERToPString (page 2-48).

OCEPToRString

The `OCEPToRString` function converts a Pascal string into an AOCE string.

```
pascal void OCEPToRString(ConstStr255Param pStr,
                          CharacterSet charSet,
                          RString *rStr,
                          unsigned short rStrLength);
```

`pStr` A pointer to the Pascal string you want to convert.

`charSet` The script code that the `OCEPToRString` function uses for the `RString` structure's header.

`rStr` A pointer to an `RString` structure. You must allocate this.

`rStrLength`
The length, in bytes, of the `body` field of the `RString` structure, not including the length of the header information. If the Pascal string is longer than the AOCE string, then only the number of bytes equal to the value of the `rStrLength` parameter are copied from the Pascal string into the AOCE string.

DESCRIPTION

The `OCEPToRString` function converts a Pascal string into an `RString` structure. The `OCEPToRString` function uses the `charSet` and `rStrLength` parameters to create the `RStringHeader` field of the new `RString` structure.

SPECIAL CONSIDERATIONS

If the Pascal string is longer than the AOCE string, then only the number of bytes equal to the value of the `rStrLength` parameter are copied from the Pascal string into the AOCE string.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$033A

SEE ALSO

The `RString` structure is described on page 2-20.

For information on converting an `RString` structure to a Pascal string, see the function `OCERToPString`, described next.

OCERToPString

The `OCERToPString` function converts an `RString` structure into a Pascal string.

```
pascal StringPtr OCERToPString (const RString *rStr);
```

`rStr` A pointer to an `RString` structure that you want to convert into a Pascal string.

DESCRIPTION

The `OCERToPString` function converts an `RString` structure into a Pascal string. As with all of the AOCE utility functions, no memory is allocated by this function, so the string pointer that is returned points directly back into the `RString` structure that you supply when you make the call.

You must check the character set, or script code of the AOCE string before calling the `OCEPToRString` function to determine how to handle the Pascal string returned by this function. Because `RString` structures contain character set information and Pascal strings do not, you need to decide how to interpret the Pascal string that is returned, because it may contain multibyte characters.

SPECIAL CONSIDERATIONS

You should check the length of the AOCE string that the `rStr` parameter points to before calling this function to see if the string is greater than 255 bytes. Because a Pascal string can contain at most 255 bytes, the `OCERToPString` function truncates the length of the Pascal string to the lower byte of the length of the AOCE string.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$033B

SEE ALSO

The `RString` structure is described on page 2-20.

To convert a Pascal string to an `RString` structure, use the function `OCEPToRString` described on page 2-47.

OCERelRString

The `OCERelRString` function compares two `RString` structures to determine their relative sorting order.

```
pascal short OCERelRString (const void *str1, const void *str2,
                            RStringKind kind);
```

`str1`	A pointer to the first `RString` structure you want to compare.
`str2`	A pointer to the second `RString` structure you want to compare
`kind`	The value the `OCERelRString` function uses to determine the proper method of comparing the two `RString` structures. See the description of the `RStringKind` type on page 2-24 for a complete definition of the different values for the `kind` parameter, and for the restrictions on when to use them.

DESCRIPTION

Given two `RString` structures pointed to by the parameters `str1` and `str2`, the `OCERelRString` function determines if the first AOCE string is greater than, equal to, or less than the second AOCE string. The `OCERelRString` uses the value of the `kind` parameter to determine how to compare the two `RString` structures. For certain kinds of `RString` structures, this function uses the International Utilities to compare them. Because the Text Utilities take into account primary and secondary ordering, this call will not return the value `sortsEqual` if the strings differ only in case (“Dave” is not equal to “dave”). For more information see the chapter “Text Utilities” in *Inside Macintosh: Text*.

The `OCERelRString` function can return the following values:

`sortsBefore`	–1	The first `RString` structure should sort before the second `RString` structure
`sortsEqual`	0	The two `RString` structures are equal
`sortsAfter`	1	The first `RString` structure should sort after the second `RString` structure

The result returned by the `OCERelRstring` function is undefined when either the `str1` parameter or the `str2` parameter is set to `nil`.

SPECIAL CONSIDERATIONS

Although this function uses the Text Utilities for comparing certain kinds of `RString` structures, it is still alright to call this routine at interrupt level.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$032D

SEE ALSO

The `RString` structure is described on page 2-20.

To compare two RString structures for equality only, use the OCEEqualRString function, described next.

OCEEqualRString

The OCEEqualRString function checks the equality of two RString structures.

```
pascal Boolean OCEEqualRString (const void *str1, const void
                                *str2, RStringKind kind);
```

str1	A pointer to the first RString structure you want to compare.
str2	A pointer to the second RString structure you want to compare.
kind	A value that defines what kind of RString structures the OCEEqualRString function is comparing.

DESCRIPTION

Given pointers to two RString structures, the OCEEqualRString function compares them for equality, and returns true if they are equal, false if they are not. If the two AOCE strings have the same length, then they are compared for equality, with the method of comparison dependent upon the value of the kind parameter. If the two AOCE strings have different lengths, then they are not equal. For certain kinds of RString structures, this function uses the Text Utilities to compare the strings. For more information see the chapter "Text Utilities" in *Inside Macintosh: Text*. See the description of the RStringKind type on page 2-24 for a complete definition of the different values for the kind parameter, and for the restrictions on when to use them.

SPECIAL CONSIDERATIONS

Although this function uses the Text Utilities for comparing certain kinds of RString structures, it is still alright to call this routine at interrupt level.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0316

SEE ALSO

The RString structure is described on page 2-20.

The RStringKind structure is described on page 2-24.

OCEValidRString

The `OCEValidRString` function checks the validity of an AOCE string.

```
pascal Boolean OCEValidRString (const void *str,RStringKind kind);
```

`str` A pointer to the AOCE string you want to validate.

`kind` The kind of AOCE string being validated.

DESCRIPTION

The `OCEValidRString` function checks the AOCE string you supply for validity based on the type of AOCE string specified by the `kind` parameter and returns `true` if the AOCE string structure is valid, `false` if it is not. See the description of the `RStringKind` type on page 2-24 for a complete definition of the different values for the `kind` parameter, and for the restrictions on when to use them. Currently this function checks for validity by ensuring that the length of the AOCE string is the proper size for its particular type. A `nil` pointer and a length of 0 for the `RString` structure are considered valid.

SPECIAL CONSIDERATIONS

The `OCEValidRString` function may be modified in the future to perform other checks for validity, so you should not assume that the only thing this function examines is the length of the AOCE string.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0338

SEE ALSO

The `RString` structure is described on page 2-20.

Creation Identifier Functions

The functions described in this section manipulate record and attribute creation identifiers in various ways. The two creation identifier data types are defined by the `CreationID` and `AttributeCreationID` structures, which are described on page 2-26.

OCEEqualCreationID

The `OCEEqualCreationID` function checks the equality of two `CreationID` structures.

```
pascal Boolean OCEEqualCreationID(const CreationID *cid1,
                                  const CreationID *cid2);
```

`cid1` A pointer to the first `CreationID` structure you want to compare.

`cid2` A pointer to the second `CreationID` structure you want to compare.

DESCRIPTION

Given pointers to two `CreationID` structures, `OCEEqualCreationID` compares the `CreationID` structures for equality, and returns `true` if their values are identical, `false` if they are not. Two `CreationID` structures are considered equal if each field in the first `CreationID` structure contains the same value as the corresponding field in the second `CreationID` structure.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$030C

SEE ALSO

The `CreationID` structure is described on page 2-26.

OCECopyCreationID

The `OCECopyCreationID` function copies one `CreationID` structure to another.

```
pascal void OCECopyCreationID(const CreationID *cid1,
                              CreationID *const cid2);
```

`cid1` A pointer to the source `CreationID` structure you want to copy from. You must provide this structure.

`cid2` A pointer to the destination `CreationID` structure you want to copy to. You must provide this structure.

DESCRIPTION

Given two `CreationID` structures pointed to by the parameters, `cid1` and `cid2`, the `OCECopyCreationID` function copies the contents of the first structure to the second.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0300

SEE ALSO

The CreationID structure is described on page 2-26.

OCENullCID

The OCENullCID function returns a pointer to a null CreationID structure.

```
pascal const CreationID *OCENullCID(void);
```

DESCRIPTION

The OCENullCID function returns a pointer to a null CreationID structure that is maintained by the AOCE toolbox. You can use the OCENullCID function to check a CreationID structure to see if it is set to NULL, or to create a NULL CID. To check for a null CreationID structure you can use the following code fragment (This fragment uses the OCEEqualCreationID function described on page 2-52):

```
if (OCEEqualCreationID (myCID, OCENullCID())
/* then myCID is NULL */
```

You do not need to deallocate the NULL CreationID structure returned by the OCENullCID function when you are done with it.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0344

SEE ALSO

The CreationID structure is described on page 2-26.

To set an existing CreationID structure to NULL, call the OCESetCreationIDtoNull function (page 2-54).

The OCECopyCreationID function is described on page 2-52.

OCEPathFinderCID

The `OCEPathFinderCID` function returns a pointer to a special `CreationID` structure called the path finder creation ID.

```
pascal const CreationID *OCEPathFinderCID(void);
```

DESCRIPTION

The `OCEPathFinderCID` function returns a pointer to the special creation identifier structure known as the path finder creation ID. The path finder creation ID is maintained by the AOCE toolbox so you do not need to deallocate it when you are finished using it. This special creation ID is used by certain functions in the AOCE Authentication Manager. This function is intended for future use and is currently only used internally by the AOCE toolbox.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$033C

SEE ALSO

The `CreationID` structure is described on page 2-26.

OCESetCreationIDtoNull

The `OCESetCreationIDtoNull` function sets a `CreationID` structure to `NULL`.

```
pascal void OCESetCreationIDtoNull(CreationID *const cid);
```

`cid` A pointer to the `CreationID` structure you want to set to `NULL`.

DESCRIPTION

The `OCESetCreationIDtoNull` function sets the `CreationID` structure you provide to `NULL`. The `OCESetCreationIDtoNull` function makes it easier for you to use other AOCE functions such as `AuthResolveCreationID` that require the `CreationID` structure passed into them to be set to `NULL` before they are called.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$032E

SEE ALSO

The `CreationID` structure is described on page 2-26.

For more information on the `AuthResolveCreationID` function see the chapter "Authentication Manager" in this book.

Packed Pathname Functions

The functions described in this section manipulate packed pathnames in various ways. The packed pathname is defined by the `PackedPathName` structure, which is described on page 2-29.

OCECopyPackedPathName

The `OCECopyPackedPathName` function copies the contents of one `PackedPathName` structure to another.

```
pascal OSErr OCECopyPackedPathName(const PackedPathName *path1,
                                   PackedPathName *path2,
                                   unsigned short path2Length);
```

`path1` A pointer to the source `PackedPathName` structure that you want to copy from.

`path2` A pointer to the destination `PackedPathName` structure that you want to copy to.

`path2Length`
The length, in bytes, of the `PackedPathName` structure pointed to by the `path2` parameter, not including the size information contained in the `dataLength` field.

DESCRIPTION

Given two `PackedPathName` structures pointed to by the parameters, `path1` and `path2`, the `OCECopyPackedPathName` function copies the contents of the first structure into the second. The `path2Length` parameter is the size, in bytes, of the destination `PackedPathName` structure excluding the size information contained in the `dataLength` field. The destination `PackedPathName` structure must be large enough to hold the entire contents of the source `PackedPathName` structure; otherwise, a memory-full error is returned. Therefore, when you allocate a new destination

PackedPathName structure as the destination, you must set its length field to the proper size before calling the OCECopyPackedPathName function.

You obtain the proper size for a PackedPathName structure from its dataLength field. Once you obtain this value, you can then use it to allocate a destination PackedPathName structure of the correct size.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0304

RESULT CODES

noErr	0	No error
memFullErr	–108	Not enough memory to copy path1 into path2

SEE ALSO

The PackedPathName structure is described on page 2-29.

OCEIsNullPackedPathName

The OCEIsNullPackedPathName function determines if the value of a PackedPathName structure is NULL.

```
pascal Boolean OCEIsNullPackedPathName(const PackedPathName
                                       *path);
```

path A pointer to the PackedPathName structure you want to evaluate.

DESCRIPTION

Given a pointer to a PackedPathName structure, the OCEIsNullPackedPathName function determines if it satisfies the conditions for being considered NULL and returns true if its value is NULL, false if it is not. The value true is returned for any of the following conditions:

- If the path parameter is set to nil.
- If the PackedPathName structure pointed to by the path parameter has a length of 0.
- If the PackedPathName structure pointed to by the path parameter is composed of 0 RString components.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$031D

SEE ALSO

The PackedPathName structure is described on page 2-29.

OCEPackedPathNameSize

The OCEPackedPathNameSize function computes the number of bytes required to create a PackedPathName structure, including the size information.

```
pascal unsigned short OCEPackedPathNameSize
                        (const RStringPtr parts[],
                         const unsigned short nParts);
```

parts — An array of pointers to RString structures containing the dNode names.

nParts — The number of individual dNode names that are contained in the parts array.

DESCRIPTION

The OCEPackedPathNameSize function computes the number of bytes of memory needed to hold a PackedPathName structure manufactured from the parts array. This length includes the size of the dataLength field of the PackedPathName structure.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0328

SEE ALSO

The PackedPathName structure is described on page 2-29.

For information on determining the number of partial pathnames within a PackedPathName structure see the OCEDNodeNameCount function, described next.

For information on packing and unpacking pathnames see the OCEUnpackPathName (page 2-58) and OCEPackPathName (page 2-60) functions.

OCEDNodeNameCount

The `OCEDNodeNameCount` function returns the number of `RString` structures, or catalog node names contained within a `PackedPathName` structure.

```
pascal unsigned short OCEDNodeNameCount
                                    (const PackedPathName *path);
```

`path` The `PackedPathName` structure that you want to evaluate.

DESCRIPTION

When you call the `OCEUnpackPathName` function to unpack a `PackedPathName` structure, you must pass it the number of dNodes that the path is composed of and allocate an array large enough to hold the pointers to each dNode name. The `OCEDNodeNameCount` function provides you with the number of dNodes contained in the path.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$032C

SEE ALSO

The `PackedPathName` structure is described on page 2-29.

For information on determining the size of a `PackedPathName` structure needed to hold all the components of a pathname, see the `OCEPackedPathNameSize` function on page 2-57.

For information on packing and unpacking pathnames see the `OCEUnpackPathName` (next) and `OCEPackPathName` (page 2-60) functions.

OCEUnpackPathName

The `OCEUnpackPathName` function unpacks a `PackedPathName` structure into its component `RString` structures.

```
pascal unsigned short OCEUnpackPathName(const PackedPathName
                                    *path, RString *const parts[],
                                    const unsigned short nParts);
```

`path` A pointer to the `PackedPathName` structure that you want unpacked.

parts	An array of pointers to RString structures that the OCEUnpackPathName function fills with pointers into the path parameter.
nParts	The size of the parts array.

DESCRIPTION

Given a pointer into a PackedPathName structure that you provide, the OCEUnpackPathName function breaks apart the structure specified by path into the individual RString structures it contains, writing pointers to these RString structures into the parts array. The parts array must be large enough to hold as many as nParts dNode names. You can determine the number of dNodes that a path contains by calling the OCEDNodeNameCount function (page 2-58).

The OCEUnpackPathName function returns the number of dNode names actually found during the process of unpacking. You should check this value to ensure that it corresponds to the nParts parameter that you supplied to verify that no discrepancies exist.

The RString structures are placed in the parts array in order from lowest to highest; that is, the first element beneath the top level in the PackedPathName structure is placed last in the parts array.

SPECIAL CONSIDERATIONS

The array in the parts parameter generated by the OCEUnpackPathName function contains pointers into the PackedPathName structure specified by the path parameter. You should not delete or reuse the PackedPathName structure pointed to by the path parameter until you are finished with the parts array as well. Otherwise, the parts array may no longer contain pointers to valid data.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0330

SEE ALSO

The PackedPathName structure is described on page 2-29.

For information on packing a pathname structure, see the OCEPackPathName function, described next.

For information on determining the size of a PackedPathName structure needed to hold all the components of a pathname, see the OCEPackedPathNameSize function on page 2-57.

For information on determining the number of partial pathnames within a `PackedPathName` structure, see the `OCEDNodeNameCount` function described on page 2-58.

OCEPackPathName

The `OCEPackPathName` function forms a `PackedPathName` structure from its component parts.

```
pascal OSErr OCEPackPathName(const RStringPtr parts[],
                             const unsigned short nParts,
                             PackedPathName *path,
                             unsigned short pathLength);
```

`parts` An array of `RString` structures that the `OCEPackPathName` function uses to form the packed pathname.

`nParts` The number of dNodes contained in the `parts` array.

`path` A pointer to a buffer that you have allocated to hold the `PackedPathName` structure.

`pathLength`
The size of the structure pointed to by the `path` parameter, not including the size information contained in the `dataLength` field. For information on determining the size of a `PackedPathName` structure needed to hold all the components of a pathname, see the `OCEPackedPathNameSize` function on page 2-57.

DESCRIPTION

The `OCEPackPathName` function takes a buffer that you supply and stores in it a `PackedPathName` structure that the function creates from an array of dNodes. The buffer must be large enough to hold the full packed pathname. If the buffer is too small to hold the entire packed pathname, then a memory-full error is returned.

The order in which you store the partial pathnames in the `parts` array is as follows: `parts[0]` should contain the last pathname element, and `parts[nParts - 1]` should contain the name of the first pathname element beneath the root.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0323

RESULT CODES

noErr	0	No error
memFullErr	–108	The buffer pointed to by the path parameter is not large enough to hold the entire contents of the parts array.

SEE ALSO

The PackedPathName structure is described on page 2-29.

For information on unpacking a pathname structure, see the OCEUnpackPathName function described on page 2-58.

For information on determining the size of a PackedPathName structure needed to hold all the components of a pathname, see the OCEPackedPathNameSize function on page 2-57.

For information on determining the number of partial pathnames within a PackedPathName structure, see the OCEDNodeNameCount function described on page 2-58.

OCEEqualPackedPathName

The OCEEqualPackedPathName checks the equality of two packed pathnames.

```
pascal Boolean OCEEqualPackedPathName(const PackedPathName *path1,
                                      const PackedPathName *path2);
```

path1	A pointer to the first PackedPathName you want to compare.
path2	A pointer to the second PackedPathName you want to compare.

DESCRIPTION

Given pointers to two PackedPathName structures, path1 and path2, the OCEEqualPackedPathName function compares them for equality and returns true if the two pathnames are equal and false if they are not. This function takes into account the proper case and diacritical marks of the various fields of the PackedPathName structures it compares. This function checks for equality in the following manner:

- If the value of both PackedPathName structures is NULL, they are equal. A PackedPathName structure is considered NULL if the pointer to it is set to nil, or if its length is 0, or if it contains 0 catalog node names.
- If the value of one PackedPathName structure is NULL, but the value of the other is not, they are not equal.
- If neither PackedPathName structures is NULL, but they do not contain the same number of catalog node names, they are not equal.
- If neither PackedPathName structures is NULL and they both contain the same number of catalog node names, then each catalog node name is compared with the

corresponding one with regard to case and diacritical marks. If every one compares exactly, then the PackedPathName structures are equal. Otherwise, they are not.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0311

SEE ALSO

The PackedPathName structure is described on page 2-29.

OCEValidPackedPathName

The OCEValidPackedPathName function checks a given PackedPathName structure for internal consistency.

```
pascal Boolean OCEValidPackedPathName(const PackedPathName *path);
```

path A pointer to the PackedPathName you want to validate.

DESCRIPTION

The OCEValidPackedPathName function returns true if the PackedPathName structure is valid; otherwise, it returns false. The OCEValidPackedPathName function checks the PackedPathName structure for validity by unpacking it and performing the following tests:

- If the pointer to the PackedPathName structure is set to nil, the OCEValidPackedPathName function considers the PackedPathName structure to be invalid and returns false.
- If the length of the PackedPathName structure is 0 it is considered valid.
- It checks that all of the catalog node names in the PackedPathName structure are valid by passing them to the OCEValidRString function (page 2-51).
- It adds up the lengths of all the catalog node names in the PackedPathName structure and verifies that the total length matches the length of the PackedPathName structure.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0334

SEE ALSO

The `PackedPathName` structure is described on page 2-29.

Catalog Discriminator Functions

The utility functions described in this section manipulate catalog discriminators. The catalog discriminator is defined by the `DirDiscriminator` structure described on page 2-30.

OCECopyDirDiscriminator

The `OCECopyDirDiscriminator` function copies the value of one `DirDiscriminator` structure to another.

```
pascal void OCECopyDirDiscriminator
                    (const DirDiscriminator *disc1,
                     DirDiscriminator *const disc2);
```

`disc1` A pointer to the source `DirDiscriminator` structure that you want to copy from. You must provide this structure.

`disc2` A pointer to the destination `DirDiscriminator` structure that you want to copy to. You must provide this structure.

DESCRIPTION

Given two `DirDiscriminator` structures pointed to by the parameters, `disc1` and `disc2`, the `OCECopyDirDiscriminator` function copies the contents of the first structure to the second.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0301

SEE ALSO

The `DirDiscriminator` structure is described on page 2-30.

OCEEqualDirDiscriminator

The `OCEEqualDirDiscriminator` function checks the equality of two `DirDiscriminator` structures.

```
pascal Boolean OCEEqualDirDiscriminator
                    (const DirDiscriminator *disc1,
                     DirDiscriminator *const disc2);
```

`disc1` A pointer to the first `DirDiscriminator` structure you want to compare.

`disc2` A pointer to the second `DirDiscriminator` structure you want to compare.

DESCRIPTION

Given pointers to two `DirDiscriminator` structures, the `OCEEqualDirDiscriminator` function determines if they are equal. It returns `true` if they are equal, `false` if they are not. The two `DirDiscriminator` structures are considered equal if their `signature` and `misc` fields match byte for byte.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$030D

SEE ALSO

The `DirDiscriminator` structure is described on page 2-30.

Record Location Information Functions

The functions described in this section manipulate record location information structures. The record location information structure is defined by the `RLI` data type, described on page 2-32.

OCENewRLI

The `OCENewRLI` function constructs an `RLI` structure from its component parts.

```
pascal void OCENewRLI(RLI *newRLI, const DirectoryName *dirName,
                            DirDiscriminator *discriminator,
                            const DNodeNum dNodeNumber,
                            const PackedPathName *path);
```

`newRLI` A pointer to the buffer where the `OCENewRLI` function stores the `RLI` structure it constructs. You must allocate this.

`dirName` A pointer to the catalog name you want incorporated into the `RLI` structure.

`discriminator` A pointer to the catalog discriminator you want incorporated into the `RLI` structure.

`dNodeNumber` The catalog node number you want incorporated into the `RLI` structure.

`path` A pointer to the packed pathname you want incorporated into the `RLI` structure.

DESCRIPTION

Given catalog name, discriminator, catalog node number, and packed pathname structures, the `OCENewRLI` function creates an `RLI` structure and replaces the contents of the buffer, `newRLI`, with the `RLI` structure that it forms.

SPECIAL CONSIDERATIONS

Because the `OCENewRLI` function does not allocate any memory, the `RLI` structure it forms uses the same `DirectoryName` structure and the same `PackedPathname` structure that you supplied as parameters. Therefore, you should not dispose of or reuse the `DirectoryName` and `PackedPathname` structures until you have finished using the `RLI` structure as well. Doing so will cause the pointers in the `RLI` structure to point to incorrect locations in memory and might cause your application to crash.

Use `OCENewRLI` instead of the `OCEUnPackRLI` function to create an `RLI` structure that you are going to make an alias for. An alias to an `RLI` structure created with the `OCEUnPackRLI` function does not work properly.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$031F

SEE ALSO

The `RLI` structure is described on page 2-32.

The `DirDiscriminator` structure is described on page 2-30.

OCEDuplicateRLI

The OCEDuplicateRLI function duplicates the contents of one RLI structure to another.

```
pascal void OCEDuplicateRLI(const RLI *rli1, RLI *rli2);
```

rli1 A pointer to the source RLI structure. You must allocate this structure.

rli2 A pointer to the destination RLI structure. You must allocate this structure; however, you do not have to allocate the structures that this RLI structure points to.

DESCRIPTION

The OCEDuplicateRLI function copies the pointers from the directoryName and path fields of the source RLI structure to the corresponding fields in the destination RLI structure. This function does not copy the data these fields point to, only the pointers to the data. After you call the OCEDuplicateRLI function, each RLI structure contains pointers to the same PackedPathName and DirectoryName structures. This means that if you free the memory for one RLI structure's PackedPathName or DirectoryName structure, you are freeing the same structure in the corresponding RLI structure as well. In addition, the OCEDuplicateRLI function copies the values from the source RLI structure's dirDiscriminator and dNodeNumber fields into the corresponding fields of the destination RLI structure.

To actually copy the contents of the structures that the DirectoryNamePtr and PackedPathNamePtr fields point to from one RLI to another, use the OCECopyRLI function, described next.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$030B

SEE ALSO

The RLI structure is described on page 2-32.

To copy the contents of one RLI structure to another see the OCECopyRLI function, described next.

To copy one PackedRLI structure to another see the OCECopyPackedRLI function on page 2-70.

For a description of the difference between copying and duplicating an RLI structure, see the section "Copying Versus Duplicating AOCE Data Structures" on page 2-15.

OCECopyRLI

The OCECopyRLI function copies the contents of one RLI structure into another.

```
pascal OSErr OCECopyRLI(const RLI *rli1, RLI *rli2);
```

rli1	A pointer to the source RLI structure. You must allocate this structure.
rli2	A pointer to the destination RLI structure. You must allocate this structure.

DESCRIPTION

Given pointers to two RLI structures pointed to by the parameters, rli1 and rli2, the OCECopyRLI function copies the contents of the first into the second. The destination RLI structure must already contain pointers to structures large enough to hold copies of the corresponding fields from the source RLI structure; otherwise, a memory-full error is returned. Therefore, when you allocate a new destination RLI structure, you must set the fields that define the length of the PackedPathName and DirectoryName structures pointed to by its path and directoryName fields to the proper size before calling the OCECopyRLI function.

You obtain the proper size for a PackedPathName from its dataLength field and that of a DirectoryName structure from its RStringHeader. Once you obtain these values, you can then use them to allocate structures of the correct size.

If you want a destination RLI structure that points to the same PackedPathName and DirectoryName structures as the source RLI structure, then use the OCEDuplicateRLI function (page 2-66). The OCEDuplicateRLI function changes the destination RLI structure's path and directoryName fields so that they point to the same data in the fields of the corresponding source RLI structure.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0307

RESULT CODES

noErr	0	No error
memFullErr	–108	The destination RLI structure is not large enough to hold the entire contents of the source RLI structure.

SEE ALSO

The RLI structure is described on page 2-32.

To create a destination RLI structure that points to the same PackedPathName and DirectoryName structures as the source RLI structure, use the OCEDuplicateRLI function on page 2-66.

To copy one PackedRLI structure to another see the OCECopyPackedRLI function on page 2-70.

The PackedPathName and DirectoryName structures are described on page 2-29 and page 2-22, respectively.

For a description of the difference between copying and duplicating an RLI structure, see the section "Copying Versus Duplicating AOCE Data Structures" on page 2-15.

OCEEqualRLI

The OCEEqualRLI function checks the equality of two record location information structures.

```
pascal Boolean OCEEqualRLI(const RLI *rli1, const RLI *rli2);
```

rli1	A pointer to the first RLI structure you want to compare.
rli2	A pointer to the second RLI structure you want to compare.

DESCRIPTION

Given pointers to two RLI structures, the OCEEqualRLI function compares them for equality and returns true if they are equal, false if they are not. This function takes into account differences in the case and diacritical marks of the catalog name and the pathname that are contained in the RLI structures.

If the RLI structure that the rli1 parameter points to contains a catalog node number and a nil pathname, and the RLI structure that the rli2 parameter points to contains the value kNULLDNodeNumber and a pathname that is not nil, then the comparison will fail. In other words, the two RLI structures must be of the same form before they can be compared for equality. The one exception to this rule is when the pathname contained in the two RLI structures is set to nil. In that case, a dNodeNumber field with a value of kNULLDNodeNumber, and a dNodeNumber field with a value of kRootDNodeNumber are treated as equal.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0315

SEE ALSO

The RLI structure is described on page 2-32.

To check two PackedRLI structures for equality, use the OCEEqualPackedRLI function (page 2-76).

OCEValidRLI

The OCEValidRLI function checks the validity of a record location information structure.

```
pascal Boolean OCEValidRLI(const RLI *theRLI);
```

theRLI A pointer to the RLI structure you want to check.

DESCRIPTION

The OCEValidRLI function returns true if the RLI structure is valid, false if it is not. It checks for validity in the following manner:

- If the pointer to the RLI structure is set to nil, then the OCEValidRLI function considers the RLI structure to be invalid and returns false.
- The OCEValidRLI function then checks if the catalog name length is greater than 0 and less than or equal to the constant kDirectoryNameMaxBytes. If it is not, then the RLI structure is not valid.
- The OCEValidRLI function then checks that the packed pathname, if specified, is valid by calling the OCEValidPackedPathName function (page 2-62). If the OCEValidPackedPathName function returns false, the RLI structure is not valid.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0337

SEE ALSO

The RLI structure is described on page 2-32.

To perform a validity check on a PackedRLI structure use the OCEValidPackedRLI function (page 2-77).

OCECopyPackedRLI

The OCECopyPackedRLI function copies the contents of one PackedRLI structure into another.

```
pascal OSErr OCECopyPackedRLI(const PackedRLI *prli1,
                              PackedRLI *prli2,
                              unsigned short prli2Length);
```

prli1 A pointer to the source PackedRLI structure.

prli2 A pointer to the destination PackedRLI structure.

prli2Length
The size of the destination PackedRLI structure, not including the size of the dataLength field.

DESCRIPTION

Given two PackedRLI structures pointed to by the parameters prli1 and prli2, the OCECopyPackedRLI function copies the contents of the first PackedRLI structure into the second. The prli2Length parameter is the size of the destination PackedRLI structure, excluding its dataLength field. The destination PackedRLI structure must be large enough to hold the entire contents of the source PackedRLI structure; otherwise, a memory-full error is returned.

You obtain the proper size for a PackedRLI structure from its dataLength field. Once you obtain this value, you can then use it to allocate a PackedRLI structure of the correct size.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0305

RESULT CODES

noErr	0	No error
memFullErr	–108	The destination PackedRLI structure is not large enough to hold the contents of the source PackedRLI structure

SEE ALSO

The PackedRLI structure is described on page 2-33.

To copy an RLI structure, use the OCECopyRLI function (page 2-67).

OCEPackedRLISize

The `OCEPackedRLISize` function computes the number of bytes of memory needed to hold a `PackedRLI` structure.

```
pascal unsigned short OCEPackedRLISize(const RLI *theRLI);
```

`theRLI` A pointer to an `RLI` structure.

DESCRIPTION

Given a pointer to an `RLI` structure, the `OCEPackedRLISize` function computes the number of bytes needed for a `PackedRLI` structure large enough to hold the data in the `RLI` structure. The number of bytes returned by the `OCEPackedRLISize` function includes the bytes in the field that specifies the length of the `PackedRLI` structure, which enables you to allocate the correct amount of memory for a `PackedRLI` structure.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$032A

SEE ALSO

The `RLI` structure is defined on page 2-32.

The `PackedRLI` structure is defined on page 2-33.

To obtain the number of bytes necessary to create a `PackedRLI` structure from the component parts of an `RLI` structure, see the `OCEPackedRLIPartsSize` function on page 2-73.

OCEPackRLI

The `OCEPackRLI` function packs a record location information structure.

```
pascal OSErr OCEPackRLI(const RLI *theRLI, PackedRLI *prli,
                        unsigned short prliLength);
```

`theRLI` A pointer to the record location information structure you want packed.

`prli` A pointer to a `PackedRLI` structure. You must allocate this.

`prliLength`
The length, in bytes, of the `PackedRLI` structure pointed to by the `prli` parameter, excluding the bytes in the `dataLength` field.

DESCRIPTION

The `OCEPackRLI` function packs the contents of an `RLI` structure into a `PackedRLI` structure. During this process, the `OCEPackRLI` function replaces the contents of the `PackedRLI` structure with new data from the `RLI` structure. The `PackedRLI` structure must be large enough to hold the contents of the `RLI` structure when packed; otherwise, a memory-full error is returned. To determine the correct size for the `PackedRLI` structure, call the `OCEPackedRLISize` function (page 2-71).

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0324

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	The `PackedRLI` structure is not large enough to hold the contents of the `RLI` structure when packed

SEE ALSO

The `RLI` structure is defined on page 2-32.

The `PackedRLI` structure is defined on page 2-33.

For information on unpacking a `PackedRLI` structure see the `OCEUnpackRLI` function, next.

To create a `PackedRLI` structure from the component parts of an `RLI` structure, use the `OCEPackRLIParts` function on page 2-74.

To determine the correct size for the `PackedRLI` structure, call the `OCEPackedRLISize` function on page 2-71.

OCEUnpackRLI

The `OCEUnpackRLI` function unpacks a `PackedRLI` structure into its component parts.

```
pascal void OCEUnpackRLI(const PackedRLI *prli, RLI *theRLI);
```

`prli1` A pointer to the `PackedRLI` structure you want unpacked.

`theRLI` A pointer to the `RLI` structure. You must allocate this; however, you do not have to allocate the structures that this `RLI` structure points to.

DESCRIPTION

Given a PackedRLI structure pointed to by the prli1 parameter, and an RLI structure pointed to by the parameter theRLI, the OCEUnpackRLI function unpacks the PackedRLI structure into its components, writing pointers to these components into the RLI structure that you supply.

SPECIAL CONSIDERATIONS

The unpacked RLI structure contains pointers into the packed structure. Therefore, you should not delete or reuse the packed structure pointed to by the prli1 parameter until you are finished with the unpacked RLI structure as well.

An alias to an RLI structure created with the OCEUnPackRLI function does not work properly. If you unpack an RLI structure with OCEUnPackRLI, create an alias to it, and then pack it with OCEPackRLI, when you try to extract the alias with OCEExtractAlias, a nil value is returned for the new PackedRLI structure. Use the OCENewRLI function (page 2-64) instead of OCEUnPackRLI whenever you create an RLI structure with an alias.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0331

SEE ALSO

The RLI structure is defined on page 2-32.

The PackedRLI structure is defined on page 2-33.

For information on packing an RLI structure see the OCEPackRLI function on page 2-71.

OCEPackedRLIPartsSize

The OCEPackedRLIPartsSize function computes the size of a PackedRLI structure needed to hold the constituent parts of an RLI structure.

```
pascal unsigned short OCEPackedRLIPartsSize
                        (const DirectoryName *dirName,
                         const RStringPtr parts[],
                         const unsigned short nParts);
```

dirName	A pointer to a catalog name structure.
parts	An array containing the pathname parts.
nParts	The number of parts contained in the parts array.

DESCRIPTION

Given the component parts of a record location information structure, the `OCEPackedRLIPartsSize` function returns the size, in bytes, needed to create a `PackedRLI` structure large enough to hold all of the data and the `PackedRLI` `dataLength` field. This function is equivalent to the `OCEPackedRLISize` function (page 2-71), except that it takes the parts of an `RLI` structure as parameters instead of an `RLI` structure.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0329

SEE ALSO

The `RLI` structure is defined on page 2-32.

The `PackedRLI` structure is defined on page 2-33.

To pack the components of an `RLI` structure into a `PackedRLI` structure, see the `OCEPackRLIParts` function, described next.

OCEPackRLIParts

The `OCEPackRLIParts` function packs the components of a record location information structure into a `PackedRLI` structure.

```
pascal OSErr OCEPackRLIParts(const DirectoryName *dirName,
                             const DirDiscriminator *discriminator,
                             const DNodeNum dNodeNumber,
                             const RStringPtr parts[],
                             const unsigned short nParts,
                             PackedRLI *prli,
                             unsigned short prliLength);
```

`dirName` A pointer to a catalog name structure you want packed.

`discriminator`
A pointer to a `DirDiscriminator` value you want packed.

`dNodeNumber`
The catalog node number you want packed.

`parts` An array of pointers to `RString` structures, each of which is a dNode name on the path. The total array is the pathname structure that you want packed.

`nParts` The number of dNode names contained in the `parts` array.

`prli` A pointer to a `PackedRLI` structure that you have allocated.

`prliLength`
The length, in bytes, of the `PackedRLI` structure pointed to by the `prli` parameter.

DESCRIPTION

From all of the component pieces of a record location information structure, the `OCEPackRLIParts` function forms a `PackedRLI` structure. You must allocate the storage for the `PackedRLI` structure before calling this function. This function is equivalent to the `OCEPackRLI` function, except that it takes the parts of an `RLI` structure as its parameters instead of an `RLI` structure. The `OCEPackRLIParts` function examines the `prliLength` parameter to see if the structure pointed to by the `prli` parameter is large enough to hold the packed contents of the `RLI` structure, and returns a memory-full error if it is not. Use the `OCEPackedRLIPartsSize` function to obtain the size needed for a `PackedRLI` structure large enough to hold the data from all of the pieces of an `RLI` structure.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0325

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	The `PackedRLI` structure is not large enough to hold the packed components of the `RLI` structure

SEE ALSO

The `RLI` structure is defined on page 2-32.

The `PackedRLI` structure is defined on page 2-33.

For information on unpacking a `PackedRLI` structure see the `OCEUnpackRLI` function on page 2-72.

To obtain the number of bytes necessary to create a `PackedRLI` structure from the component parts of an `RLI` structure, see the `OCEPackedRLIPartsSize` function on page 2-73.

OCEEqualPackedRLI

The OCEEqualPackedRLI function checks the equality of two packed record location information structures.

```
pascal Boolean OCEEqualPackedRLI(const PackedRLI *prli1,
                                 const PackedRLI *prli2);
```

prli1 A pointer to the first PackedRLI structure you want to compare.

prli2 A pointer to the second PackedRLI structure you want to compare.

DESCRIPTION

The OCEEqualPackedRLI function determines if two PackedRLI structures are equal and returns true if they are, false if they are not. This function checks for equality in the following manner:

- If the value of both PackedRLI structures is NULL they are equal. The PackedRLI structures are set to NULL if the pointers to them are nil, or if they have a length of 0.
- If only one PackedRLI structure is NULL, the PackedRLI structures are not equal.
- If neither PackedRLI structures is NULL, then they are unpacked and their discriminator and dNodeNumber field's values are compared. If these values are not identical, then the PackedRLI structures are not equal. If the values are identical, then the DirectoryName and PackedPathName structures are compared for equality by calling the OCEEqualRString (page 2-50) and OCEEqualPackedPathName (page 2-61) functions. If the DirectoryName and PackedPathName structures are equal then the PackedRLI structures are equal; otherwise, the PackedRLI structures are not equal.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0313

SEE ALSO

The PackedRLI structure is defined on page 2-33.

To check the equality of two RLI structures use the OCEEqualRLI function (page 2-68).

OCEValidPackedRLI

The OCEValidPackedRLI function checks the validity of a packed record location information structure.

```
pascal Boolean OCEValidPackedRLI(const PackedRLI *prli);
```

prli A pointer to a PackedRLI structure.

DESCRIPTION

The OCEValidPackedRLI function checks a PackedRLI structure for validity and returns true if it is valid, false if it is not. The OCEValidPackedRLI function determines validity by unpacking the PackedRLI structure and then performing the following tests on it:

- If the pointer to the PackedRLI structure is nil, or the PackedRLI structure has a length of 0, then the PackedRLI structure is not valid.
- The OCEValidPackedRLI function determines if the PackedRLI structure is larger than the smallest possible PackedRLI structure. If it is not, then the PackedRLI structure is not valid.
- The OCEValidPackedRLI function then checks that the catalog name of the PackedRLI structure is valid by calling the OCEValidRString function (page 2-51). If the OCEValidRString function returns false, then the PackedRLI structure is not valid.
- The OCEValidPackedRLI function then checks the validity of the packed pathname of the PackedRLI structure by calling the OCEValidPackedPathName function (page 2-62). If the OCEValidPackedPathName function returns false, then the PackedRLI structure is not valid.
- The OCEValidPackedRLI function then adds up the sizes of all of the fields in the PackedRLI structure and compares the total number of bytes to the value contained in the dataLength field of the PackedRLI structure. If the two values are equal, then the PackedRLI structure is valid; otherwise, it is not valid.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0336

SEE ALSO

The PackedRLI structure is defined on page 2-33.

To check the validity of an RLI structure, use the OCEValidRLI function (page 2-69).

OCEExtractAlias

The `OCEExtractAlias` function returns an alias record from a packed record location information structure.

```
pascal AliasPtr OCEExtractAlias(const PackedRLI *prli);
```

`prli` A pointer to the `PackedRLI` structure containing the alias you want to extract.

DESCRIPTION

If the `PackedRLI` structure describes a personal catalog, the `OCEExtractAlias` function extracts an HFS alias to the personal catalog.

To use the alias, connect it to a handle and call the `ResolveAlias` function as shown in the following code sample.

```
aliasPtr = OCEExtractAlias()
status = PtrToHand(
   (Ptr) aliasPtr,
   (Handle *) &aliasHandle,
   aliasPtr->aliasSize
   );
if (status == noErr)
status = ResolveAlias(NULL, aliasHandle, &theFSSpec, &wasChanged);
```

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0318

SEE ALSO

The `PackedRLI` structure is defined on page 2-33.

See the chapter “Alias Manager” in *Inside Macintosh: Files* for more information on aliases and the alias structure.

OCEGetDirectoryRootPackedRLI

The `OCEGetDirectoryRootPackedRLI` function returns a pointer to a special packed `RLI` structure that represents the root of all catalogs.

```
pascal const PackedRLI * OCEGetDirectoryRootPackedRLI (void)
```

DESCRIPTION

You use the `OCEGetDirectoryRootPackedRLI` function whenever you need to obtain the record location information for the root of all catalogs. This `PackedRLI` structure is maintained by the AOCE toolbox, and therefore you never need to free the `PackedRLI` structure returned by the `OCEGetDirectoryRootPackedRLI` function when you have finished using it.

Clients of the AOCE standard catalog-browsing panel can use the `PackedRLI` returned by this function to tell the Standard Catalog panel to begin displaying catalogs from the root, thus allowing the user to see all of the catalogs configured on the computer.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0346

SEE ALSO

The `PackedRLI` structure is defined on page 2-33.

The catalog-browsing panel is described in the chapter “Standard Catalog Package” in this book.

Local Record Identifier Functions

The functions described in this section manipulate local record identifier structures. The local record identifier is defined by the `LocalRecordID` structure (page 2-27).

OCENewLocalRecordID

The `OCENewLocalRecordID` function converts the data you supply into a `LocalRecordID` structure.

```
pascal void OCENewLocalRecordID(const RString *recordName,
                                const RString *recordType,
                                const CreationID *cid,
                                LocalRecordID *lRID);
```

`recordName`
A pointer to an `RString` structure containing the record name you want stored in the `LocalRecordID` structure.

`recordType`
A pointer to an `RString` structure containing the record type you want stored in the `LocalRecordID` structure.

`cid`	A pointer to the `CreationID` structure you want stored in the `LocalRecordID` structure.
`lRID`	A pointer to a `LocalRecordID` structure you have allocated.

DESCRIPTION

The `OCENewLocalRecordID` function converts a record name, record type, and creation identifier into a `LocalRecordID` structure. You must allocate the storage for the `LocalRecordID` structure before calling this function.

SPECIAL CONSIDERATIONS

Because the `OCENewLocalRecordID` function does not allocate any memory, the `LocalRecordID` structure it forms uses the same `RString` structures and the same `CreationID` structure that you supplied as parameters. Therefore, you should not dispose of or reuse the `RSTring` and `CreationID` structures until you have finished using the `LocalRecordID` structure as well. Doing so will cause the pointers in the `LocalRecordID` structure to point to incorrect locations in memory and might cause your application to crash.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$031E

SEE ALSO

The `LocalRecordID` structure is defined on page 2-27.

OCECopyLocalRecordID

The `OCECopyLocalRecordID` function copies one `LocalRecordID` structure into another.

```
pascal OSErr OCECopyLocalRecordID(const LocalRecordID *lRID1,
                                  LocalRecordID *lRID2);
```

`lRID1`	A pointer to the source `LocalRecordID` structure.
`lRID2`	A pointer to the destination `LocalRecordID` structure.

DESCRIPTION

Given two `LocalRecordID` structures, the `OCECopyLocalRecordID` function copies the contents of the first one into the second. The destination `LocalRecordID` structure

must contain pointers to RString structures large enough to hold copies of the corresponding fields from the source LocalRecordID structure; otherwise, a memory-full error is returned. Therefore, when you allocate a new destination LocalRecordID structure, you must set the length fields of the RString structures pointed to by recordName and recordType to their proper values before calling the OCECopyLocalRecordID function. You obtain the correct size for these Rstring structures from their headers in the source LocalRecordID structure.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0302

RESULT CODES

noErr	0	No error
memFullErr	–108	The destination LocalRecordID structure is not large enough to hold the contents of the source LocalRecordID structure

SEE ALSO

The LocalRecordID structure is defined on page 2-27.

OCEEqualLocalRecordID

The OCEEqualLocalRecordID function checks the equality of two LocalRecordID structures.

```
pascal Boolean OCEEqualLocalRecordID(const LocalRecordID *lRID1,
                                     const LocalRecordID *lRID2);
```

lRID1	A pointer to the first LocalRecordID structure you want to compare.
lRID2	A pointer to the second LocalRecordID structure you want to compare.

DESCRIPTION

The OCEEqualLocalRecordID function compares the two LocalRecordID structures for equality in the following manner:

- The recordName and recordType fields of the two LocalRecordID structures are compared for equality by calling the OCEEqualRString (page 2-50) function and passing it the proper RStringKind value for each field.
- The cid fields of the two LocalRecordID structures are compared for equality by calling the OCEEqualCreationID function (page 2-52).

If the `recordName`, `recordType`, and `CreationID` fields of the two `LocalRecordID` structures are equal, then the `OCEEqualLocalRecordID` function returns `true`; otherwise, it returns `false`.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$030E

SEE ALSO

The `LocalRecordID` structure is defined on page 2-27.

The `RStringKind` structure is defined on page 2-24.

Short Record Identifier Functions

The functions described in this section manipulate short record identifiers. The short record identifier is defined by the `ShortRecordID` structure (page 2-35).

OCENewShortRecordID

The `OCENewShortRecordID` function converts data you supply into a `ShortRecordID` structure.

```
pascal void OCENewShortRecordID(const PackedRLI *theRLI,
                                const CreationID *cid,
                                ShortRecordIDPtr *sRID);
```

`theRLI`	A pointer to the packed record location information structure containing data you want stored in the `ShortRecordID` structure.
`cid`	A pointer to the creation identifier structure containing data you want stored in the `ShortRecordID` structure.
`sRID`	A pointer to a `ShortRecordID` structure you have allocated.

DESCRIPTION

The `OCENewShortRecordID` function converts a `CreationID` structure and a `PackedRLI` structure into a `ShortRecordID` structure. You must allocate the `ShortRecordID` structure before calling this function.

SPECIAL CONSIDERATIONS

Because the `OCENewRecordID` function does not allocate any memory, the `ShortRecordID` structure it forms uses the same `PackedRLI` structure and the same

CreationID structure that you supplied as parameters. Therefore, you should not dispose of or reuse the PackedRLI and CreationID structures until you have finished using the ShortRecordID structure as well. Doing so will cause the pointers in the ShortRecordID structure to point to incorrect locations in memory and might cause your application to crash.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0321

SEE ALSO

The ShortRecordID structure is defined on page 2-35.

OCECopyShortRecordID

The OCECopyShortRecordID function copies one ShortRecordID structure into another.

```
pascal OSErr OCECopyShortRecordID(const ShortRecordID *sRID1,
                                  ShortRecordID *sRID2);
```

sRID1	A pointer to the source ShortRecordID structure.
sRID2	A pointer to the destination ShortRecordID structure.

DESCRIPTION

Given two ShortRecordID structures pointed to by the sRID1 and sRID2 parameters, the OCECopyShortRecordID function copies the data from the first one into the second. The destination ShortRecordID structure must contain pointers to structures large enough to hold copies of the corresponding fields from the source ShortRecordID structure; otherwise, a memory-full error is returned. Therefore, when you allocate a new destination ShortRecordID structure, you must set the dataLength field of its PackedRLI component to the proper value before calling the OCECopyShortRecordID function.

You obtain the correct size for a PackedRLI structure from the value contained in its dataLength field. Once you obtain this value, you can then use it to allocate a PackedRLI structure of the correct size.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$030A

RESULT CODES

noErr	0	No error
memFullErr	–108	The destination ShortRecordID structure is not large enough to hold the contents of the source ShortRecordID structure

SEE ALSO

The ShortRecordID structure is defined on page 2-35.

OCEEqualShortRecordID

The OCEEqualShortRecordID function checks the equality of two short record identifier structures.

```
pascal Boolean OCEEqualShortRecordID(const ShortRecordID *sRID1,
                                     const ShortRecordID *sRID2);
```

sRID1	A pointer to the first ShortRecordID structure you want to compare.
sRID2	A pointer to the second ShortRecordID structure you want to compare.

DESCRIPTION

If both ShortRecordID structures are equal, then the OCEEqualShortRecordID function returns true; otherwise, it returns false.

The OCEEqualShortRecordID function compares two ShortRecordID structures for equality in the following manner:

- If both pointers to the ShortRecordID structures are set to nil, then they are equal.
- If one of the pointers to a ShortRecordID structure is set to nil and the other is not, then the OCEEqualShortRecordID function returns false.
- If neither pointer to the ShortRecordID structures is set to nil, then the cid fields of the two ShortRecordID structures are compared for equality by calling the OCEEqualCreationID function (page 2-52). If the OCEEqualCreationID function returns false, then the ShortRecordID structures are not equal.
- If the CreationID fields of the two ShortRecordID structures are equal, then the PackedRLI structures pointed to by the PackedRLIPtr fields of the two ShortRecordID structures are compared for equality by calling the OCEEqualPackedRLI function (page 2-76). If the OCEEqualPackedRLI function returns true, then the two ShortRecordID structures are equal; otherwise, they are not.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0317

SEE ALSO

The ShortRecordID structure is defined on page 2-35.

Record Identifier Functions

The functions in this section manipulate record identifier structures. The record identifier is defined by the RecordID data structure (page 2-34).

OCEGetIndRecordType

The OCEGetIndRecordType function returns a standard record type based on the index value you pass to it.

```
pascal RString *OCEGetIndRecordType
                        (const OCERecordTypeIndex stringIndex);
```

stringIndex
One of the index values from the OCERecordTypeIndex enumerated list.

DESCRIPTION

To obtain a standard record type, you call the OCEGetIndRecordType function and pass it an index value based on the type of record you want. The record type index (page 2-28) is an enumerated list containing all of the standard AOCE record types.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$031B

SEE ALSO

The recordType field is part of the LocalRecordID structure defined on page 2-27.

For an enumerated list containing all of the standard AOCE record types, see the record type index on page 2-28.

OCENewRecordID

The `OCENewRecordID` function converts data you supply into a `RecordID` structure.

```
pascal void OCENewRecordID(const PackedRLI *theRLI,
                           const LocalRecordID *lRID, RecordID *rid);
```

`theRLI` A pointer to the `PackedRLI` structure you want stored in the `RecordID` structure.

`lRID` A pointer to the `LocalRecordID` structure you want stored in the `RecordID` structure.

`rid` A pointer to the destination `RecordID` structure. You must allocate this structure.

DESCRIPTION

The `OCENewRecordID` function converts a `PackedRLI` structure and `LocalRecordID` structure into a `RecordID` structure.

SPECIAL CONSIDERATIONS

Because the `OCENewRecordID` function does not allocate any memory, the `RecordID` structure it forms uses the same `PackedRLI` structure and the same `LocalRecordID` structure that you supplied as parameters. Therefore, you should not dispose of or reuse the `PackedRLI` and `LocalRecordID` structures until you have finished using the `RecordID` structure as well. Doing so will cause the pointers in the `RecordID` structure to point to incorrect locations in memory and might cause your application to crash.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0320

SEE ALSO

The `RecordID` structure is defined on page 2-34.

OCECopyRecordID

The `OCECopyRecordID` function copies one `RecordID` structure to another.

```
pascal OSErr OCECopyRecordID(const RecordID *rid1,
                             const RecordID *rid2);
```

`rid1`	The source `RecordID` structure.
`rid2`	The destination `RecordID` structure.

DESCRIPTION

Given two `RecordID` structures pointed to by the `rid1` and `rid2` parameters, the `OCECopyRecordID` function copies the contents of the first one into the second. The destination `RecordID` structure must contain pointers to structures large enough to hold copies of the corresponding fields from the source `RecordID` structure; otherwise, a memory-full error is returned. Therefore, when you allocate a new destination `RecordID` structure, you must set the length fields of its `LocalRecordID.recordName`, `LocalRecordId.recordType`, and `LocalRecordID.PackedRLI` fields to their proper values before calling the `OCECopyRecordID` function.

You obtain the correct size for the `LocalRecordID.recordName` and `LocalRecordID.recordType` fields of a `RecordID` structure from the values contained in their `RStringHeader` fields. Once you obtain these values, you can then use them to allocate `recordName` and `recordType` structures of the correct size.

You obtain the correct size for a `LocalRecordId.PackedRLI` structure from the value contained in its `dataLength` field. Once you obtain this value you can then use it to allocate a `PackedRLI` structure of the correct size.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0309

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	The destination `RecordID` structure is not large enough to hold the contents of the source `RecordID` structure

SEE ALSO

The `RecordID` structure is defined on page 2-34.

OCEEqualRecordID

The `OCEEqualRecordID` function checks the equality of two record identifier structures.

```
pascal Boolean OCEEqualRecordID(const RecordID *rid1,
                                const RecordID *rid2);
```

rid1	A pointer to the first RecordID you want to compare.
rid2	A pointer to the second RecordID you want to compare.

DESCRIPTION

The OCEEqualRecordID function compares two RecordID structures for equality and returns true if they are equal, false if they are not. This function checks the two RecordID structures for equality in the following manner:

- If both pointers to the RecordID structures are set to nil, then they are equal.
- If one of the pointers to a RecordID structure is set to nil and the other is not, then the OCEEqualRecordID function returns false.
- If neither pointer to the RecordID structures is set to nil, then the CreationID structures pointed to by the LocalRecordID.cid fields of the two RecordID structures are compared for equality by calling the OCEEqualCreationID function (page 2-52). If the OCEEqualCreationID function returns false, then the two RecordID structures are not equal.
- If the CreationID structures identified by the LocalRecordID.cid fields of the two RecordID structures are equal, then the PackedRLI structures pointed to by the PackedRLIPtr fields of the two RecordID structures are compared for equality by calling the OCEEqualPackedRLI function (page 2-76). If the OCEEqualPackedRLI function returns false, then the two RecordID structures are not equal.
- If the PackedRLI structures pointed to by the PackedRLIPtr fields of the two RecordID structures are equal, then the LocalRecordID.recordName and LocalRecordID.recordType fields of the two RecordID structures are compared for equality by calling the OCEEqualRString (page 2-50) function and passing it the proper RStringKind value for each field. If the OCEEqualRString function returns false, the two RecordID structures are not equal.

If the conditions for equality listed above are satisfied, then the two RecordID structures are equal; otherwise, they are not equal.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0314

SEE ALSO

The RecordID structure is defined on page 2-34.

The RStringKind structure is defined on page 2-24.

Packed Record Identifier Functions

The functions described in this section manipulate packed record identifiers. Packed record identifiers are defined by the PackedRecordID structure (page 2-35).

OCECopyPackedRecordID

The `OCECopyPackedRecordID` function copies one `PackedRecordID` structure to another.

```
pascal OSErr OCECopyPackedRecordID(const PackedRecordID *pRID1,
                                   const PackedRecordID *pRID2,
                                   unsigned short pRID2Length);
```

`pRID1` A pointer to the source `PackedRecordID` structure.

`pRID2` A pointer to the destination `PackedRecordID` structure.

`pRID2Length`
The length, in bytes, of the destination `PackedRecordID` structure, not including the bytes in the `dataLength` field.

DESCRIPTION

Given two `PackedRecordID` structures pointed to by the `pRID1` and `pRID2` parameters, the `OCECopyPackedRecordID` function copies the contents of the first into the second. The `pRID2Length` parameter is the size of the destination `PackedRecordID` structure, excluding its `dataLength` field. The destination `PackedRecordID` structure must be large enough to hold the entire contents of the source `PackedRecordID` structure; otherwise, a memory-full error is returned.

You obtain the proper size for a `PackedRecordID` structure from the value contained in its `dataLength` field. Once you obtain this value, you can then use it to allocate a destination `PackedRecordID` structure of the correct size.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0306

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	The `pRID2` parameter is not large enough to hold the entire contents of `pRID1`

SEE ALSO

The `PackedRecordID` structure is defined on page 2-35.

OCEPackedRecordIDSize

The `OCEPackedRecordIDSize` function computes the number of bytes of memory needed to hold a `PackedRecordID` structure.

```
pascal unsigned short OCEPackedRecordIDSize(const RecordID *rid);
```

`rid` A pointer to a `RecordID` structure.

DESCRIPTION

The `OCEPackedRecordIDSize` function returns the number of bytes that a `PackedRecordID` needs to hold the packed data from a specified `RecordID` structure. The number of bytes returned by the `OCEPackedRecordIDSize` function includes the size of the `datalength` field of the `PackedRecordID` structure.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$032B

SEE ALSO

The `RecordID` structure is defined on page 2-34.

The `PackedRecordID` structure is defined on page 2-35.

To unpack a `PackedRecordID` structure into a `RecordID` structure, use the `OCEUnpackRecordID` function (page 2-91).

OCEPackRecordID

The `OCEPackRecordID` function packs a `RecordID` structure into a `PackedRecordID` structure.

```
pascal OSErr OCEPackRecordID(const RecordID *rid,
                             PackedRecordID *pRID,
                             unsigned short packedRecordIDLength);
```

`rid` A pointer to the `RecordID` structure you want packed.

`pRID` A pointer to a `PackedRecordID` structure. You must allocate this structure.

`packedRecordIDLength`
The maximum length, in bytes, of the `PackedRecordID` structure, excluding the bytes in the `dataLength` field.

DESCRIPTION

The `OCEPackRecordID` function packs a `RecordID` structure into a `PackedRecordID` structure. The `PackedRecordID` structure must be large enough to contain the entire contents of the `RecordID` in packed format; otherwise, a memory-full error is returned. You obtain the size of a `PackedRecordID` structure large enough to hold the data in a `RecordID` structure by calling the `OCEPackedRecordIDSize` function described on page 2-90.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0326

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	The `PackedRecordID` structure is not large enough to hold the packed data from the `RecordID` structure

SEE ALSO

The `RecordID` structure is defined on page 2-34.

The `PackedRecordID` structure is defined on page 2-35.

To unpack a `PackedRecordID` structure into a `RecordID` structure, see the `OCEUnpackRecordID` function, described next.

OCEUnpackRecordID

The `OCEUnpackRecordID` function unpacks a `PackedRecordID` structure into a `RecordID` structure.

```
pascal void OCEUnpackRecordID(const PackedRecordID *pRID,
                              RecordID *rid);
```

`pRID` A pointer to the `PackedRecordID` structure you want to unpack.

`rid` A pointer to a `RecordID` structure. You must allocate this structure.

DESCRIPTION

Given a `PackedRecordID` structure pointed to by the `pRID` parameter and a `RecordID` structure pointed to by the `rid` parameter, the `OCEUnpackRecordID` function unpacks the `PackedRecordID` structure into the `RecordID` structure.

SPECIAL CONSIDERATIONS

Because the `OCEUnpackRecordID` function does not allocate any memory, the unpacked `RecordID` structure contains pointers into the `PackedRecordID` structure. Therefore, do not delete or reuse the `PackedRecordID` structure until you have finished using the unpacked `RecordID` structure as well. Doing so will cause the pointers in the `RecordID` structure to point to incorrect locations in memory, and your application may crash when you try to access the `RecordID` structure.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0332

SEE ALSO

The `RecordID` structure is defined on page 2-34.

The `PackedRecordID` structure is defined on page 2-35.

To pack a `RecordID` structure into a `PackedRecordID` structure, see the `OCEPackRecordID` function described on page 2-90.

OCEEqualPackedRecordID

The `OCEEqualPackedRecordID` function checks the equality of two `PackedRecordID` structures.

```
pascal Boolean OCEEqualPackedRecordID
                                (const PackedRecordID *pRID1,
                                 const PackedRecordID *pRID2);
```

`pRID1` A pointer to the first `PackedRecordID` structure you want to compare.

`pRID2` A pointer to the second `PackedRecordID` structure you want to compare.

DESCRIPTION

The `OCEEqualPackedRecordID` function compares two `PackedRecordID` structures for equality and returns `true` if they are equal and `false` if they are not.

This function checks the two `PackedRecordID` structures for equality in the following manner:

- If both pointers to the `PackedRecordID` structures are `nil`, then they are equal.
- If one of the pointers to a `PackedRecordID` structure is `nil` and the other is not, then the `PackedRecordID` structures are not equal.

- If neither pointer to the PackedRecordID structures is nil, then they are unpacked and the CreationID structures identified by the LocalRecordId.cid fields of the two unpacked PackedRecordID structures are compared for equality by calling the OCEEqualCreationID function (page 2-52). If the OCEEqualCreationID function returns false, then the two PackedRecordID structures are not equal.
- If the CreationID structures identified by the LocalRecordId.cid fields of the two unpacked PackedRecordID structures are equal, then the PackedRLI structures pointed to by the PackedRLIPtr fields of the two PackedRecordID structures are compared for equality by calling the OCEEqualPackedRLI function (page 2-76). If the OCEEqualPackedRLI function returns false, then the two PackedRecordID structures are not equal.
- If the PackedRLI structures pointed to by the PackedRLIPtr fields of the two (unpacked) PackedRecordID structures are equal, then the LocalRecordID.recordName and LocalRecordID.recordType fields of the two (unpacked) PackedRecordID structures are compared for equality by calling the OCEEqualRString (page 2-50) function and passing it the proper RStringKind value for each field. If the OCEEqualRString function returns false, the two PackedRecordID structures are not equal.

If the conditions for equality listed above are satisfied, then the two PackedRecordID structures are equal; otherwise, they are not equal.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0312

SEE ALSO

The PackedRecordID structure is defined on page 2-35.

The RStringKind structure is defined on page 2-24.

OCEValidPackedRecordID

The OCEValidPackedRecordID function checks the validity of a packed record identifier.

```
pascal Boolean OCEValidPackedRecordID(const PackedRecordID *pRID);
```

pRID A pointer to the PackedRecordID you want to validate.

DESCRIPTION

Given a pointer to a `PackedRecordID` structure, the `OCEValidPackedRecordID` function checks it for validity based on its internal structure and returns `true` if it is valid and `false` if it is not. The `OCEValidPackedRecordID` function checks a `PackedRecordID` structure for validity in the following manner:

- If the pointer to the `PackedRecordID` structure is set to `nil`, or the length of the `PackedRecordID` structure is 0, then the `PackedRecordID` structure is invalid.
- If the pointer to the `PackedRecordID` structure is not `nil` and the length of the structure is greater than 0, then it is unpacked and the `RLI` component of the `PackedRecordID` structure is validated by calling the `OCEValidRLI` function (page 2-69). If the `OCEValidRLI` function returns `false`, then the `PackedRecordID` structure is not valid.
- If the `RLI` component of the `PackedRecordID` structure is valid, then the `recordName` and `recordType` fields of the `PackedRecordID` structure are validated by calling the `OCEValidRString` function. If the `OCEValidRString` function returns `false`, then the `PackedRecordID` structure is not valid.
- If all of the conditions tested for validity are `true`, then the entire `PackedRecordID` structure is valid.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0335

SEE ALSO

The `PackedRecordID` structure is defined on page 2-35.

Attribute Type Functions

The function described in this section returns a standard attribute type. The attribute type is defined by the `AttributeType` data structure and is described on page 2-39.

OCEGetIndAttributeType

The `OCEGetIndAttributeType` function returns an attribute type based on the index value you pass to it.

```
pascal AttributeType *OCEGetIndAttributeType(const
                                OCEAttributeTypeIndex stringIndex);
```

`stringIndex`
One of the index values from the `OCEAttributeTypeIndex` enumerated list.

DESCRIPTION

To obtain a standard attribute type, you call the `OCEGetIndAttributeType` function and pass it an index value based on the kind of attribute type you want. The attribute type index (page 2-40) is an enumerated list containing all of the standard AOCE attribute types.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$031A

SEE ALSO

The `AttributeType` structure is defined on page 2-39.

For an enumerated list of all the standard AOCE attribute types, see the attribute type index on page 2-40.

Catalog Services Specification Functions

The functions described in this section manipulate the various catalog services specification data structures. The catalog services specification is defined by the `DSSpec` data structure (page 2-36) and its packed form by the `PackedDSspec` structure (page 2-37). These functions perform such tasks as copying, comparing, unpacking, and retrieving information from `DSSpec` structures.

Other forms of the `DSSpec` structure include the `OCERecipient` and the packed form, `OCEPackedRecipient`, which are defined in the chapter "Interprogram Messaging Manager" in this book. The functions, such as `OCEPackRecipient`, that manipulate these data structures are also described in the chapter "Interprogram Messaging Manager."

OCECopyPackedDSSpec

The `OCECopyPackedDSSpec` function copies data from one `PackedDSSpec` into another.

```
pascal OSErr OCECopyPackedDSSpec(const PackedDSSpec *pdss1,
         const PackedDSSpec *pdss2, unsigned short pdss2Length);
```

`pdss1` A pointer to the source `PackedDSSpec` structure.

`pdss2` A pointer to the destination `PackedDSSpec` structure.

`pdss2Length`
The length, in bytes, of the destination `PackedDSSpec` structure, not including the header information.

DESCRIPTION

Given two `PackedDSSpec` structures pointed to by the `pdss1` and `pdss2` parameters, the `OCECopyPackedDSSpec` function copies the first into the second. The `pdss2Length` parameter is the size, in bytes, of the destination `PackedDSSpec` structure, excluding its header. The destination `PackedDSSpec` structure must be large enough to hold the entire contents of the source `PackedDSSpec` structure; otherwise, a memory-full error is returned.

You obtain the proper size for a `PackedDSSpec` structure from the value contained in its `dataLength` field. Once you obtain this value, you can then use it to allocate a destination `PackedDSSpec` structure of the correct size.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0303

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	The destination `PackedDSSpec` structure is not large enough to hold the contents of the source `PackedDSSpec` structure

SEE ALSO

The `PackedDSSpec` data structure is defined on page 2-37.

OCEPackedDSSpecSize

The `OCEPackedDSSpecSize` function computes the number of bytes of memory needed to hold a packed `DSSpec` structure.

```
pascal unsigned short OCEPackedDSSpecSize(const DSSpec *dss);
```

`dss` A pointer to the `DSSpec` structure whose size, when packed, you want to determine.

DESCRIPTION

The `OCEPackedDSSpecSize` function computes the number of bytes required to hold the information contained in a `DSSpec` structure when it is packed. The number of bytes returned by the `OCEPackedDSSpecSize` function includes the `dataLength` field of the `PackedDSSpec` structure.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0327

SEE ALSO

The `DSSpec` structure is defined on page 2-36.

The `PackedDSSpec` structure is defined on page 2-37.

To pack a `DSSpec` structure, use the `OCEPackDSSpec` function, described next.

OCEPackDSSpec

The `OCEPackDSSpec` function forms a `PackedDSSpec` structure from a `DSSpec` structure.

```
pascal OSErr OCEPackDSSpec(const DSSpec *dss, PackedDSSpec *pdss,
                           unsigned short pdssLength);
```

`dss` A pointer to the `DSSpec` structure that you want to pack.

`pdss` A pointer to a `PackedDSSpec` structure. You must allocate this structure.

`pdssLength` The maximum number of bytes that can be stored in the `PackedDSSpec` structure, not including the header information.

DESCRIPTION

The `OCEPackDSSpec` function packs the contents of a `DSSpec` structure into a `PackedDSSpec` structure. The `PackedDSSpec` structure must be large enough to contain the packed `RecordID` information and any extension value as well; otherwise, a memory-full error is returned. Use the `OCEPackDSSpecSize` function to obtain the size of a `PackedDSSpec` structure.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0322

RESULT CODES

noErr	0	No error
memFullErr	–108	The PackedDSSpec structure is not large enough to hold all of the packed information

SEE ALSO

The DSSpec structure is defined on page 2-36.

The PackedDSSpec structure is defined on page 2-37.

To obtain the size of a PackedDSSpec structure, use the OCEPackDSSpecSize function on page 2-96.

For information on unpacking a PackedDSSpec structure, see the OCEUnpackDSSpec function, described next.

OCEUnpackDSSpec

The OCEUnpackDSSpec function unpacks a PackedDSSpec structure.

```
pascal void OCEUnpackDSSpec(const PackedDSSpec *pdss, DSSpec *dss,
                            RecordID *rid);
```

pdss	A pointer to the PackedDSSpec structure you want to unpack.
dss	A pointer to a DSSpec structure. You must allocate this structure.
rid	A pointer to a RecordID structure. The OCEUnpackDSSpec function extracts the RecordID information from the PackedDSSpec structure and places it in this RecordID structure. You must allocate this structure.

DESCRIPTION

The OCEUnpackDSSpec function extracts the information from a PackedDSSpec structure and places it in a DSSpec structure and a RecordID structure. The OCEUnpackDSSpec function extracts the record identifier (if any) into the RecordID structure, places the rest of the information into the DSSpec structure, and then sets the entitySpecifier field of the DSSpec structure to point to the RecordID structure. The OCEUnpackDSSpec function returns a pointer to the extension (if any) in the extensionValue field of the DSSpec structure, and returns the length of that extension in the extensionSize field of the DSSpec structure. If there is no extension, the OCEUnpackDSSpec function sets the extensionValue field of the DSSpec structure to nil.

SPECIAL CONSIDERATIONS

The unpacked `DSSpec` and `RecordID` structures contain pointers into the `PackedDSSpec` structure. You should not delete or reuse the `PackedDSSpec` structure until you have finished using the `DSSpec` and `RecordID` structures as well.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$032F

SEE ALSO

The `DSSpec` structure is defined on page 2-36.

The `PackedDSSpec` structure is defined on page 2-37.

The `RecordID` structure is defined on page 2-34.

To pack a `DSSpec` structure, use the `OCEPackDSSpec` function (page 2-97).

OCEEqualDSSpec

The `OCEEqualDSSpec` function checks the equality of two `DSSpec` structures.

```
pascal Boolean OCEEqualDSSpec(const DSSpec *pdss1,
                              const DSSpec *pdss2);
```

`pdss1` A pointer to the first `DSSpec` structure you want to compare.

`pdss2` A pointer to the second `DSSpec` structure you want to compare.

DESCRIPTION

Given two `DSSpec` structures pointed to by the `pdss1` and `pdss2` parameters, the `OCEEqualDSSpec` function compares them for equality and returns `true` if they are equal and `false` if they are not.

This function checks the two `DSSpec` structures for equality in the following manner:

- If both pointers to the `DSSpec` structures are `nil`, then they are equal.
- If one of the pointers to a `DSSpec` structure is `nil` and the other is not, then the two `DSSpec` structures are not equal.
- If neither pointer to the `DSSpec` structures is `nil`, then the `CreationID` structures, identified by the `RecordID->LocalRecordID.cid` fields of the two `DSSpec` structures, are compared for equality by calling the `OCEEqualCreationID` function (page 2-52). If the `OCEEqualCreationID` function returns `false`, then the two `DSSpec` structures are not equal.

- If the `CreationID` structures are equal, then the `PackedRLI` structures pointed to by the `RecordID->PackedRLIPtr` fields of the two `DSSpec` structures are compared for equality by calling the `OCEEqualPackedRLI` function (page 2-76). If the `OCEEqualPackedRLI` function returns `false`, then the two `DSSpec` structures are not equal.
- If the `PackedRLI` structures are equal, then the `LocalRecordID.recordName` and `LocalRecordID.recordType` fields belonging to the `RecordID` structure of the two `DSSpec` structures are compared for equality by calling the `OCEEqualRString` (page 2-50) function and passing it the proper `RStringKind` value for each field. If the `OCEEqualRString` function returns `false`, the two `DSSpec` structures are not equal.
- If the `LocalRecordID.recordName` and `LocalRecordID.recordType` fields are equal then the values of the `extensionType` fields of the `DSSpec` structures are examined. If they are not identical then the `DSSpec` structures are not equal.
- If the `extensionType` fields of the two `DSSpec` structures are identical, then the `extensionSize` fields of the `DSSpec` structures are compared. If they are not identical, then the two `DSSpec` structures are not equal.
- If the `extensionSize` fields of the two `DSSpec` structures are identical, then the `extensionValue` fields of the `DSSpec` structures are compared. They are compared byte by byte for equality, and if they are not identical then the two `DSSpec` structures are not equal.

If the conditions for equality listed above are satisfied, then the two `DSSpec` structures are equal; otherwise, they are not equal.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$030E

SEE ALSO

The `DSSpec` data structure is defined on page 2-36.

To compare two `PackedDSSpec` structures for equality use the `OCEEqualPackedDSSpec` function, described next.

OCEEqualPackedDSSpec

The `OCEEqualPackedDSSpec` function checks the equality of two `PackedDSSpec` structures.

```
pascal Boolean OCEEqualPackedDSSpec(const PackedDSSpec *pdss1,
                                    const PackedDSSpec *pdss2);
```

pdss1	A pointer to the first PackedDSSpec structure you want to compare.
pdss2	A pointer to the second PackedDSSpec structure you want to compare.

DESCRIPTION

Given two PackedDSSpec structures pointed to by the pdss1 and pdss2 parameters, the OCEEqualPackedDSSpec function compares them for equality and returns true if they are equal, and false if they are not.

This function checks the two PackedDSSpec structures for equality in the following manner:

- If both pointers to the PackedDSSpec structures are nil, then they are equal.
- If one of the pointers to a PackedDSSpec structure to nil and the other is not, then the two PackedDSSpec structures are not equal.
- If neither pointer to the PackedDSSpec structures is nil, then the two structures are unpacked and the CreationID structures, identified by the RecordID->LocalRecordID.cid fields of the two DSSpec structures, are compared for equality by calling the OCEEqualCreationID function (page 2-52). If the OCEEqualCreationID function returns false, then the two PackedDSSpec structures are not equal.
- If the CreationID structures are equal, then the PackedRLI structures pointed to by the RecordID->PackedRLIPtr fields of the two unpacked PackedDSSpec structures are compared for equality by calling the OCEEqualPackedRLI function (page 2-76). If the OCEEqualPackedRLI function returns false, then the two PackedDSSpec structures are not equal.
- If the PackedRLI structures are equal, then the LocalRecordID.recordName and LocalRecordID.recordType fields belonging to the RecordID structure of the two unpacked PackedDSSpec structures are compared for equality by calling the OCEEqualRString (page 2-50) function and passing it the proper RStringKind value for each field. If the OCEEqualRString function returns false, the two PackedDSSpec structures are not equal.
- If the LocalRecordID.recordName and LocalRecordID.recordType fields belonging to the RecordID structure of the two unpacked PackedDSSpec structures are equal then the values of the extensionType fields of the PackedDSSpec structures are examined. If they are not identical then the PackedDSSpec structures are not equal.
- If the extensionType fields of the two unpacked PackedDSSpec structures are identical, then the extensionSize fields of the unpacked PackedDSSpec structures are compared. If they are not identical, then the two PackedDSSpec structures are not equal.
- If the extensionSize fields of the two unpacked PackedDSSpec structures are identical, then the extensionValue fields of the unpacked PackedDSSpec structures are compared. They are compared byte by byte for equality, and if they are not identical then the two PackedDSSpec structures are not equal.

If the conditions for equality listed above are satisfied, then the two PackedDSSpec structures are equal; otherwise, they are not equal.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0310

SEE ALSO

The PackedDSSpec data structure is defined on page 2-37.

The RStringKind data structure is defined on page 2-24.

To compare two DSSpec structures for equality use the OCEEqualDSSpec function, described on page 2-99.

OCEValidPackedDSSpec

The OCEValidPackedDSSpec function checks the validity of a PackedDSSpec structure.

```
pascal Boolean OCEValidPackedDSSpec(const PackedDSSpec *pdss);
```

pdss A pointer to the PackedDSSpec that you want to verify.

DESCRIPTION

The OCEValidPackedDSSpec function examines a PackedDSSpec structure to ensure validity for its particular type and returns true if it is valid, false if it is not.

The OCEValidPackedDSSpec function determines validity for a PackedDSSpec structure in the following manner:

- If the pointer to the PackedDSSpec structure is nil, then the PackedDSSpec structure is invalid.
- If the length of the PackedDSSpec structure is 0, then the PackedDSSpec structure is valid.
- If the pointer to the PackedDSSpec structure is not nil, and the length of the PackedDSSpec structure is greater than 0, then the PackedDSSpec structure is unpacked and its extensionType field is examined for validity. If the extensionType field of the PackedDSSpec has a value of 'entn', then the OCEValidPackedDSSpec function checks to make sure that the PackedDSSpec structure contains a valid entitySpecifier field by calling the OCEValidPackedRecordID function (page 2-93). If the OCEValidPackedRecordID function returns false, the PackedDSSpec structure is not valid.
- If the extensionType field does not have a value of 'entn' and it is not nil, then the RecordID field of the PackedDSSpec is examined to see if it contains an RLI component. If it does, then the RLI structure is checked for validity by calling the OCEValidRLI function. If the OCEValidRLI function returns false, then the

PackedDSSpec structure is invalid. The CreationID structure, identified by the RecordID->LocalRecordID.cid field of the unpacked PackedDSSpec structure, is not tested for validity.

- If the RLI component of the PackedDSSpec is valid, then the LocalRecordID.recordName and LocalRecordID.recordType fields of the RecordID component of the PackedDSSpec structure are examined for validity by calling the OCEValidRString function (page 2-51). If the OCEValidRString function returns false, then the PackedDSSpec is invalid.

If all of the conditions for validity are satisfied, then the PackedDSSpec structure is valid; otherwise, it is not.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
__OCEUtils	$0333

SEE ALSO

The PackedDSSpec data structure is defined on page 2-37.

OCEGetDSSpecInfo

The OCEGetDSSpecInfo function returns information about a DSSpec structure.

```
pascal OSType OCEGetDSSpecInfo(const DSSpec *spec);
```

spec A pointer to the DSSpec structure you want to get information about.

DESCRIPTION

The OCEGetDSSpecInfo function returns certain information about the specific DSSpec structure you pass to it. If the DSSpec structure does not have an entity specifier, it is invalid, that is, it returns kOCEInvalidDSSpec. If it does have an entity specifier, then it must have an extension type value of 'entn'; otherwise, it is invalid.

If the DSSpec structure has no extension, the OCEGetDSSpecInfo function determines whether it represents the root of all catalogs, a single catalog, a catalog node, or a record. If it has no extension and is not any of these types, it is considered invalid. If the DSSpec structure does have an extension, this function simply returns the extension type. The OCEGetDSSpecInfo function only performs the rudimentary checks just described. It does not do a complete check of the DSSpec structure for validity. Call the OCEValidPackedDSSpec function (page 2-102) to check a PackedDSSpec structure for validity.

The values that are returned by the `OCEGetDSSpecInfo` function are described in this enumerated list:

```
enum /* OCEGetDSSpecInfo types */
{
   kOCEInvalidDSSpec= '0x3F3F3F3FL',/* could not be determined */
   kOCEDirsRootDSSpec= 'root',      /* root of all catalogs
                                       ("Catalog" icon) */
   kOCEDirectoryDSSpec= 'dire',     /* catalog */
   kOCEDNodeDSSpec= 'dnod',         /* dNode */
   kOCERecordDSSpec= 'reco',        /* record */
   kOCEentnDSSpec= 'entn',          /* extensionType is 'entn' */
   kOCENOTentnDSSpec= 'not '        /* extensionType is
                                          not 'entn' */
};
```

Field descriptions

`kOCEInvalidDSSpec`
The type does not conform to any known type.

`kOCEDirsRootDSSpec`
The `DSSpec` structure represents the root of all catalogs.

`kOCEDirectoryDSSpec`
The `DSSpec` structure represents a catalog.

`kOCEDNodeDSSpec`
The `DSSpec` structure represents a catalog node.

`kOCERecordDSSpec`
The `DSSpec` structure represents a record.

`kOCEentnDSSpec`
The extension type of the `DSSpec` structure is `'entn'`.

`kOCENOTentnDSSpec`
The `entitySpecifier` field of the `DSSpec` structure is not `nil` and the extension type of the `DSSpec` structure is not `'entn'`.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0319

SEE ALSO

The `DSSpec` data structure is defined on page 2-36.

To obtain the extension type of a `DSSpec` structure, use the `OCEGetExtensionType` function, described next.

OCEGetExtensionType

The `OCEGetExtensionType` function returns the extension type embedded in a `PackedDSSpec` structure.

```
pascal OSType OCEGetExtensionType(const PackedDSSpec *pdss);
```

`pdss` A pointer to a `PackedDSSpec` structure from which you want to retrieve the extension type.

DESCRIPTION

Given a pointer to a `PackedDSSpec` structure, the `OCEGetExtensionType` function extracts the extension type of the `PackedDSSpec` structure and returns it to you.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$031C

SEE ALSO

The `DSSpec` data structure is defined on page 2-36.

To obtain information about a `DSSpec` structure, see the `OCEGetDSSpecInfo` function on page 2-103.

OCEStreamPackedDSSpec

The `OCEStreamPackedDSSpec` function takes a `DSSpec` structure and converts it from a pointer-based structure into a stream of bytes.

```
pascal OSErr OCEStreamPackedDSSpec(const DSSpec *dss,
                                   MyDSSpecStreamer stream,
                                   long userData,
                                   unsigned long *actualCount);
```

`dss` A pointer to the `DSSpec` structure you want to process.

`stream` A pointer to a function that you supply.

`userData` Data supplied by you that is passed on to your `stream` function. The `userData` parameter can contain anything your particular stream method needs.

`actualCount` A pointer to the total number of bytes (streamed out) by the `OCEStreamPackedDSSpec` function.

DESCRIPTION

The `OCEStreamPackedDSSpec` function converts a `DSSpec` structure into a stream of bytes by calling the `stream` function that you provide. You can use this function whenever you want to write the contents of a `DSSpec` structure as a series of bytes to a file, into a buffer in memory, or any other place.

The `stream` function that you provide contains the specific code that writes out the data. The `OCEStreamPackedDSSpec` function calls your `stream` function repeatedly and passes your function the current portion of the data that needs to be streamed, the length of this data, an `eof` flag that is set by the `OCEStreamPackedDSSpec` function if this is the last of the data to be streamed, and a parameter containing any application-specific data that you define. For example, if you were writing a `stream` function that wrote out a `DSSpec` structure to a file on a hard disk, you might want to store a pointer in the `userData` parameter to a block of data that contains such information as the filename and size of the file.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$033D

RESULT CODES

`noErr`	0	No error

SEE ALSO

The `DSSpec` data structure is defined on page 2-36.

The callback function `MyDSSpecStreamer` is described next.

Application-Defined Functions

This section describes a callback function that you supply to `OCEStreamPackedDSSpec`. See the section "Application-Defined Functions" in the chapter "Catalog Manager" in this book for more information on how AOCE callback functions operate.

MyDSSpecStreamer

The `MyDSSpecStreamer` function provides a method for processing data from the `OCEStreamPackedDSSpec` function.

```
typedef pascal OSErr (*MyDSSpecStreamer)(void *buffer,
                    unsigned long count, Boolean eof ,
                    long userData);
```

buffer	A pointer to the data that your streamer method processes. This is supplied by the OCEStreamPackedDSSpec function each time it calls your MyDSSpecStreamer function.
count	The length, in bytes, of the current data in the buffer.
eof	A flag that is set by the OCEStreamPackedDSSpec function the last time that it calls your MyDSSpecStreamer function. This flag informs you that when the OCEStreamPackedDSSpec function finishes processing the data currently in the buffer, it will have completed processing the DSSpec structure.
userData	The data that you supply in the userData parameter to the OCEStreamPackedDSSpec function. This is passed directly to your MyDSSpecStreamer function.

DESCRIPTION

The MyDSSpecStreamer function is called by the OCEStreamPackedDSSpec function (page 2-105) to process the data from a DSSpec structure in discreet segments. You write this routine to process the data in the way that you want. The OCEStreamPackedDSSpec function calls your MyDSSpecStreamer function various times and passes your function progress information as well as the current portion of the DSSpec to process. Any errors returned by this function are passed on to the OCEStreamPackedDSSpec function.

SEE ALSO

The DSSpec data structure is defined on page 2-36.

The OCEStreamPackedDSSpec function is defined on page 2-105.

Summary of the AOCE Utilities

C Summary

Constants and Data Types

```
/* OCE String Constants */
#define RStringHeader \
   CharacterSet charSet;\
   unsigned short dataLength;

enum {
   kRString32Size          = 32,       /* max size of RString32 */
   kRString64Size          = 64,       /* max size of RString64 */
   kNetworkSpecMaxBytes    = 32,       /* max size of NetworkSpec */
   kPathNameMaxBytes       = 1024,     /* max size of PackedPathName */
   kDirectoryNameMaxBytes  = 32,       /* max size of DirectoryName */
   kAttributeTypeMaxBytes  = 32,       /* max size of AttributeType */
   kAttrValueMaxBytes      = 65536,    /* max size of any attribute value */
   kRStringMaxBytes        = 256,      /* max size of recordName or
                                          recordType */
   kRStringMaxChars        = 128       /* max # of chars in recordName
                                          RString, or recordType */
};

#define kMinPackedRStringLength (sizeof (ProtoRString))

/* RStringKind Values */
enum {
   kOCEDirName             = 0,        /* RString is a Catalog Name */
   kOCERecordOrDNodeName   = 1,        /* RString is a recordName or
                                          catalog node name */
   kOCERecordType          = 2,        /* RString is a recordType */
   kOCENetworkSpec         = 3,        /* RString is a NetworkSpec */
   kOCEAttrType            = 4,        /* RString is an AttributeType */
   kOCEGenericSensitive    = 5,        /* RString is a case and diacritical
                                          mark sensitive generic string */
```

```
   kOCEGenericInsensitive  = 6         /* RString is a case and diacritical
                                          mark insensitive generic string */
};

/* OCEDirectoryKind Values */
enum {
   kDirAllKinds            = 0,        /* All catalog types */
   kDirADAPKind            = 'adap',   /* an PowerShare catalog */
   kDirPersonalDirectoryKind
                           = 'pdir',   /* a personal catalog */
   kDirDSAMKind            = 'dsam'    /* catalog service access module */
}

/* Catalog Node Constants */
enum {
   kNULLDNodeNumber        = 0,        /* none specified */
   kRootDNodeNumber        = 2         /* the root of the tree */
};

/* Values returned by OCEGetDSSpecInfo() */
enum {
   kOCEInvalidDSSpec       = 0x3F3F3F3FL,/* '????' could not be
                                            determined */
   kOCEDirsRootDSSpec      = 'root',   /* root of all catalogs
                                          ("Catalog" icon) */
   kOCEDirectoryDSSpec     = 'dire',   /* catalog */
   kOCEDNodeDSSpec         = 'dnod',   /* Dnode */
   kOCERecordDSSpec        = 'reco',   /* record */
   kOCEentnDSSpec          = 'entn',   /* extensionType is 'entn' */
   kOCENOTentnDSSpec       = 'not '    /* extensionType is not 'entn' */
};

/* AttributeTag values */
enum {
   typeRString             = 'rstr',   /* attribute value is an RString */
   typePackedDSSpec        = 'dspc',   /* attribute value is a DSSpec */
   typeBinary              = 'bnry'    /* attribute value is a sequence
                                          of bytes */
};
```

```
/* Cluster info */
enum {
   kcanContainRecordsBit,              /* a cluster */
   kForeignNodeBit                     /* a foreign catalog */
};

/* DirNodeKind */
enum {
   kcanContainRecords= 1L<<kcanContainRecordsBit,
   kForeignNode= 1L<<kForeignNodeBit
};

/* RLI Constants */

#define kMinPackedRLISize (sizeof (ProtoPackedRLI) + \
               sizeof (DirDiscriminator) + sizeof (DNodeNum) +\
               kMinPackedRStringLength + sizeof (ProtoPackedPathName))

#define kRLIMaxBytes (sizeof (RString) + sizeof (DirDiscriminator) + \
                        sizeof (DNodeNum) + kPathNameMaxBytes)

#define PackedRLIHeader unsigned short dataLength /* number of bytes
                                                     in data field */

/* RecordID Constants */

#define kPackedRecordIDMaxBytes (kPathNameMaxBytes + sizeof (DNodeNum) + \
                        sizeof (DirDiscriminator) + sizeof (CreationID) + \
                        (3 * sizeof (RString)))

#define PackedRecordIDHeader unsigned short dataLength /* length of data field
                                          in the PackedRecordID structure */

/* DSSpec Constants */

#define kPackedDSSpecMaxBytes(sizeof (PackedRecordID) + sizeof (OSType) + \
         sizeof (unsigned short))

#define PackedDSSpecHeader unsigned short dataLength;

/* Indices for the standard definitions for standard record types */

#define  kUserRecTypeNum               1     /* "User" */
#define  kGroupRecTypeNum              2     /* "Group" */
#define  kMnMRecTypeNum                3     /* "AppleMail™ M&M" */
#define  kMnMForwarderRecTypeNum       4     /* "AppleMail™ Fwdr" */
```

```
#define  kNetworkSpecRecTypeNum        5     /* "NetworkSpec" */
#define  kADAPServerRecTypeNum         6     /* "PowerShare Server" */
#define  kADAPDNodeRecTypeNum          7     /* "PowerShare DNode" */
#define  kADAPDNodeRepRecTypeNum       8     /* "PowerShare DNode Rep" */
#define  kServerSetupRecTypeNum        9     /* "Server Setup" */
#define  kDirectoryRecTypeNum          10    /* "Catalog" */
#define  kDNodeRecTypeNum              11    /* "DNode" */
#define  kSetupRecTypeNum              12    /* "Setup" */
#define  kMSAMRecTypeNum               13    /* "MSAM" */
#define  kDSAMRecTypeNum               14    /* "CSAM" */
#define  kAttributeValueRecTypeNum     15    /* "Attribute Value" */
#define  kBusinessCardRecTypeNum       16    /* "Business Card" */
#define  kMailServiceRecTypeNum        17    /* "Mail Service" */
#define  kCombinedRecTypeNum           18    /* "Combined" */
#define  kOtherServiceRecTypeNum       19    /* "Other Service" */
#define  kAFPServiceRecTypeNum         20    /* "Other Service afps" */

#define  kFirstOCERecTypeNum kUserRecTypeNum       /* first standard OCE
                                                      record type */

#define  kLastOCERecTypeNum kAFPServiceRecTypeNum/* last standard OCE
                                                      record type */

#define  kNumOCERecTypes      (kLastOCERecTypeNum - kFirstOCERecTypeNum + 1)

/* Indices for the standard definitions for standard attribute types
(OCEAttributeTypeIndex): */

#define  kMemberAttrTypeNum            1001  /* "Member" */
#define  kAdminsAttrTypeNum            1002  /* "Administrators" */
#define  kMailSlotsAttrTypeNum         1003  /* "mailslots" */
#define  kPrefMailAttrTypeNum          1004  /* "pref mailslot" */
#define  kAddressAttrTypeNum           1005  /* "Address" */
#define  kPictureAttrTypeNum           1006  /* "Picture" */
#define  kAuthKeyAttrTypeNum           1007  /* "auth key" */
#define  kTelephoneAttrTypeNum         1008  /* "Telephone" */
#define  kNBPNameAttrTypeNum           1009  /* "NBP Name" */
#define  kQMappingAttrTypeNum          1010  /* "ForwarderQMap" */
#define  kDialupSlotAttrTypeNum        1011  /* "DialupSlotInfo" */
#define  kHomeNetAttrTypeNum           1012  /* "Home Internet" */
#define  kCoResAttrTypeNum             1013  /* "Co-resident M&M" */
#define  kFwdrLocalAttrTypeNum         1014  /* "FwdrLocalRecord" */
#define  kConnectAttrTypeNum           1015  /* "Connected To" */
#define  kForeignAttrTypeNum           1016  /* "Foreign RLIs" */
```

```
#define  kOwnersAttrTypeNum            1017  /* "Owners" */
#define  kReadListAttrTypeNum          1018  /* "ReadList" */
#define  kWriteListAttrTypeNum         1019  /* "WriteList" */
#define  kDescriptorAttrTypeNum        1020  /* "Descriptor" */
#define  kCertificateAttrTypeNum       1021  /* "Certificate" */
#define  kMsgQsAttrTypeNum             1022  /* "MessageQs" */
#define  kPrefMsgQAttrTypeNum          1023  /* "PrefMessageQ" */
#define  kMasterPFAttrTypeNum          1024  /* "MasterPF" */
#define  kMasterNetSpecAttrTypeNum     1025  /* "MasterNetSpec" */
#define  kServersOfAttrTypeNum         1026  /* "Servers Of" */
#define  kParentCIDAttrTypeNum         1027  /* "Parent CID" */
#define  kNetworkSpecAttrTypeNum       1028  /* "NetworkSpec" */
#define  kLocationAttrTypeNum          1029  /* "Location" */
#define  kTimeSvrTypeAttrTypeNum       1030  /* "TimeServer Type" */
#define  kUpdateTimerAttrTypeNum       1031  /* "Update Timer" */
#define  kShadowsOfAttrTypeNum         1032  /* "Shadows Of" */
#define  kShadowServerAttrTypeNum      1033  /* "Shadow Server" */
#define  kTBSetupAttrTypeNum           1034  /* "TB Setup" */
#define  kMailSetupAttrTypeNum         1035  /* "Mail Setup" */
#define  kSlotIDAttrTypeNum            1036  /* "SlotID" */
#define  kGatewayFileIDAttrTypeNum     1037  /* "Gateway FileID" */
#define  kMailServiceAttrTypeNum       1038  /* "Mail Service" */
#define  kStdSlotInfoAttrTypeNum       1039  /* "Std Slot Info" */
#define  kAssoDirectoryAttrTypeNum     1040  /* "Asso. Catalog" */
#define  kDirectoryAttrTypeNum         1041  /* "Catalog" */
#define  kDirectoriesAttrTypeNum       1042  /* "Catalogs" */
#define  kSFlagsAttrTypeNum            1043  /* "SFlags" */
#define  kLocalNameAttrTypeNum         1044  /* "Local Name" */
#define  kLocalKeyAttrTypeNum          1045  /* "Local Key" */
#define  kDirUserRIDAttrTypeNum        1046  /* "Dir User RID" */
#define  kDirUserKeyAttrTypeNum        1047  /* "Dir User Key" */
#define  kDirNativeNameAttrTypeNum     1048  /* "Dir Native Name" */
#define  kCommentAttrTypeNum           1049  /* "Comment" */
#define  kRealNameAttrTypeNum          1050  /* "Real Name" */
#define  kPrivateDataAttrTypeNum       1051  /* "Private Data" */
#define  kDirTypeAttrTypeNum           1052  /* "Catalog Type" */
#define  kDSAMFileAliasAttrTypeNum     1053  /* "CSAM File Alias" */
#define  kCanAddressToAttrTypeNum      1054  /* "Can Address To" */
#define  kDiscriminatorAttrTypeNum     1055  /* "Discriminator" */
#define  kAliasAttrTypeNum             1056  /* "Alias" */
#define  kParentMSAMAttrTypeNum        1057  /* "Parent MSAM" */
#define  kParentDSAMAttrTypeNum        1058  /* "Parent CSAM" */
#define  kSlotAttrTypeNum              1059  /* "Slot" */
```

```
#define  kAssoMailServiceAttrTypeNum   1060  /* "Asso. Mail Service" */
#define  kFakeAttrTypeNum              1061  /* "Fake" */
#define  kInheritSysAdminAttrTypeNum   1062  /* "Inherit SysAdministrators"*/
#define  kPreferredPDAttrTypeNum       1063  /* "Preferred PD" */
#define  kLastLoginAttrTypeNum         1064  /* "Last Login" */
#define  kMailerAOMStateAttrTypeNum    1065  /* "Mailer AOM State" */
#define  kMailerSendOptionsAttrTypeNum 1066  /* "Mailer Send Options" */
#define  kJoinedAttrTypeNum            1067  /* "Joined" */
#define  kUnconfiguredAttrTypeNum      1068  /* "Unconfigured" */
#define  kVersionAttrTypeNum           1069  /* "Version" */
#define  kLocationNamesAttrTypeNum     1070  /* "Location Names" */
#define  kActiveAttrTypeNum            1071  /* "Active" */
#define  kDeleteRequestedAttrTypeNum   1072  /* "Delete Requested" */
#define  kGatewayTypeAttrTypeNum       1073  /* "Gateway Type" */

#define  kFirstOCEAttrTypeNum kMemberAttrTypeNum/* first standard OCE
                                                   attribute type */

#define  kLastOCEAttrTypeNum kGatewayTypeAttrTypeNum/* last standard OCE
                                                       attribute type */

#define  kNumOCEAttrTypes (kLastOCEAttrTypeNum - kFirstOCEAttrTypeNum + 1)
                                                  /* the total number of
                                                   attributes */

/* OCE String Types */

typedef short CharacterSet;            /* script code info */

struct RString                         /* RString */
{
   RStringHeader
   Byte  body[kRStringMaxBytes];
};

typedef struct RString RString;
typedef RString *RStringPtr, **RStringHandle;

struct RString64                       /* RString64 */
{
   RStringHeader
   Byte     body[kRString64Size];
};

typedef struct RString64 RString64;
```

```
struct RString32                        /* RString32 */
{
   RStringHeader
   Byte     body[kRString32Size];
};

typedef struct RString32 RString32;

struct ProtoRString                     /* ProtoRString */
{
   RStringHeader
   /* The body for the ProtoRstring should be defined here */
};

typedef struct ProtoRString ProtoRString;
typedef ProtoRString *ProtoRString;

struct DirectoryName                    /* DirectoryName */
{
   RStringHeader
   Byte     body[kDirectoryNameMaxBytes];
};

typedef struct DirectoryName DirectoryName;
typedef DirectoryName *DirectoryNamePtr;

struct NetworkSpec                      /* NetworkSpec */
{
   RStringHeader
   Byte  body[kNetworkSpecMaxBytes];
};

typedef struct NetworkSpec NetworkSpec;
typedef NetworkSpec *NetworkSpecPtr;

typedef unsigned short RStringKind;

/* RecordID Types */
struct CreationID
{
   unsigned long  source;  /* private to a catalog.*/
   unsigned long  seq;     /* private to a catalog*/
};

typedef struct CreationID CreationID;
```

```
typedef CreationID AttributeCreationID;

struct LocalRecordID
{
   CreationID  cid;              /* creation ID of the record */
   RStringPtr  recordName;       /* name of the record */
   RStringPtr  recordType;       /* type of record */
};

typedef struct LocalRecordID LocalRecordID;
typedef LocalRecordID *LocalRecordIDPtr;

struct PackedPathName
{
   unsigned short dataLength;    /* number of bytes in data field */
   Byte           data[kPathNameMaxBytes - sizeof (unsigned short)];
};

typedef struct PackedPathName PackedPathName;
typedef PackedPathName *PackedPathNamePtr;

struct ProtoPackedPathName {

   unsigned short dataLength;

/* Followed by data */

};

typedef struct ProtoPackedPathName ProtoPackedPathName;
typedef ProtoPackedPathName *ProtoPackedPathNamePtr;

struct DirDiscriminator {
   OCEDirectoryKind  signature;     /* private to catalog */
   unsigned long     misc;          /* private to catalog */
};

typedef struct DirDiscriminator DirDiscriminator;

typedef unsigned long   DNodeNum;

struct RLI {
   DirectoryNamePtr     directoryName;
   DirDiscriminator     discriminator;
```

```
   DNodeNum             dNodeNumber;
   PackedPathNamePtr    path;
};

typedef struct RLI RLI;
typedef RLI *RLIPtr;

struct PackedRLI {
   dataLength;
   Byte           data[kRLIMaxBytes];  /* packed record
                                          location info */
};

typedef struct PackedRLI PackedRLI;
typedef PackedRLI *PackedPLIPtr;

struct ProtoPackedRLI {
   dataLength
/* Followed by data */
};

typedef struct ProtoPackedRLI ProtoPackedRLI;
typedef ProtoPackedRLI *ProtoPackedRLIPtr;

struct RecordID {
   PackedRLIPtr      rli;     /* identifies record's catalog
                                 and dNode */
   LocalRecordID     local;   /* identifies record within
                                 its dNode */
};

typedef struct RecordID RecordID;
typedef RecordID *RecordIDPtr;

struct PackedRecordID {
   dataLength;                          /* length of data field */
   Byte  data[kPackedRecordIDMaxBytes];/* packed record ID */
};

typedef struct PackedRecordID PackedRecordID;
typedef PackedRecordID *PackedRecordIDPtr;
```

```
struct ShortRecordID
{
   PackedRLIPtr rli;
   CreationID cid;
};

typedef struct ShortRecordID ShortRecordID;
typedef ShortRecordID *ShortRecordIDPtr;

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

typedef struct DSSpec DSSpec;
typedef DSSpec *DSSpecPtr;

struct PackedDSSpec {
   dataLength
   Byte        data[kPackedDSSpecMaxBytes];
};

typedef struct PackedDSSpec PackedDSSpec;
typedef PackedDSSpec *PackedDSSpecPtr;

struct ProtoPackedDSSpec {
   dataLength
/* Followed by data */
};

typedef struct ProtoPackedDSSpec ProtoPackedDSSpec;
typedef ProtoPackedDSSpec *ProtoPackedDSSpecPtr;

/* Attribute Structures */
struct AttributeType
{
   RStringHeader
   Byte  body[kAttributeTypeMaxBytes];
};

typedef struct AttributeType AttributeType;
typedef AttributeType *AttributeTypePtr;
```

```
struct AttributeValue {
   AttributeTag   tag;        /* format of attribute value */
   unsigned long  dataLength; /* # of bytes in attribute value */
   Ptr            bytes;      /* points to attribute value data */
};

typedef struct AttributeValue AttributeValue;
typedef AttributeValue *AttributeValuePtr;

typedef CreationID   AttributeCreationID;

struct Attribute {
   AttributeType        attributeType; /* type of the attribute */
   AttributeCreationID  cid;           /* the creationID of the
                                          attribute */
   AttributeValue       value;         /* the attribute value */
};

typedef struct Attribute Attribute;
typedef Attribute *AttributePtr;

typedef DescType AttributeTag;/* same type used in AppleEvents */

/* recordType index */

typedef unsigned short OCERecordTypeIndex;

/* AttributeType index */

typedef unsigned short OCEAttributeTypeIndex;

/* OCE Catalog Types */

typedef unsigned long OCEDirectoryKind;

/* OCE Catalog Node Types */

typedef unsigned long DirNodeKind;
```

AOCE Utility Functions

AOCE String Functions

```
pascal OSErr OCECopyRString
                           (const RString *str1, RString *str2, unsigned
                            short str2Length);
```

```
pascal void OCECToRString   (const char *cStr, CharacterSet charSet,RString
                             *rStr,unsigned short rStrLength);
pascal void OCEPToRString   (ConstStr255Param pStr,CharacterSet charSet,
                             RString *rStr,unsigned short rStrLength);
pascal StringPtr OCERToPString
                            (const RString *rStr);
pascal short OCERelRString  (const void *str1, const void *str2,RStringKind
                             kind);
pascal Boolean OCEEqualRString
                            (const void *str1,const void *str2,RStringKind
                             kind);
pascal Boolean OCEValidRString
                            (const void *str,RStringKind kind);
```

Creation Identifier Functions

```
pascal Boolean OCEEqsrualCreationID
                            (const CreationID *cid1,
                             const CreationID *cid2);
pascal void OCECopyCreationID
                            (const CreationID *cid1,CreationID *const cid2);
pascal const CreationID     *OCENullCID(void);
pascal const CreationID     *OCEPathFinderCID(void);
pascal void OCESetCreationIDtoNull
                            (CreationID *const cid);
```

Packed Pathname Functions

```
pascal OSErr OCECopyPackedPathName
                            (const PackedPathName *path1,PackedPathName
                             *path2,unsigned short path2Length);
pascal Boolean OCEIsNullPackedPathName
                            (const PackedPathName *path);
pascal unsigned short OCEPackedPathNameSize
                            (const RStringPtr parts[], const unsigned short
                             nParts);
pascal unsigned short OCEDNodeNameCount
                            (const PackedPathName *path);
pascal unsigned short OCEUnpackPathName
                            (const PackedPathName *path, RString *const
                             parts[], const unsigned short nParts);
```

```
pascal OSErr OCEPackPathName
                            (const RStringPtr parts[],const unsigned short
                             nParts,PackedPathName *path,unsigned short
                             pathLength);
pascal Boolean OCEEqualPackedPathName
                            (const PackedPathName *path1, const
                             PackedPathName *path2);
pascal Boolean OCEValidPackedPathName
                            (const PackedPathName *path);
```

Catalog Discriminator Functions

```
pascal void OCECopyDirDiscriminator
                            (const DirDiscriminator *disc1,
                             DirDiscriminator *const disc2);
pascal Boolean OCEEqualDirDiscriminator
                            (const DirDiscriminator *disc1, const
                             DirDiscriminator *disc2);
```

Record Location Information Functions

```
pascal void OCENewRLI       (RLI *newRLI, const DirectoryName *dirName,
                             DirDiscriminator *discriminator,const DNodeNum
                             dNodeNumber,const PackedPathName *path);
pascal void OCEDuplicateRLI (const RLI *rli1, RLI *rli2);
pascal OSErr OCECopyRLI     (const RLI *rli1, RLI *rli2);
pascal Boolean OCEEqualRLI  (const RLI *rli1, const RLI *rli2);
pascal Boolean OCEValidRLI  (const RLI *theRLI);
pascal OSErr OCECopyPackedRLI
                            (const PackedRLI *prli1, PackedRLI
                             *prli2,unsigned short prli2Length);
pascal unsigned short OCEPackedRLISize
                            (const RLI *theRLI);
pascal OSErr OCEPackRLI     (const RLI *theRLI, PackedRLI *prli, unsigned
                             short prliLength);
pascal void OCEUnpackRLI    (const PackedRLI *prli, RLI *theRLI);
pascal unsigned short OCEPackedRLIPartsSize
                            (const DirectoryName *dirName, const RStringPtr
                             parts[], const unsigned short nParts);
```

```
pascal OSErr OCEPackRLIParts
                            (const DirectoryName *dirName, const
                             DirDiscriminator *discriminator, const
                             DNodeNum dNodeNumber, const RStringPtr
                             parts[], const unsigned short nParts,
                             PackedRLI *prli, unsigned short prliLength);
pascal Boolean OCEEqualPackedRLI
                            (const PackedRLI *prli1, const PackedRLI
                             *prli2);
pascal Boolean OCEValidPackedRLI
                            (const PackedRLI *prli);
pascal AliasPtr OCEExtractAlias
                            (const PackedRLI *prli);
pascal const PackedRLI      * OCEGetDirectoryRootPackedRLI (void)
```

Local Record Identifier Functions

```
pascal void OCENewLocalRecordID
                            (const RString *recordName, const RString
                             *recordType, const CreationID *cid,
                             LocalRecordID *lRID);
pascal OSErr OCECopyLocalRecordID
                            (const LocalRecordID *lRID1, LocalRecordID
                             *lRID2);
pascal Boolean OCEEqualLocalRecordID
                            (const LocalRecordID *lRID1, const
                             LocalRecordID *lRID2);
```

Short Record Identifier Functions

```
pascal void OCENewShortRecordID
                            (const PackedRLI *theRLI, const CreationID
                             *cid, ShortRecordIDPtr *sRID);
pascal OSErr OCECopyShortRecordID
                            (const ShortRecordID *sRID1,ShortRecordID
                             *sRID2);
pascal Boolean OCEEqualShortRecordID
                            (const ShortRecordID *sRID1,const ShortRecordID
                             *sRID2);
```

Record Identifier Functions

```
pascal RString *OCEGetIndRecordType
                            (const OCERecordTypeIndex stringIndex);
```

```
pascal void OCENewRecordID  (const PackedRLI *theRLI, const LocalRecordID
                             *lRID, RecordID *rid);
pascal OSErr OCECopyRecordID
                            (const RecordID *rid1,const RecordID *rid2);
pascal Boolean OCEEqualRecordID
                            (const RecordID *rid1,const RecordID *rid2);
```

Packed Record Identifier Functions

```
pascal OSErr OCECopyPackedRecordID
                            (const PackedRecordID *pRID1, const
                             PackedRecordID *pRID2, unsigned short
                             pRID2length);
pascal unsigned short OCEPackedRecordIDSize
                            (const RecordID *rid);
pascal OSErr OCEPackRecordID
                            (const RecordID *rid, PackedRecordID *pRID,
                             unsigned short packedRecordIDlength);
pascal void OCEUnpackRecordID
                            (const PackedRecordID *pRID, RecordID *rid);
pascal Boolean OCEEqualPackedRecordID
                            (const PackedRecordID *pRID1, const
                             PackedRecordID *pRID2);
pascal Boolean OCEValidPackedRecordID
                            (const PackedRecordID *pRID);
```

Attribute Type Functions

```
pascal AttributeType        *OCEGetIndAttributeType(const
                             OCEAttributeTypeIndex stringIndex);
```

Catalog Services Specification Functions

```
pascal OSErr OCECopyPackedDSSpec
                            (const PackedDSSpec *pdss1, const PackedDSSpec
                             *pdss2, unsigned short pdss2Length);
pascal unsigned short OCEPackedDSSpecSize
                            (const DSSpec *dss);
pascal OSErr OCEPackDSSpec  (const DSSpec *dss, PackedDSSpec *pdss,
                             unsigned short pdssLength);
pascal void OCEUnpackDSSpec (const PackedDSSpec *pdss, DSSpec *dss,
                             RecordID *rid);
pascal Boolean OCEEqualDSSpec
                            (const DSSpec *pdss1, const DSSpec *pdss2);
```

```
pascal Boolean OCEEqualPackedDSSpec
                            (const PackedDSSpec *pdss1, const PackedDSSpec
                             *pdss2);
pascal Boolean OCEValidPackedDSSpec
                            (const PackedDSSpec *pdss);
pascal OSType OCEGetDSSpecInfo
                            (const DSSpec *spec);
pascal OSType OCEGetExtensionType
                            (const PackedDSSpec *pdss);
pascal OSErr OCEStreamPackedDSSpec
                            (const DSSpec *dss,MyDSSpecStreamer stream,
                             long userData, unsigned long *actualCount);
```

Application-Defined Functions

```
typedef pascal OSErr        (*MyDSSpecStreamer)(void *buffer, unsigned long
                             count, Boolean eof, long userData);
```

Pascal Summary

Constants

```
CONST

{ OCE String Constants }
   kRString32Size             = 32;    { max size of RString32 }
   kRString64Size             = 64;    { max size of RString64 }
   kNetworkSpecMaxBytes       = 32;    { max size of NetworkSpec }
   kPathNameMaxBytes          = 1024;  { max size of PackedPathName }
   kDirectoryNameMaxBytes     = 32;    { max size of DirectoryName }
   kAttributeTypeMaxBytes     = 32;    { max size of AttributeType }
   kAttrValueMaxBytes         = 65536; { max size of any attribute value }
   kRStringMaxBytes           = 256;   { max bytes in recordName,recordType }
   kRStringMaxChars           = 128;   { max chars in recordName,recordType }

   kMinPackedRStringLength = sizeof(ProtoRString);

{ values of RStringKind }
   kOCEDirName             = 0;
   kOCERecordOrDNodeName   = 1;
   kOCERecordType          = 2;
   kOCENetworkSpec         = 3;
```

```
   kOCEAttrType            = 4;
   kOCEGenericSensitive    = 5;
   kOCEGenericInsensitive  = 6;

{ values of OCEDirectoryKind }
   kDirAllKinds            = 0;
   kDirADAPKind            = 'adap';
   kDirPersonalDirectoryKind
                           = 'pdir';
   kDirDSAMKind            = 'dsam';

{ Catalog Node Constants }
   kNULLDNodeNumber           = 0;        { none specified }
   kRootDNodeNumber           = 2;        { the root of the tree }

{ Values returned by OCEGetDSSpecInfo() }
   kOCEInvalidDSSpec       = '????',   { could not be determined }
   kOCEDirsRootDSSpec      = 'root',   { root of all catalogs }
   kOCEDirectoryDSSpec     = 'dire',   { catalog }
   kOCEDNodeDSSpec         = 'dnod',   { Dnode }
   kOCERecordDSSpec        = 'reco',   { record }
   kOCEentnDSSpec          = 'entn',   { extensionType is 'entn' }
   kOCENOTentnDSSpec       = 'not '    { extensionType is not 'entn' }

{ AttributeTag Values }
   typeRString             = 'rstr',   { attribute value is an RString }
   typePackedDSSpec        = 'dspc',   { attribute value is a DSSpec }
   typeBinary              = 'bnry'    { attribute value is a sequence
                                         of bytes }

{ Cluster info }
   kcanContainRecordsBit,  = 0{ a cluster }
   kForeignNodeBit         = 1{ a foreign catalog }

{ values of DirNodeKind }
   kcanContainRecords      = $00000001;{<<kcanContainRecordsBit}
   kForeignNode            = $00000002;{<<kForeignNodeBit}

{ RLI Constants }

   kRLIMaxBytes      = (sizeof (RString) + sizeof (DirDiscriminator) +
                        sizeof (DNodeNum) + kPathNameMaxBytes);
```

```
   kMinPackedRLISize  = (sizeof (ProtoPackedRLI) +
                         sizeof (DirDiscriminator) + sizeof (DNodeNum) +
                         kMinPackedRStringLength +
                         sizeof (ProtoPackedPathName));

{ RecordID Constants }
   kPackedRecordIDMaxBytes = kPathNameMaxBytes + sizeof(DNodeNum) +
      sizeof(DirDiscriminator) + sizeof(CreationID) + (3*sizeof(RString));

{ DSSpec Constants }

   kPackedDSSpecMaxBytes = (sizeof (PackedRecordID) + sizeof (OSType) +
                         sizeof (INTEGER));

   { Indices for the standard definitions for standard record types }

   kUserRecTypeNum                        = 1;  { "User" }
   kGroupRecTypeNum                       = 2;  { "Group" }
   kMnMRecTypeNum                         = 3;  { "AppleMail™ M&M" }
   kMnMForwarderRecTypeNum                = 4;  { "AppleMail™ Fwdr" }
   kNetworkSpecRecTypeNum                 = 5;  { "NetworkSpec" }
   kADAPServerRecTypeNum                  = 6;  { "PowerShare Server" }
   kADAPDNodeRecTypeNum                   = 7;  { "PowerShare DNode" }
   kADAPDNodeRepRecTypeNum                = 8;  { "PowerShare DNode Rep" }
   kServerSetupRecTypeNum                 = 9;  { "Server Setup" }
   kDirectoryRecTypeNum                   = 10; { "Catalog" }
   kDNodeRecTypeNum                       = 11; { "DNode" }
   kSetupRecTypeNum                       = 12; { "Setup" }
   kMSAMRecTypeNum                        = 13; { "MSAM" }
   kDSAMRecTypeNum                        = 14; { "CSAM" }
   kAttributeValueRecTypeNum              = 15; { "Attribute Value" }
   kBusinessCardRecTypeNum                = 16; { "Business Card" }
   kMailServiceRecTypeNum                 = 17; { "Mail Service" }
   kCombinedRecTypeNum                    = 18; { "Combined" }
   kOtherServiceRecTypeNum                = 19; { "Other Service" }
   kAFPServiceRecTypeNum                  = 20; { "Other Service afps" }

   kFirstOCERecTypeNum = kUserRecTypeNum;       { first standard OCE record
                                                  type }

   kLastOCERecTypeNum = kAFPServiceRecTypeNum;  { last standard OCE record
                                                  type }

   kNumOCERecTypes = (kLastOCERecTypeNum - kFirstOCERecTypeNum + 1);
```

```
{ Indices for the standard definitions for standard attribute types
(OCEAttributeTypeIndex): }


   kMemberAttrTypeNum                    = 1001;{ "Member" }
   kAdminsAttrTypeNum                    = 1002;{ "Administrators" }
   kMailSlotsAttrTypeNum                 = 1003;{ "mailslots" }
   kPrefMailAttrTypeNum                  = 1004;{ "pref mailslot" }
   kAddressAttrTypeNum                   = 1005;{ "Address" }
   kPictureAttrTypeNum                   = 1006;{ "Picture" }
   kAuthKeyAttrTypeNum                   = 1007;{ "auth key" }
   kTelephoneAttrTypeNum                 = 1008;{ "Telephone" }
   kNBPNameAttrTypeNum                   = 1009;{ "NBP Name" }
   kQMappingAttrTypeNum                  = 1010;{ "ForwarderQMap" }
   kDialupSlotAttrTypeNum                = 1011;{ "DialupSlotInfo" }
   kHomeNetAttrTypeNum                   = 1012;{ "Home Internet" }
   kCoResAttrTypeNum                     = 1013;{ "Co-resident M&M" }
   kFwdrLocalAttrTypeNum                 = 1014;{ "FwdrLocalRecord" }
   kConnectAttrTypeNum                   = 1015;{ "Connected To" }
   kForeignAttrTypeNum                   = 1016;{ "Foreign RLIs" }
   kOwnersAttrTypeNum                    = 1017;{ "Owners" }
   kReadListAttrTypeNum                  = 1018;{ "ReadList" }
   kWriteListAttrTypeNum                 = 1019;{ "WriteList" }
   kDescriptorAttrTypeNum                = 1020;{ "Descriptor" }
   kCertificateAttrTypeNum               = 1021;{ "Certificate" }
   kMsgQsAttrTypeNum                     = 1022;{ "MessageQs" }
   kPrefMsgQAttrTypeNum                  = 1023;{ "PrefMessageQ" }
   kMasterPFAttrTypeNum                  = 1024;{ "MasterPF" }
   kMasterNetSpecAttrTypeNum             = 1025;{ "MasterNetSpec" }
   kServersOfAttrTypeNum                 = 1026;{ "Servers Of" }
   kParentCIDAttrTypeNum                 = 1027;{ "Parent CID" }
   kNetworkSpecAttrTypeNum               = 1028;{ "NetworkSpec" }
   kLocationAttrTypeNum                  = 1029;{ "Location" }
   kTimeSvrTypeAttrTypeNum               = 1030;{ "TimeServer Type" }
   kUpdateTimerAttrTypeNum               = 1031;{ "Update Timer" }
   kShadowsOfAttrTypeNum                 = 1032;{ "Shadows Of" }
   kShadowServerAttrTypeNum              = 1033;{ "Shadow Server" }
   kTBSetupAttrTypeNum                   = 1034;{ "TB Setup" }
   kMailSetupAttrTypeNum                 = 1035;{ "Mail Setup" }
   kSlotIDAttrTypeNum                    = 1036;{ "SlotID" }
   kGatewayFileIDAttrTypeNum             = 1037;{ "Gateway FileID" }
   kMailServiceAttrTypeNum               = 1038;{ "Mail Service" }
   kStdSlotInfoAttrTypeNum               = 1039;{ "Std Slot Info" }
   kAssoDirectoryAttrTypeNum             = 1040;{ "Asso. Catalog" }
```

```
kDirectoryAttrTypeNum                = 1041;{ "Catalog" }
kDirectoriesAttrTypeNum              = 1042;{ "Catalogs" }
kSFlagsAttrTypeNum                   = 1043;{ "SFlags" }
kLocalNameAttrTypeNum                = 1044;{ "Local Name" }
kLocalKeyAttrTypeNum                 = 1045;{ "Local Key" }
kDirUserRIDAttrTypeNum               = 1046;{ "Dir User RID" }
kDirUserKeyAttrTypeNum               = 1047;{ "Dir User Key" }
kDirNativeNameAttrTypeNum            = 1048;{ "Dir Native Name" }
kCommentAttrTypeNum                  = 1049;{ "Comment" }
kRealNameAttrTypeNum                 = 1050;{ "Real Name" }
kPrivateDataAttrTypeNum              = 1051;{ "Private Data" }
kDirTypeAttrTypeNum                  = 1052;{ "Catalog Type" }
kDSAMFileAliasAttrTypeNum            = 1053;{ "CSAM File Alias" }
kCanAddressToAttrTypeNum             = 1054;{ "Can Address To" }
kDiscriminatorAttrTypeNum            = 1055;{ "Discriminator" }
kAliasAttrTypeNum                    = 1056;{ "Alias" }
kParentMSAMAttrTypeNum               = 1057;{ "Parent MSAM" }
kParentDSAMAttrTypeNum               = 1058;{ "Parent CSAM" }
kSlotAttrTypeNum                     = 1059;{ "Slot" }
kAssoMailServiceAttrTypeNum          = 1060;{ "Asso. Mail Service" }
kFakeAttrTypeNum                     = 1061;{ "Fake" }
kInheritSysAdminAttrTypeNum          = 1062;{ "Inherit System
                                              Administrators" }
kPreferredPDAttrTypeNum              = 1063;{ "Preferred PD" }
kLastLoginAttrTypeNum                = 1064;{ "Last Login" }
kMailerAOMStateAttrTypeNum           = 1065;{ "Mailer AOM State" }
kMailerSendOptionsAttrTypeNum        = 1066;{ "Mailer Send Options" }
kJoinedAttrTypeNum                   = 1067;{ "Joined" }
kUnconfiguredAttrTypeNum             = 1068;{ "Unconfigured" }
kVersionAttrTypeNum                  = 1069;{ "Version" }
kLocationNamesAttrTypeNum            = 1070;{ "Location Names" }
kActiveAttrTypeNum                   = 1071;{ "Active" }
kDeleteRequestedAttrTypeNum          = 1072;{ "Delete Requested" }
kGatewayTypeAttrTypeNum              = 1073;{ "Gateway Type" }

kFirstOCEAttrTypeNum = kMemberAttrTypeNum;{ first standard OCE attr type }

kLastOCEAttrTypeNum  = kGatewayTypeAttrTypeNum;{ last standard OCE
                                                 attr type }

kNumOCEAttrTypes     = (kLastOCEAttrTypeNum - kFirstOCEAttrTypeNum + 1);
```

Data Types

```
TYPE

{ OCE String Types }

   {RStringHeader}
   RStringHeader = RECORD
      charSet: CharacterSet;
      dataLength: INTEGER;
   END;

   { RString }
   RString = RECORD
      charSet: CharacterSet;
      dataLength: INTEGER;
      body: PACKED ARRAY[1..kRStringMaxBytes] OF Byte;
   END;

   { ProtoRString }
   ProtoRString = RECORD
      charSet: CharacterSet;
      dataLength: INTEGER;
      { Followed by body }
   END;

   RStringPtr = ^RString;

   RStringHandle = ^RStringPtr;

   ProtoRStringPtr = ^ProtoRString;

   {RString64}
   RString64 = RECORD
      charSet: CharacterSet;
      dataLength: INTEGER;
      body: PACKED ARRAY[1..kRString64Size] OF Byte;
   END;

   {RString32}
   RString32 = RECORD
      charSet: CharacterSet;
      dataLength: INTEGER;
      body: PACKED ARRAY[1..kRString32Size] OF Byte;
   END;
```

```
   Rstring32Ptr = ^Rstring32;

struct DirectoryName                    /* DirectoryName */
{
   RStringHeader
   Byte     body[kDirectoryNameMaxBytes];
};

   {NetworkSpec}
   NetworkSpec = RECORD
      charSet: CharacterSet;
      dataLength: INTEGER;
      body: PACKED ARRAY[1..kNetworkSpecMaxBytes] OF Byte;
   END;

   NetworkSpecPtr = ^NetworkSpec;

   RStringKind = INTEGER;

{ RecordID Types }

   {CreationID}
   CreationID = RECORD
      source: LONGINT;
      seq: LONGINT;
   END;

   AttributeCreationID = CreationID;

   CreationIDPtr = ^CreationID;

      {PackedPathName}
   PackedPathName = RECORD
      dataLength: INTEGER;
      data: PACKED ARRAY[1..kPathNameMaxBytes - sizeof(INTEGER)] OF Byte;
   END;

   {ProtoPackedPathName}
   ProtoPackedPathName = RECORD
      dataLength: INTEGER;
      { Followed by data }
   END;

   PackedPathNamePtr = ^PackedPathName;

   ProtoPackedPathNamePtr = ^ProtoPackedPathName;
```

```
{DirDiscriminator}
DirDiscriminator = RECORD
   signature: OCEDirectoryKind;
   misc: LONGINT;
END;

{ Catalog node number }
DNodeNum = LONGINT;

{ RLI }
RLI = RECORD
   directoryName: DirectoryNamePtr;
   discriminator: DirDiscriminator;
   dNodeNumber: DNodeNum;
   path: PackedPathNamePtr;
END;

RLIPtr = ^RLI;

{ PackedRLIHeader }
PackedRLIHeader = RECORD
   dataLength: INTEGER;
END;

{ PackedRLI }
PackedRLI = RECORD
   dataLength: INTEGER;
   data: PACKED ARRAY[1..kRLIMaxBytes] OF Byte;
END;

{ ProtoPackedRLI }
ProtoPackedRLI = RECORD
   dataLength: INTEGER;
   { Followed by data }
END;

PackedRLIPtr = ^PackedRLI;

ProtoPackedRLIPtr = ^ProtoPackedRLI;

{ LocalRecordID }
LocalRecordID = RECORD
   cid: CreationID;
```

```
      recordName: RStringPtr;
      recordType: RStringPtr;
   END;

   LocalRecordIDPtr = ^LocalRecordID;

   { ShortRecordID }
   ShortRecordID = RECORD
      rli: PackedRLIPtr;
      cid: CreationID;
   END;

   ShortRecordIDPtr = ^ShortRecordID;

   { RecordID }
   RecordID = RECORD
      rli: PackedRLIPtr;
      local: LocalRecordID;
   END;

   RecordIDPtr = ^RecordID;

   { PackedRecordIDHeader }
   PackedRecordIDHeader = RECORD
      dataLength: INTEGER;
   END;

   { PackedRecordID }
   PackedRecordID = RECORD
      dataLength: INTEGER;
      data: PACKED ARRAY[1..kPackedRecordIDMaxBytes] OF Byte;
   END;

   { ProtoPackedRecordID }
   ProtoPackedRecordID = RECORD
      dataLength: INTEGER;
      { Followed by data }
   END;

   PackedRecordIDPtr = ^PackedRecordID;

   ProtoPackedRecordIDPtr = ^ProtoPackedRecordID;

{ DSSpec Structures }
```

```
   { DSSpec }
   DSSpec = RECORD
      entitySpecifier: ^RecordID;
      extensionType: OSType;
      extensionSize: INTEGER;
      extensionValue: Ptr;
   END;

   DSSpecPtr = ^DSSpec;

   { PackedDSSpecHeader }
   PackedDSSpecHeader = RECORD
      dataLength: INTEGER;
   END;

   { PackedDSSpec }
   PackedDSSpec = RECORD
      dataLength: INTEGER;
      data: PACKED ARRAY[1..kPackedDSSpecMaxBytes] OF Byte;
   END;

   ProtoPackedDSSpec = RECORD
      dataLength: INTEGER;
      { Followed by data }
   END;

   PackedDSSpecPtr = ^PackedDSSpec;

   PackedDSSpecHandle = ^PackedDSSpecPtr;

   ProtoPackedDSSpecPtr = ^ProtoPackedDSSpec;

{ Attribute Structures }
   AttributeType = RECORD
      charSet: CharacterSet;
      dataLength: INTEGER;
      body: PACKED ARRAY[1..kAttributeTypeMaxBytes] OF Byte;
   END;

   AttributeTypePtr = ^AttributeType;

   { AttributeValue }
   AttributeValue = RECORD
      tag: AttributeTag;
```

```
      dataLength: LONGINT;
      bytes: Ptr;
   END;

   AttributeValuePtr = ^AttributeValue;

   AttributeTag = DescType;

   { Attribute }
   Attribute = RECORD
      attributeType: AttributeType;
      cid: AttributeCreationID;
      value: AttributeValue;
   END;

   AttributePtr = ^Attribute;

{ recordType index }
   OCERecordTypeIndex = INTEGER;

{ AttributeType index }
   OCEAttributeTypeIndex = INTEGER;

{ OCE Catalog Types }
   OCEDirectoryKind = LONGINT;

{ OCE Catalog Node Types }
   DirNodeKind = LONGINT;

{ MyDSSpecStreamer callback routine }
   MyDSSpecStreamer = ProcPtr;
```

AOCE Utility Functions

AOCE String Functions

```
FUNCTION OCECopyRString     (str1: RStringPtr; str2: RStringPtr;
                             str2Length: INTEGER): OSErr;
PROCEDURE OCECToRString     (cStr: Ptr; charSet: CharacterSet; rStr:
                             RStringPtr; rStrLength: INTEGER);
PROCEDURE OCEPToRString     (pStr: Str255; charSet: CharacterSet; rStr:
                             RStringPtr; rStrLength: INTEGER);
FUNCTION OCERToPString      (rStr: RStringPtr): StringPtr; INLINE $303C,
                             kOCERToPString, $AA5C;
```

```
FUNCTION OCERelRString      (str1: UNIV Ptr; str2: UNIV Ptr; kind:
                             RStringKind): INTEGER;
FUNCTION OCEEqualRString    (str1: UNIV Ptr; str2: UNIV Ptr; kind:
                             RStringKind): BOOLEAN;
FUNCTION OCEValidRString    (str: UNIV Ptr; kind: RStringKind): BOOLEAN;
```

Creation Identifier Functions

```
FUNCTION OCEEqualCreationID (cid1: CreationID; cid2: CreationID): BOOLEAN;
PROCEDURE OCECopyCreationID (cid1: CreationID; VAR cid2: CreationID);
FUNCTION OCENullCID: CreationIDPtr;
FUNCTION OCEPathFinderCID: CreationIDPtr;
PROCEDURE OCESetCreationIDtoNull
                            (VAR cid: CreationID);
```

Packed pathname Functions

```
FUNCTION OCECopyPackedPathName
                            (path1: PackedPathNamePtr; path2:
                             PackedPathNamePtr; path2Length: INTEGER):
                             OSErr;
FUNCTION OCEIsNullPackedPathName
                            (path: PackedPathNamePtr): BOOLEAN;
FUNCTION OCEPackedPathNameSize
                            (VAR parts: RStringPtr; nParts: INTEGER):
                             INTEGER;
FUNCTION OCEDNodeNameCount  (path: PackedPathNamePtr): INTEGER;
FUNCTION OCEUnpackPathName  (path: PackedPathNamePtr; VAR parts:
                             RStringPtr; nParts: INTEGER): INTEGER;
FUNCTION OCEPackPathName    (VAR parts: RStringPtr; nParts: INTEGER; path:
                             PackedPathNamePtr; pathLength: INTEGER): OSErr;
FUNCTION OCEEqualPackedPathName
                            (path1: PackedPathNamePtr; path2:
                             PackedPathNamePtr): BOOLEAN;
FUNCTION OCEValidPackedPathName
                            (path: PackedPathNamePtr): BOOLEAN;
```

Catalog Discriminator Functions

```
PROCEDURE OCECopyDirDiscriminator
                            (disc1: DirDiscriminator; VAR disc2:
                             DirDiscriminator);
```

```
FUNCTION OCEEqualDirDiscriminator
                          (disc1: DirDiscriminator; disc2:
                           DirDiscriminator): BOOLEAN;
```

Record Location Information Functions

```
PROCEDURE OCENewRLI       (VAR newRLI: RLI; dirName: DirectoryName; VAR
                           discriminator: DirDiscriminator dNodeNumber:
                           DNodeNum; path: PackedPathName);
PROCEDURE OCEDuplicateRLI (rli1: RLI; VAR rli2: RLI);
FUNCTION OCECopyRLI       (rli1: RLI; VAR rli2: RLI): OSErr;
FUNCTION OCEEqualRLI      (rli1: RLI; rli2: RLI): BOOLEAN;
FUNCTION OCEValidRLI      (theRLI: RLI): BOOLEAN;
FUNCTION OCECopyPackedRLI (prli1: PackedRLIPtr; prli2: PackedRLIPtr;
                           prli2Length: INTEGER): OSErr;
FUNCTION OCEPackedRLISize (theRLI: RLI): INTEGER;
FUNCTION OCEPackRLI       (theRLI: RLI; prli: PackedRLIPtr; prliLength:
                           INTEGER): OSErr;
PROCEDURE OCEUnpackRLI    (prli: PackedRLIPtr; VAR theRLI: RLI);
FUNCTION OCEPackedRLIPartsSize
                          (dirName: DirectoryNamePtr; VAR parts:
                           RStringPtr; nParts: INTEGER): INTEGER;
FUNCTION OCEPackRLIParts  (dirName: DirectoryNamePtr; discriminator:
                           DirDiscriminator; dNodeNumber: DNodeNum; VAR
                           parts: RStringPtr; nParts: INTEGER; prli:
                           PackedRLIPtr; prliLength: INTEGER): OSErr;
FUNCTION OCEEqualPackedRLI (prli1: PackedRLIPtr; prli2: PackedRLIPtr):
                           BOOLEAN;
FUNCTION OCEValidPackedRLI (prli: PackedRLIPtr): BOOLEAN;
FUNCTION OCEExtractAlias  (prli: PackedRLIPtr): AliasPtr;
FUNCTION OCEGetDirectoryRootPackedRLI
                          ():PackedRLIPtr;
```

Local Record Identifier Functions

```
PROCEDURE OCENewLocalRecordID
                          (recordName: RStringPtr; recordType:RStringPtr;
                           cid: CreationID; VAR lRID: LocalRecordID);
FUNCTION OCECopyLocalRecordID
                          (lRID1: LocalRecordID; VAR lRID2:
                           LocalRecordID): OSErr;
FUNCTION OCEEqualLocalRecordID
                          (lRID1: LocalRecordID; lRID2: LocalRecordID):
                           BOOLEAN;
```

Short Record Identifier Functions

```
PROCEDURE OCENewShortRecordID
                            (theRLI: PackedRLIPtr; cid: CreationID; sRID:
                             ShortRecordIDPtr);
FUNCTION OCECopyShortRecordID
                            (sRID1: ShortRecordID; VAR sRID2:
                             ShortRecordID): OSErr;
FUNCTION OCEEqualShortRecordID
                            (sRID1: ShortRecordID; sRID2: ShortRecordID):
                             BOOLEAN;
```

Record Identifier Functions

```
FUNCTION OCEGetIndRecordType
                            (STRINGIndex: OCERecordTypeIndex): RStringPtr;
PROCEDURE OCENewRecordID    (theRLI: PackedRLIPtr; lRID: LocalRecordID; VAR
                             rid: RecordID);
FUNCTION OCECopyRecordID    (rid1: RecordID; rid2: RecordID): OSErr;
FUNCTION OCEEqualRecordID   (rid1: RecordID; rid2: RecordID): BOOLEAN;
```

Packed Record Identifier Functions

```
FUNCTION OCECopyPackedRecordID
                            (pRID1: PackedRecordIDPtr; pRID2:
                             PackedRecordIDPtr; pRID2Length: INTEGER):
                             OSErr;
FUNCTION OCEPackedRecordIDSize
                            (rid: RecordID): INTEGER;
FUNCTION OCEPackRecordID    (rid: RecordID; VAR pRID: PackedRecordIDPtr;
                             packedRecordIDLength: INTEGER): OSErr;
PROCEDURE OCEUnpackRecordID (pRID: PackedRecordIDPtr; VAR rid: RecordID);
FUNCTION OCEEqualPackedRecordID
                            (pRID1: PackedRecordIDPtr; pRID2:
                             PackedRecordIDPtr): BOOLEAN;
FUNCTION OCEValidPackedRecordID
                            (pRID: PackedRecordIDPtr): BOOLEAN;
```

Attribute Type Functions

```
FUNCTION OCEGetIndAttributeType
                            (STRINGIndex: OCEAttributeTypeIndex):
                             AttributeTypePtr;
```

Catalog Services Specification Functions

```
FUNCTION OCECopyPackedDSSpec
                            (pdss1: PackedDSSpecPtr; pdss2:
                             PackedDSSpecPtr; pdss2Length: INTEGER): OSErr;
FUNCTION OCEPackedDSSpecSize
                            (dss: DSSpec): INTEGER;
FUNCTION OCEPackDSSpec      (dss: DSSpec; VAR pdss: PackedDSSpecPtr;
                             pdssLength: INTEGER): OSErr;
PROCEDURE OCEUnpackDSSpec   (pdss: PackedDSSpecPtr; VAR dss: DSSpec; VAR
                             rid: RecordID);
FUNCTION OCEEqualDSSpec     (pdss1: DSSpec; pdss2: DSSpec): BOOLEAN;
FUNCTION OCEEqualPackedDSSpec
                            (pdss1: PackedDSSpecPtr; pdss2:
                             PackedDSSpecPtr): BOOLEAN;
FUNCTION OCEValidPackedDSSpec
                            (pdss: PackedDSSpecPtr): BOOLEAN;
FUNCTION OCEGetDSSpecInfo   (spec: DSSpec): OSType;
FUNCTION OCEGetExtensionType
                            (pdss: PackedDSSpecPtr): OSType;
FUNCTION OCEStreamPackedDSSpec
                            (dss: DSSpec; stream: MyDSSpecStreamer;
                             userData: LONGINT; VAR actualCount: LONGINT):
                             OSErr;
```

Application-Defined Functions

```
FUNCTION MyDSSpecStreamer   (VAR buffer: void; count: LONGINT; eof:
                             BOOLEAN; userData: LONGINT): OSErr;}
```

Assembly Language Summary

Trap Macros Requiring Routine Selectors

```
__OCEUtils
```

Selector	Routine
$0300	kOCECopyCreationID
$0301	kOCECopyDirDiscriminator
$0302	kOCECopyLocalRecordID
$0303	kOCECopyPackedDSSpec
$0304	kOCECopyPackedPathName
$0305	kOCECopyPackedRLI
$0306	kOCECopyPackedRecordID
$0307	kOCECopyRLI
$0308	kOCECopyRString
$0309	kOCECopyRecordID
$030A	kOCECopyShortRecordID
$030B	kOCEDuplicateRLI
$030C	kOCEEqualCreationID
$030D	kOCEEqualDirDiscriminator
$030E	kOCEEqualDSSpec
$030F	kOCEEqualLocalRecordID
$0310	kOCEEqualPackedDSSpec
$0311	kOCEEqualPackedPathName
$0312	kOCEEqualPackedRecordID
$0313	kOCEEqualPackedRLI
$0314	kOCEEqualRecordID
$0315	kOCEEqualRLI
$0316	kOCEEqualRString
$0317	kOCEEqualShortRecordID
$0318	kOCEExtractAlias
$0319	kOCEGetDSSpecInfo
$031A	kOCEGetIndAttributeType
$031B	kOCEGetIndRecordType
$031C	kOCEGetXtnType
$031D	kOCEIsNullPackedPathName
$031E	kOCENewLocalRecordID
$031F	kOCENewRLI

Selector	Routine
$0320	kOCENewRecordID
$0321	kOCENewShortRecordID
$0322	kOCEPackDSSpec
$0323	kOCEPackPathName
$0324	kOCEPackRLI
$0325	kOCEPackRLIParts
$0326	kOCEPackRecordID
$0327	kOCEPackedDSSpecSize
$0328	kOCEPackedPathNameSize
$0329	kOCEPackedRLIPartsSize
$032A	kOCEPackedRLISize
$032B	kOCEPackedRecordIDSize
$032C	kOCEDNodeNameCount
$032D	kOCERelRString
$032E	kOCESetCreationIDtoNull
$032F	kOCEUnpackDSSpec
$0330	kOCEUnpackPathName
$0331	kOCEUnpackRLI
$0332	kOCEUnpackRecordID
$0333	kOCEValidPackedDSSpec
$0334	kOCEValidPackedPathName
$0335	kOCEValidPackedRecordID
$0336	kOCEValidPackedRLI
$0337	kOCEValidRLI
$0338	kOCEValidRString
$0339	kOCECToRString
$033A	kOCEPToRString
$033B	kOCERToPString
$033C	kOCEPathFinderCID
$033D	kOCEStreamPackedDSSpec
$0344	kOCENullCID
$0345	kOCEGetAccessControlDSSpec
$0346	kOCEGetRootPackedRLI

Result Codes

There is no allocated range of result codes for the Utility Manager. Functions may, however, return standard Macintosh result codes such as `noErr` 0 (No error) and `memFullErr` –108 (Buffer not large enough).

CHAPTER 3

Standard Mail Package

Contents

This chapter describes the AOCE Standard Mail Package. The AOCE Standard Mail Package provides a high-level interface that makes it easy for you to add electronic-mail capabilities to your applications.

The Standard Mail Package provides two separate services:

- an easy way to send a letter or a file from within your application without user intervention
- a complete user interface that you can use to convert any of your application's documents into electronic mail

In addition, you can use the Standard Catalog Package to provide a user interface for browsing AOCE catalogs and selecting records from within your application.

If you want to design and implement your own electronic messaging service using the AOCE toolbox, see the chapter "Interprogram Messaging Manager" in this book.

About the Standard Mail Package

The AOCE Standard Mail Package provides a high-level interface to the AOCE Interprogram Messaging (IPM) Manager. It works together with the Catalog Browser and the Digital Signature Manager to present a consistent and easy-to-use user interface for addressing letters, signing letters, and sending your application's documents as electronic mail.

The Standard Mail Package can be divided into two main parts: the send-letter functions and the mailer functions. The Standard Mail Package relies on other components of the Apple Open Collaboration Environment, but you do not have to call the underlying AOCE services directly to add electronic-mail capabilities to your application.

The Send-Letter Functions

The Standard Mail Package provides a basic, very easily implemented method of sending documents and other files that can be used either by users of applications or by applications acting without user intervention (agents). You can use the Standard Mail Package functions (described in "Send-Letter Functions" on page 3-37) to enclose a file with an AppleMail letter and then send the letter, to send a document as an image file, or to send a file so that it appears in the recipient's In Tray not as a letter but as the original file.

The send-letter functions provide no interface for opening a letter from within your application. When the user double-clicks a document in the In Tray, the Finder attempts to launch the application that was used to send the letter, and that application opens the document. If that application is not present, the Finder displays a dialog box asking the user whether it should open the letter with the AppleMail application provided with the AOCE software.

The Mailer Functions

The Standard Mail Package also provides a more sophisticated electronic-mail interface. This interface adds a new region—known as a **mailer**—to any window.

"Providing Mailers in Your Windows" on page 3-45 describes functions to create a new mailer, reposition a mailer in your window, control the way the user cycles through fields in the mailer and your document using the Tab key, and dispose of a mailer.

You can use the functions described in "Handling Events in Mailers" beginning on page 3-63 to handle events and Apple events that pertain to mailers and to make sure that your menu commands accurately reflect the state of the mailer while a user is working in it.

You use the functions described in "Sending and Saving Mail" beginning on page 3-72 to send, save, or read a document containing a mailer. Use the routines in "Printing Mailers" beginning on page 3-107 when you want to print a document containing a mailer.

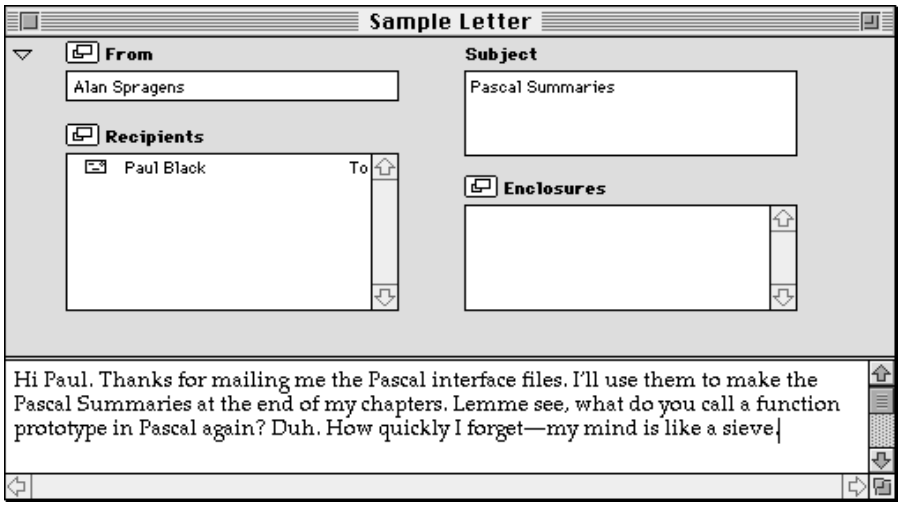
You can use the functions described in "Getting and Setting Information in the Mailer" beginning on page 3-110 to read and set values in mailer fields and send options from within your program instead of through the mailer or the standard dialog boxes provided by the mailer.

You use the `SMPOpenLetter` function, described in "Reading Mail" beginning on page 3-93, to open a letter to read its contents.

Mailers

The mailer lets the sender enter addresses and subject information, enclose other files and folders in the letter, and add a digital signature to the letter. It lets the recipient read all of this information and verify the digital signature. Figure 3-1 shows a mailer in an application window. Each time the user forwards a letter, another mailer holding addresses for the forwarder and the new recipients is added to the letter. The mailers for a forwarded letter are collectively referred to as a **mailer set.**

Figure 3-1 Mailer in an application window



The preferred user interface for an AOCE Standard Mail Package letter is to place the mailer inside your document's windows, just below the title bar. However, if your application's windows are not suitable for displaying mailers, you can place the mailer in its own, separate window. The user can display the mailer in either of two states: *contracted* or *expanded*. Figure 3-2 shows a sample mailer in the contracted state and Figure 3-3 shows the same mailer in the expanded state.

Figure 3-2 Mailer in the contracted state

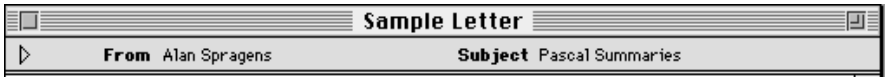
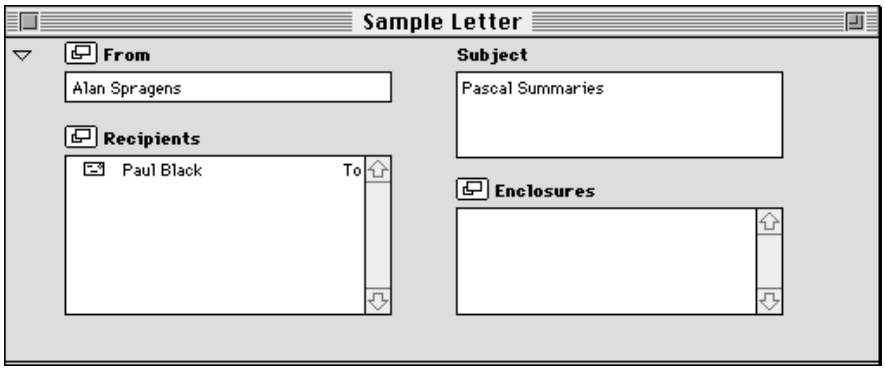


Figure 3-3 Mailer in the expanded state



The user can drag an address from the Finder or another mailer into the Recipients field or can open an addressing panel, as shown in Figure 3-4. The user can select among four versions of the addressing panel by clicking one of the icons at the left side of the panel. These versions of the addressing panel allow the user to select an address from the default personal catalog or from any AOCE catalog, to find a record by typing in all or part of the name of the record, or to type in the entire address. These four versions of the panel are shown in Figure 3-5.

Figure 3-4 Mailer with addressing panel open

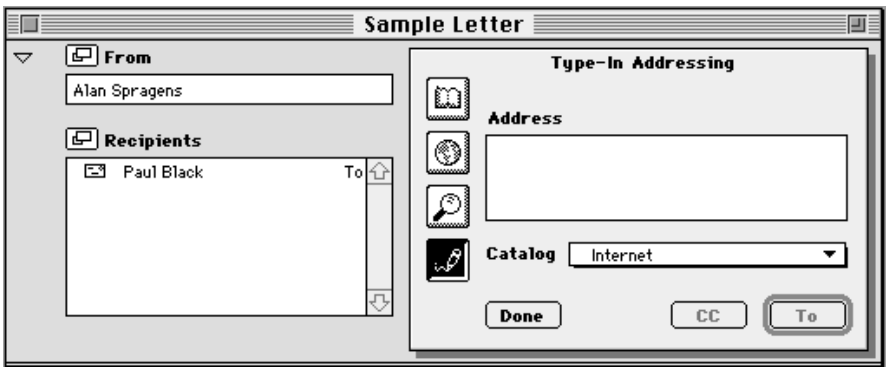
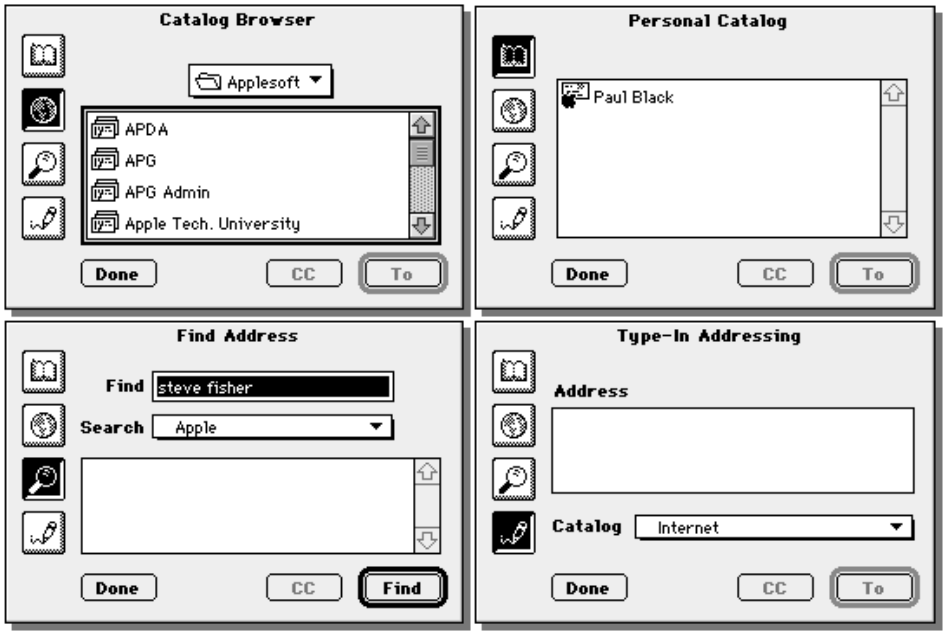


Figure 3-5 The four versions of the addressing panel



Letter Formats

When you use a mailer to send a document as a letter, you can send the document in a "native" format (that is, any one of the document formats supported by your application), you can send an image of your document, you can send the content of your document in a special format called **standard interchange format,** or you can send the document in any two or all three of these formats simultaneously.

You may choose to employ the AOCE Standard Mail Package at either of two levels: full mailer support or basic mailer support. An application that offers full mailer support can read and write both standard interchange format and images. An application that offers basic mailer support can send either images or standard interchange format or both. Either type of application might also send documents in one of the application's native formats.

When you send your document, the Standard Mail Package delivers it to the addressees' In Trays. When a recipient double-clicks a document in the In Tray, the application used to send the document (if present) opens it, and the mailer appears at the top of the window. If the file includes an image of the document or a standard interchange format version of its content, any application that offers full mailer support can open it.

Each user who has AOCE software has the AppleMail application, which provides full mailer support. Thus, every user who has AOCE software can read, either as an image or in standard interchange format, every document sent by an application that provides either full or basic mailer support. In addition, if your application can send and read documents sent in its own native formats, users who have your application have access to the complete document when they receive it.

A letter consists of a header that contains addressing and priority information, followed by blocks of data, followed by enclosures. Certain types of data blocks have standard definitions, such as the image block and standard interchange format blocks. The image block contains an image of the document being sent; you must provide an image-drawing routine (page 3-123) to draw each page. The `SMPImage` function (page 3-88) creates the image block and adds it to the letter. Standard interchange format blocks contain a version of your document that can include text, styled text, sounds, pictures, and QuickTime movies. Standard interchange format can be converted by access modules and read by any standard letter application (such as the AppleMail application provided with the AOCE software). You can define other blocks in any way you wish. You use the `SMPAddBlock` function (page 3-91) to add blocks to a letter.

You can send your own document in one of its native formats as an enclosure to the letter, known as a **main enclosure** (also referred to as a *content enclosure*), or incorporate it into data blocks, as you wish. The Standard Mail Package user interface also allows the user to enclose other files. (The main enclosure is not visible to the user as an enclosure.)

Note
If you are using the IPM Manager or the MSAM API to send letters to the Standard Mail Package, you should avoid sending any nested letters that contain standard content. If the Standard Mail Package receives a letter that contains a nested letter, it ignores any content (standard interchange format or image format) within the nested letter. It displays the header and nesting information of the nested letter as a forwarded mailer. u

The Standard Catalog Package

The Standard Catalog Package provides authentication and letter-addressing services that complement the routines described in this chapter. See the chapter "Standard Catalog Package" in this book for more information.

Using the Standard Mail Package

This section describes how to initialize the Standard Mail Package and use it to create a mailer, send mail, receive mail, forward and reply to mail, close a letter, and handle events in the mailer.

Initializing the Standard Mail Package

Before you can enable Standard Mail Package features in your application, you must use the Gestalt Manager to ensure that the system on which your application is running supports the Standard Mail Package.

To determine the version of the Standard Mail Package mailer functions, call the `Gestalt` function with the selector `gestaltSMPMailerVersion`. The function returns the version number of the mailers in the low-order word of the `response` parameter. For example, a value of 0x0101 indicates version 1.0.1. If the Standard Mail Package is not present and available, the `Gestalt` function returns 0 for the version number. Similarly, to determine the version of the send-letter functions, use the selector `gestaltSMPSendLetterVersion`.

Listing 3-1 shows a function that returns `true` only if the Standard Mail Package is installed and available.

Listing 3-1 Testing for the presence of Standard Mail Package services

```
Boolean MyTestForStandardMail(void)
{
   OSErr    err;
   long     response;
```

```
   err = Gestalt(gestaltSMPMailerVersion, &response);
   if ((err!=noErr) || (response==0))
      return false;

   return true;
}
```

If the Standard Mail Package is not available, you should disable those features of your application while allowing the user to use its other features normally.

After determining that the Standard Mail Package is available, you must initialize it using the `SMPInitMailer` function, passing in the version number of the package current for the services incorporated into your application. If, at run time, the current version of the Standard Mail Package is later than the one with which you compiled your application, the package initializes in compatibility mode, supporting the older version's functions. If, conversely, the run-time version is earlier, `SMPInitMailer` returns an error. The code in Listing 3-2 calls the initialization function.

Listing 3-2 Initializing the Standard Mail Package

```
OSErr MyInitStandardMail(void)
{
   OSErr err;

   SetCursor(&gWatchCursor);
   err = SMPInitMailer(kSMPVersion);
   SetCursor(&qd.arrow);
   return err;
}
```

Creating a Mailer

The Standard Mail Package enables any application to add support for mailing documents directly to other users on the network without going through intermediate mail applications. It provides standard user interface elements needed to address, send, and receive documents through the mailer, which appears as a special pane in the window of the document to be sent. This section describes how to add a mailer to a window.

Listing 3-3 uses the `SMPGetDimensions` (page 3-48) function to find the dimensions of the standard mailer window, and it creates a document window just large enough to accommodate the mailer. More typically, you would size your application windows according to the requirements of your application and use `SMPGetDimensions` to place the mailer and perform actions such as adjusting the content area of your window. The function then creates the mailer by calling the `SMPNewMailer` function (page 3-46), and it makes the mailer the initial target of user actions with the `SMPBecomeTarget`

function (page 3-54), specifying the target field within the mailer as `kSMPOther` (that is, a field other than the Recipients, From, or other enumerated field as described on page 3-32). The application-defined `MyErrorAlert` function in the listings throughout this section reports errors to the user in a standard manner.

Listing 3-3 Creating a mailer

```
void
MyBuildMailerWindow(void)
{
   Rect        boundsRect;
   Point       mailerCorner;
   short       mailerWidth;
   short       mailerContractedHeight;
   short       mailerExpandedHeight;

   boundsRect = qd.screenBits.bounds;
   boundsRect.top += ((GetMBarHeight() + 1) * 2);
   InsetRect(&boundsRect, 4, 4);
   gStatus = SMPGetDimensions(
      &mailerWidth,
      &mailerContractedHeight,
      &mailerExpandedHeight
      );
   if (gStatus != noErr)
      MyErrorAlert(gStatus, "\pSMPGetDimensions");
   else {
      boundsRect.right = boundsRect.left + mailerWidth;
      boundsRect.bottom = boundsRect.top + mailerExpandedHeight;
   }
   gMailerWindow = NewWindow(
      NULL,             /* no window storage */
      &boundsRect,      /* window shape */
      "\pMiniMailer",   /* window title */
      TRUE,             /* visible */
      documentProc,     /* document, no zoom box */
      (WindowPtr) -1L,  /* in front */
      TRUE,             /* has close box */
      0                 /* refCon (ignored) */
   );
   if (gMailerWindow == NULL) {
      MyErrorAlert(MemError(), "\pNewWindow (fatal)");
      ExitToShell();
```

```
   }
   SetPort(gMailerWindow);    /* set port to be safe */
   SetPt(&mailerCorner, 0, 0);/* locate mailer in window */
   gStatus = SMPNewMailer(    /* create Standard Mailer */
      gMailerWindow,          /* in this window */
      mailerCorner,           /* mailer top-left */
      FALSE,                  /* cannot contract */
      TRUE,                   /* initially expanded */
      0,                      /* default identity */
      nil,                    /* no prepare-to-draw callback */
      0                       /* no client data */
      );
   if (gStatus != noErr) {
      MyErrorAlert(gStatus, "\pSMPNewMailer (fatal)");
      ExitToShell();
   }
}
```

The `SMPNewMailer` function call shown in Listing 3-3 passes a value of 0 for the identity of the caller, which invokes the most recently authenticated user identity (see “Authenticating a User” on page 3-36). Note that setting the `Boolean` parameter `canContract` of the `SMPNewMailer` function to `FALSE` is unusual; Listing 3-3 does it because the window exists only to accommodate the mailer. To add a mailer to an existing document window, call the `SMPNewMailer` function, passing in the window pointer, followed by the `SMPGetDimensions` function, to adjust the size of the window content area.

Sending Mail

The first step in sending a letter is to display the send-options dialog box. This dialog box is similar to the standard print dialog box, offering the user options as to how the letter should be sent. Listing 3-4 illustrates a way to display the send-options dialog box. The code assumes that your application stores as a resource a list of formats in which it can send letters; these formats should be those in which your application can save documents. It also assumes that your application stores user preference values, including send options, in a global struct named `gPreferences`.

Listing 3-4 Displaying the send-options dialog box

```
GetResString(nativeFormat, kAppNameID, kAppName);
GetWTitle(window, docTitle);
nativeFormatArray[0] = (StringPtr)nativeFormat;
SetCursor(&qd.arrow);
err = SMPSendOptionsDialog(window, docTitle, nativeFormatArray, 1,
```

```
   kSMPNativeMask | kSMPImageMask | kSMPStandardInterchangeMask,
   &gPreferences.sendFormat, nil, 0L, &gPreferences.sendFormat,
   &gPreferences.sendOptions);
if (err == userCanceledErr)
   return;
if (err !== noErr) {
   MyErrorAlert(err, "\pSMPSendOptionsDialog");
   return;
}
```

The `SMPSendOptionsDialog` function (page 3-73) prompts the user for send options. It returns the name of the format that should be used to send the letter, which is used in the next part of the process. The process of sending a letter is begun by calling the `SMPBeginSend` function (page 3-81), passing in the user's send options (see Listing 3-5). The Standard Mail Package uses this information to build the header for the letter. Any subsequent content-adding function calls apply to the letter specified in the `SMPBeginSend` call. Listing 3-5 shows how to perform the send operation.

Listing 3-5 Performing the send operation

```
SetCursor(&gWatchCursor);
/* Use creator if you have native format, else use AppleMail. */
if ((gPreferences.sendFormat.whichFormats & kSMPNativeMask != 0) {
   letterCreator = kMyAppCreator;
   letterType = kMyLtrMsgType;
}
else {
   letterCreator = 'lap2';
   letterType = kMailLtrMsgType;
}
err = SMPBeginSend(window, letterCreator, letterType,
         &gPreferences.sendOptions, &mustAddContent);
if (err != noErr) {
   SetCursor(&qd.arrow);
   MyErrorAlert(err, "\pSMPBeginSend");
   return;
}
if (mustAddContent) {
   err = MyAddLetterBlocks(window, infoPtr,
                        &gPreferences.sendFormat);
   if (err != noErr)
      MyErrorAlert(err);
}
```

```
err = SMPEndSend(window, (err == noErr));
if (err != noErr)
   MyErrorAlert(err, "\pSMPEndSend");
```

The application-defined `MyAddLetterBlocks` function adds the actual blocks of content to the letter. It adds blocks only if the `mustAddContent` Boolean value, returned from `SMPBeginSend`, is set to `true`; there is no need to add content blocks to a letter forwarded unchanged. The function adds blocks in any combination of the three types of content formats: native application format, standard interchange (AppleMail) format, and image format. The `MyAddLetterBlocks` function calls appropriate subroutines to add the blocks.

Finally, you must call the `SMPEndSend` function (page 3-84) to send the letter. Its second parameter is a Boolean value that specifies whether to execute the send operation or to cancel the send process begun with `SMPBeginSend`. The example in Listing 3-5 uses this parameter to ensure that if `MyAddLetterBlocks` or any of its subroutines returns a nonzero error code, the send operation is canceled.

The `MyAddLetterBlocks` function and its subroutine functions are illustrated in Listing 3-6. The `MyAddLetterBlocks` function checks the `sendFormat` parameter returned from the `SMPSendOptionsDialog` function to determine which formats to add, and it calls one, two, or all three of the functions that actually add the content blocks.

Listing 3-6 Adding the letter content

```
OSErr MyAddLetterBlocks(WindowPtr window, WInfoPtr infoPtr,
         SMPSendFormat *sendFormat, StringPtr nativeFormatName)
{
   OSErr err = noErr;
   /* Add image (snapshot). */
   if (!sendFormat ||
         (sendFormat->whichFormats & kSMPImageMask)) {
      err = MyAddLetterImage(window, infoPtr);
      if (err != noErr)
         return err;
   }

   /* Add standard letter interchange format (AppleMail). */

   if (!sendFormat ||
      (sendFormat->whichFormats & kSMPStandardInterchangeMask)) {
      err = MyAddAppleMailContent(window, infoPtr);
      if (err != noErr)
         return err;
   }
```

```
   /* Add main content enclosure (native). */

   if (!sendFormat ||
         (sendFormat->whichFormats & kSMPNativeMask)) {
      err = MyAddNativeContent(window,infoPtr,nativeFormatName);
      if (err != noErr)
         return err;
   }
   return err;
}
```

Native application content is stored in files accessed by file system `FSSpec` data structures. Thus, to add native content you must save the content to a temporary file before adding it to the letter. You can use an application-defined utility routine (the `MySaveFileToTemp` function, not shown here) for this purpose. Once the temporary file is available, the `MyAddNativeContent` function (Listing 3-7) calls `SMPAddMainEnclosure` (page 3-90), passing in the letter window pointer and the file specification. Finally, the `MyAddNativeContent` function calls the `SMPAddBlock` function (page 3-91) to add a block indicating the name of the native format used in the letter.

Listing 3-7 Adding the application's native-format content

```
OSErr MyAddNativeContent(WindowPtr window, WInfoPtr infoPtr,
                         StringPtr nativeFormatName)
{
   OSErr err;
   FSSpec fSpec;
   OCECreatorType blockType;

   /* Save file temporarily so you can add by FSSpec. */

   err = MySaveFileToTemp(infoPtr, &fSpec);
   if (err != noErr)
      return err;
   err = SMPAddMainEnclosure(window, &fSpec);
   FSpDelete(&fSpec);

   /* Add native-format name string block. */

   if (err == noErr) {
      blockType.msgCreator = kMailAppleMailCreator;
      blockType.msgType = kSMPNativeFormatName;
```

```
      err = SMPAddBlock(window, &blockType, false,
                  &nativeFormatName[1], nativeFormatName[0],
                  kMailFromStart, 0);
   }
   return err;
}
```

In Listing 3-8, the `MyAddAppleMailContent` function creates and adds a content block segment in one of the AppleMail standard interchange formats. This example represents data in PICT format, indicated by the constant `kMailPictSegmentType`, passed as a parameter to the `SMPAddContent` function (page 3-85). Other standard interchange formats handle text, styled text, sound, and movies.

Listing 3-8 Adding AppleMail standard interchange-format content

```
OSErr MyAddAppleMailContent(WindowPtr window, WInfoPtr infoPtr)
{
   OSErr err;
   PicHandle thePicture;

   thePicture = MyDrawImageToPicture(window, infoPtr);
   if (thePicture) {
      HLock((Handle)thePicture);
      err = SMPAddContent(window, kMailPictSegmentType, false,
            *thePicture, GetHandleSize((Handle)thePicture),
            nil, true, smRoman);
      KillPicture(thePicture);
   }
   else return kInternalError;
   return err;
}
```

The code shown in Listing 3-9 creates an image from your document and adds it to the letter. The `SMPImage` function (page 3-88) requires you to pass in a pointer to a callback routine, an application-defined function (described on page 3-123) that actually draws the image of your document.

The `SMPImage` function adds the image blocks to the letter. You provide it with input parameters of the pointer to the letter window, a pointer to your image-drawing callback function (`MyDrawImageProc`, in Listing 3-9), a reference constant (used to pass a pointer to a block of information about the window in this example), and a Boolean value indicating whether your image-drawing function can draw in color (in Listing 3-9, it does not). The `MyDrawImageProc` function first sets up the resolution and size of the page using information in the print record for the window (in this example, a pointer to the print record is contained in the window information block passed in the reference

constant). Next, `MyDrawImageProc` calls the `SMPNewPage` function (page 3-41) to set up the graphics drawing port, as your image-drawing routine must do before drawing each page, then calls `MyDrawAllShapes` to image the page.

The application-defined `MyDrawAllShapes` function (called in Listing 3-9 but not shown) images the entire page with QuickDraw calls. The same function is called in the application-defined `MyDrawImageToPicture` function, which is used to add standard interchange format AppleMail content to a letter (see Listing 3-8). In that case the `MyDrawImageToPicture` function must provide a graphics port for QuickDraw to draw into.

Listing 3-9 Adding image-format content

```
OSErr MyAddLetterImage(WindowPtr window, WInfoPtr infoPtr)
{
   return SMPImage(window, MyDrawImageProc, (long)infoPtr, false);
}

pascal void MyDrawImageProc(long refCon, Boolean inColor)
{
   #pragma unused (inColor)
   OpenCPicParams newHeader;
   OSErr    err;
   Point    zeroPt = (0, 0);
   WInfoPtr infoPtr;
   TPrPtr   prInfo;

   infoPtr = (WInfoPtr)refCon;
   prInfo = (**(infoPtr->printRecord)).prInfo;

   newHeader.srcRect = prInfo.rPage;
   newHeader.hRes = FixRatio(prInfo.iHRes, 1);
   newHeader.vRes = FixRatio(prInfo.iVRes, 1);
   newHeader.version = -2;
   newHeader.reserved1 = 0;
   newHeader.reserved2 = 0L;
   err = SMPNewPage(&newHeader);
   if (err != noErr)
      MyErrorAlert(err, "\pSMPNewPage");
   MyDrawAllShapes(infoPtr, zeroPt);
}
```

Receiving Mail

A mail-aware application can receive mail in either of two ways: the user can double-click the letter in the mailbox In Tray or in the Finder. In either case, this action generates an Open Documents core Apple event (`'aevt' 'odoc'`) that the Finder sends to your application. If the letter is on disk, the Apple event includes a file specification of type `FSSpec`; if it is in the In Tray, the Apple event includes instead a letter specification of type `LetterSpec`. The portion of an Apple event handler shown in Listing 3-10 shows how to process both file and letter specifications. The Standard Mail Package handles both file and letter specifications through the letter descriptor structure, which includes both formats.

Listing 3-10 Apple event handler processing both file and letter specifications

```
AECountItems(&docList, &itemsInList);
for (index = 1; index <= itemsInList; index++) {
   err = AESizeOfNthItem(&docList, index, &returnedType, &size);
   if (err != noErr)
      return err;
   if (returnedType == typeLetterSpec) {
      diskForm = false;
      err = AEGetNthPtr(&docList, index, typeLetterSpec, &keywd,
            &returnedType, (Ptr)&myLetterSpec, sizeof(LetterSpec),
            &actualSize);
   } else if ((returnedType == typeAlias) ||
                              (returnedType == typeFSS)) {
      diskForm = true;
      err = AEGetNthPtr(&docList, index, typeFSS, &keywd,
            &returnedType, (Ptr)&myFSS, sizeof(myFSS),
            &actualSize);
   }
   if (err != noErr)
         return err;
   if ((returnedType == typeLetterSpec) ||
         (returnedType == typeAlias) ||
         (returnedType == typeFSS)) {
      err = MyHandleOpenDoc(diskForm, &myFSS, &myLetterSpec);
      if (err != noErr)
         return err;
   }
}
```

The `MyHandleOpenDoc` function shown in Listing 3-11 uses this information to open a letter in the mailbox or on disk. The `SMPOpenLetter` function (page 3-94) registers with

the Standard Mail Package the window passed to it and associates it with the letter identified in the `LetterDescriptor` structure. The `SMPGetMainEnclosureFSSpec` function (page 3-103) then extracts the native format document from the letter, and an application-defined content-drawing routine (`MyDrawLetterContent`, in this example) draws the document into the window.

Listing 3-11 Opening a letter

```
OSErr MyHandleOpenDoc(Boolean diskForm, FSSpec *myFSS,
                                  LetterSpec *myLetterSpec)
{
   OSErr             err;
   LetterDescriptor  letterDesc;
   Point             upLeft = (0, 0);
   Rect              newWindowRect;

   letterDesc.diskForm = diskForm;
   if (diskForm)
   {
      letterDesc.fileSpec = *myFSS;
   }
   else
   {
      letterDesc.fileSpec = *myLetterSpec;
   }

   newWindow = MyMakeWindow(kDrawMailerWindow, &newWindowRect,
                                                  "\pTitle", false);
   if (newWindow == NULL)
   {
      MyErrorAlert(memFullErr, "\pSMPOpenLetter");
      return memFullErr;
   }

   err = SMPOpenLetter(&letterDesc, newWindow, upLeft, true,
                           gPreferences.expandOnOpen, nil, 0L);
   if (err != noErr)
   {
      MyErrorAlert(err, "\pSMPOpenLetter");
      return err;
   }

   err = SMPGetMainEnclosureFSSpec(newWindow, &enclSpec);
```

```
   if (err != noErr)
   {
      MyErrorAlert(err, "\pSMPOpenLetter");
      return err;
   }

   return MyDrawLetterContent(newWindow, &enclSpec);
}
```

Forwarding and Replying to Mail

After opening a letter, the user has the option to reply or to forward it. The user can also remove the mailer, changing the letter into a regular document.

To forward a letter, you must add a new mailer to the existing letter. A letter has a new mailer attached each time it is forwarded. The mailers form a set with each mailer superimposed upon the preceding mailers. The user can view the mailers in the set by clicking a dog-ear in the corner of the mailer window pane to cycle through the set or by choosing among the names in a pop-up menu appearing in the Forwarded By field.

The first step in forwarding a letter is to expand the existing mailer, if it is contracted. Next, you call the `SMPMailerForward` function (page 3-49) to create the new mailer and add it to the letter. Finally, you should adjust your menu items in the configuration appropriate for sending mail, which is done in Listing 3-12 by an application-defined function `MyFixMailerMenus`. The parameter constant `kDefaultIdentity` has a value of 0, with the effect described in "Authenticating a User" on page 3-36.

Listing 3-12 Forwarding a letter

```
err = SMPExpandOrContract(window, true);
/* Ignore errors if window is already expanded. */
err = SMPMailerForward(window, kDefaultIdentity);
if (err != noErr)
   MyErrorAlert(err, "\pSMPMailerForward");
MyFixMailerMenus(window);
```

The first step in replying to a letter is to create a new window in which the user will write the reply. When this new window exists, you can call the `SMPMailerReply` function (page 3-51), passing in among other parameters the new window and the existing letter window. The function causes the reply letter to be created, automatically addressed to the originator of the original letter. The code shown in Listing 3-13 illustrates how to handle replying to a letter.

Listing 3-13 Replying to a letter

```
replyWindow = MyMakeWindow(kDrawMailerWindow, &newWindowRect,
                  newTitle, false);
err = SMPMailerReply(window, replyWindow, replyToAll, topLeft,
                  true, true, kDefaultIdentity, nil, 0L);
if (err != noErr)
   MyErrorAlert(err, "\pSMPMailerReply");
ShowWindow(replyWindow);
```

The application-defined function `MyMakeWindow` creates a window and adjusts its content area to accommodate the mailer. The `SMPMailerReply` function adds the mailer, and the `ShowWindow` function causes the window to become visible.

Closing a Letter

Closing a letter window requires you to adhere to a short procedure: displaying the close-options dialog box, checking for open enclosures and in-progress copy operations, removing the mailer from the window, and closing the window.

Before closing a letter window, you can display the close-options dialog box, which gives the user an opportunity to delete the letter or tag it before closing it. Listing 3-14 assumes the existence of a data structure `gPreferences` containing user-preference flags, including one determining whether or not you should display the close-options dialog box. The code uses these preferences also to fill in the default values in the close-options dialog box when it is displayed by the `SMPCloseOptionsDialog` function (page 3-60).

Listing 3-14 Preparing to close a letter

```
if (gPreferences.closeOptionsDialog) {
   SetCursor(&qd.arrow);
   err = SMPCloseOptionsDialog(window
                     &gPreferences.closeOptions);
   if (err != noErr)
      returnValue = false;
}
```

The next step in the letter-closing procedure is to ensure that there are no open enclosures attached to the letter, that there are no Finder copy operations in progress, and that there are no other conditions that prevent closing the window. Finder copy operations occur when the user is in the process of copying a document to or from the enclosures list. If either situation is true, or if for some other reason a nonzero result was returned from the `SMPPrepareToClose` function, the application-defined function `MyStopAlert` notifies the user and prevents the letter from closing. Listing 3-15 illustrates these checks.

Listing 3-15 Checking status prior to closing a letter

```
err = SMPPrepareToClose(window);
if (err == kSMPHasOpenAttachments) {
   SetCursor(&qd.arrow);
   MyStopAlert(kMyHasOpenAttachID, nil);
   returnValue = false;
}
else if (err == kSMPCopyInProgress) {
   SetCursor(&qd.arrow);
   MyStopAlert(kMyCopyInProgress, nil);
   returnValue = false;
}
else if (err != noErr) {
   SetCursor(&qd.arrow);
   MyStopAlert(kMyCannotCloseWindow, nil);
   returnValue = false;
}
```

The final steps in the closing procedure are to remove the mailer from the window and close the window, as shown in Listing 3-16. The `SMPDisposeMailer` function (page 3-61) removes the mailer from the window passed in as a parameter and releases the memory associated with the letter window. Then the application-defined routine `MyDestroyWindow` disposes of the rest of the window and document structures in memory.

Listing 3-16 Closing the letter

```
err = SMPDisposeMailer(window, closeOptions);
if (err != noErr)
   MyErrorAlert(err, "\pSMPDisposeMailer");
return MyDestroyWindow(window);
```

Handling Mailer Events

The general strategy for handling events in a window with a mailer is to hand the events to the Standard Mail Package first. The Standard Mail Package has built-in routines to handle many events, including mouse-down events, key-down events, update events for the mailer, activate events, deactivate events, and null events. The Standard Mail Package then hands the event back to the application with an indication that either it handled the event or your application must handle the event.

Your application should retrieve events in the normal manner, with the `WaitNextEvent` system call. When a mailer window is frontmost, call the `SMPMailerEvent` function (page 3-63), passing in the event record. The

SMPMailerEvent function returns a set of flags in its whatHappened parameter indicating what action it took, if any, and whether your application must handle the event. (These flags, of type SMPMailerResult, are described on page 3-65.) Your application can then process the event appropriately. The event-handling function shown in Listing 3-17 receives the event record following a WaitNextEvent call, calls SMPMailerEvent, and passes the SMPMailerResult value to an application-defined routine named MyProcessWhatHappened. The parameter of type WInfoPtr is a pointer to an application-defined data structure containing status information about the mailer window.

Listing 3-17 Processing events in a mailer window

```
void *MyMailerEventHandler(WindowPtr window, WInfoPtr infoPtr,
                           EventRecord *ev)
{
   SMPMailerResult   whatHappened;
   OSErr             err;

   err = SMPMailerEvent(ev, &whatHappened, nil, 0L);
   if (err != noErr)
      MyErrorAlert(err, "\pSMPMailerEvent");
   return (void *)(MyProcessWhatHappened(window, infoPtr,
                                         whatHappened));
}

Boolean MyProcessWhatHappened(WindowPtr window, WInfoPtr infoPtr,
                              SMPMailerResult whatHappened)
{
   OSErr          err;
   SMPMailerState state;
   long           *lastChanged;

   /* Check if mailer has changed since last menu adjustment. */
   err = SMPGetMailerState(window, &state);
   if (err != noErr)
      MyErrorAlert(err, "\pSMPGetMailerState");
   lastChanged = (long *)&infoPtr->otherData[kLastChangedData];
   if (*lastChanged != state.changeCount) {
      *lastChanged = state.changeCount;
      infoPtr->changed = true;
      MyFixMailerMenus(window, infoPtr);
   }
   if ((whatHappened & kSMPContractedMask) != 0)
```

```
      MyHandleContract(window,infoPtr);
   if ((whatHappened & kSMPExpandedMask) != 0)
      MyHandleExpand(window, infoPtr);
   if ((whatHappened & kSMPMailerBecomesTargetMask) != 0 ||
         (whatHappened & kSMPAppBecomesTargetMask) != 0))
      MyFixMailerMenus(window, infoPtr);
   /* Check menus for every event the mailer handles. */
   if ((whatHappened & kSMPAppShouldIgnoreEventMask) != 0)
      MyFixMailerMenus(window, infoPtr);
   if ((whatHappened & kSMPAppMustHandleEventMask) != 0)
      return false; /* app must handle this event */
   else return true; /* mailer handled this event completely */
}
```

Most of the postprocessing of the event involves adjusting the menus, because the mailer event may have affected which commands should be active. In addition, if the `kSMPContractedBit` flag or `kSMPExpandedBit` flag is set as a a result of the event, the code calls one of the application-defined routines: `MyHandleContract` or `MyHandleExpand`. These routines call the `SMPGetDimensions` function to determine the size of the expanded or contracted mailer, so that the application can adjust the size of the content region of the window. If the user wants to expand the mailer, you must then call the `SMPExpandOrContract` function (page 3-56) to expand the mailer to its full size. However, if the user wants to contract the mailer to a single line, you need not call `SMPExpandOrContract` because the Standard Mail Package performs the contraction; you need only adjust the size of your content region and invalidate it to update its content.

In addition, the Standard Mail Package requires you to add some logic to your application's mouse-click handler for a window that includes a mailer. You must notify the Standard Mail Package before you allow the user to change the content of a letter, to accommodate the needs of its digital signature capability. Before changing the letter, you must call the `SMPPrepareToChange` function (page 3-83); if the letter has been digitally signed, a dialog box appears warning the user that the impending change will invalidate the signature. As in Listing 3-18, your routine should check the return value from `SMPPrepareToChange` and exit if the user has clicked the Cancel button in the dialog box.

The Standard Mail Package maintains its own undo buffer to support undoing mailer operations. You must clear this buffer before doing operations on data in the content area of your window so that only one undo operation is pending for the window. After calling the application's click-handler function, if the letter's contents have changed, you should call the `SMPContentChanged` function (page 3-76).

Listing 3-18 Handling a mouse click in a mailer window

```
void *MyMailerMouseClickHandler(WindowPtr window,
                                WInfoPtr infoPtr)
{
   void        *returnVal;
   OSErr       err;
   Boolean     alreadyChanged;

   /* Make sure you can change the letter. */
   alreadyChanged = infoPtr->changed;
   if (!alreadyChanged) {
      err = SMPPrepareToChange(window);
      if (err == userCanceledErr)
         return nil;
   }

   /* Since content is changing, clear mailer undo buffer. */
   err = SMPClearUndo(window);
   if (err != noErr)
      MyErrorAlert(err, "\pSMPClearUndo");

   /* Call app's click handler. */
   returnVal = MyClickHandler(window, infoPtr);
   if (!alreadyChanged && infoPtr->changed) {
      err = SMPContentChanged(window);
      if (err != noErr)
         MyErrorAlert(err, "\pSMPContentChanged");
   }

   return returnVal;
}
```

The previous section alluded to the undo buffer kept by the Standard Mail Package to support undo operations in the mailer portion of letters. The Standard Mail Package supports the Clipboard-based edit commands Cut, Copy, Paste, Clear, Select All, as well as the Undo command. The function shown in Listing 3-19 is a mailer Cut command handler; it shows how to support the Clipboard by calling the `SMPMailerEditCommand` function (page 3-67), then processing the result by calling the application-defined `MyProcessWhatHappened` function. You can use a similar strategy for the Copy, Paste, Clear, Select All, and Undo commands.

Listing 3-19 Supporting the Clipboard in a mailer edit command

```
void *MyMailerCutCommand(WindowPtr window, WInfoPtr infoPtr)
{
   OSErr             err;
   SMPMailerResult   whatHappened;

   err = SMPMailerEditCommand(window, kSMPCutCommand,
                                          &whatHappened);
   if (err != noErr)
      MyErrorAlert(err, "\pSMPMailerEditCommand");
   return (void *)(MyProcessWhatHappened(window, infoPtr,
                                              whatHappened));
}
```

Standard Mail Package Reference

This section describes the data types and routines provided by the Standard Mail Package.

Data Structures

The Standard Mail Package routines use the data types described in this section.

Recipient Descriptor

The recipient descriptor, used by the `SMPSendLetter` and `SMPResolveToRecipient` functions, describes an addressee for a message or letter.

Note

You must call the `DisposePtr` function to deallocate the `recipient` field before you can dispose of the recipient descriptor. u

```
struct SMPRecipientDescriptor
{
   struct SMPRecipientDescriptor *next;      /* pointer to next element */
   OSErr                         result;     /* result code */
   OCEPackedRecipient            *recipient; /* packed recipient address */
   unsigned long                 size;       /* size of recipient address */
   MailRecipient                 theAddress; /* unpacked recipient address */
   RecordID                      theRID;     /* record ID of recipient */
};
```

Field descriptions

`next`	A pointer to the next element in a linked list of recipient descriptors. This field must be set to `nil` in the last descriptor in the list.
`result`	The result code returned by the `SMPSendLetter` function. If the `SMPSendLetter` function fails because of a bad recipient descriptor, you can examine this field in each of the recipient descriptors to determine which caused the problem.
`recipient`	A pointer to the packed address of the recipient of the letter.
`size`	The length, in bytes, of the recipient's address.
`theAddress`	The unpacked address of the recipient.
`theRID`	The record ID of the recipient. If the `SMPSendLetter` function fails because of a bad recipient descriptor, you can use this record ID to determine the name of the addressee that caused the error.

Enclosure Descriptor

The enclosure descriptor is an element of a linked list that describes an enclosure to be sent with a letter. See the description of the `SMPSendLetter` function on page 3-37 for more information about the use of this data structure.

```
struct SMPEnclosureDescriptor
{
   struct SMPEnclosureDescriptor *next;         /* pointer to next element */
   OSErr                         result;        /* result code */
   FSSpec                        fileSpec;      /* file specifier  */
                                                /*  of enclosure */
   OSType                        fileCreator;   /* creator of enclosure */
   OSType                        fileType;      /* file type of enclosure */
};
```

Field descriptions

`next`	A pointer to the next element in the linked list. If this is the only or last element in the list, set this field to `nil`. If you use the `SMPResolveToRecipient` function to create the linked list, the function fills in this field for you.
`result`	The result code returned by the `SMPSendLetter` function. If the `SMPSendLetter` function fails because of a bad enclosure descriptor, you can examine this field in each of the enclosure descriptors to determine which caused the problem.
`fileSpec`	File specifier of the enclosure.
`fileCreator`	File creator of the enclosure. The `SMPSendLetter` function uses this field only if you send the enclosure directly as a file (that is, you set the `sendAs` field of the parameter block for the `SMPSendLetter` function to `kSMPSendFileOnlyMask`).

`fileType` File type of the enclosure. The `SMPSendLetter` function uses this field only if you send the enclosure directly as a file (that is, you set the `sendAs` field of the parameter block for the `SMPSendLetter` function to `kSMPSendFileOnlyMask`).

Letter Descriptor

The letter descriptor, used by the `SMPOpenLetter` (page 3-94) and `SMPGetNextLetter` (page 3-97) functions, identifies a letter in the In Tray or on disk.

```
struct LetterDescriptor {
   Boolean onDisk;
   union {
      FSSpec fileSpec;
      LetterSpec mailboxSpec;
   }u;
};
```

Field descriptions

`onDisk` A Boolean value that indicates whether the letter is on disk or in the In Tray. If this value is set to `true`, the file is on disk.

`fileSpec` The file specification structure of the letter. Use this field of the structure if the file is on disk.

`mailboxSpec` The letter specification structure of the letter. Use this field if the letter is in the In Tray. When the user double-clicks a letter in the In Tray and the letter's creator is your application, you receive an `'aevt' 'odoc'` Apple event that includes this specifier. The `LetterSpec` structure is defined on page 3-35.

Letter Information Structure

The letter information structure, which is used by the `SMPGetLetterInfo` (page 3-93) function, describes a letter in the In Tray.

```
struct SMPLetterInfo {
   OSType      letterCreator;
   OSType      letterType;
   RString32   subject;
   RString32   sender;
};
```

Field descriptions

`letterCreator` The creator of the letter. The field indicates what application created the letter and is identical to the creator used by the application for files.

`letterType`	The letter type, which is identical to the file type that the creating application would use for the letter. Letters containing only AOCE standard content are of type `'lttr'`.
`subject`	The contents of the Subject field in the mailer.
`sender`	The contents of the From field in the mailer.

Creator Type Structure

The Standard Mail Package uses the creator type structure to specify block types. The creator type structure is defined by the `OCECreatorType` data type.

```
struct OCECreatorType {
   OSType   msgCreator;    /* block creator */
   OSType   msgType;       /* block type */
};
```

Field descriptions

`msgCreator`	The creator of the block. You can specify any four-character value in this field; usually it is the signature of your application that adds the block of data to the letter. For example, the creator of a block added by the AppleMail application provided with the AOCE software is `'apml'`.
`msgType`	The type of the block. You can define your own four-character block types to serve your own purposes. Apple Computer, Inc., reserves all block types consisting entirely of lowercase letters. For example, the type of an image block as defined by the AppleMail application is `'imag'`.

Image Block Information Structure

An image block in a letter (a block with a creator type of `'apml'` and a block type of `'imag'`) starts with an image block information structure, defined by the `TPfPgDir` data type (defined by the Printing Manager).

```
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 resource 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].

Allocate and lock down a buffer equal to the size of the page. Then call the SMPReadBlock function (page 3-106) with the pointer to that buffer in the buffer parameter and the offset iPgPos[*n* – 1] in the dataOffset parameter.

Letter Parameter Block

The SMPSendLetter function uses the SMPLetterPB parameter block. The fields of the parameter block are described with the SMPSendLetter function on page 3-38.

```
struct SMPLetterPB
{
   OSErr                       result;        /* function result */
   RStringPtr                  subject;       /* subject of letter */
   AuthIdentity                senderIdentity;/* identity of sender */
   SMPRecipientDescriptorPtr   toList;        /* list of addressees */
   SMPRecipientDescriptorPtr   ccList;        /* list of cc addressees */
   SMPRecipientDescriptorPtr   bccList;       /* list of bcc addressees */
   ScriptCode                  script;        /* script code for language */
   Size                        textSize;      /* length of body data */
   Ptr                         textBuffer;    /* body of the letter */
   SMPPSendAs                  sendAs;        /* file, enclosure, or image */
   Byte                        padByte;       /* reserved */
   SMPEnclosureDescriptorPtr   enclosures;    /* files to be enclosed */
   SMPDrawImageProcPtr         drawImageProc; /* your imaging routine */
   long                        imageRefCon;   /* for your use */
   Boolean                     supportsColor; /* true for a color grafPort */
};
```

Close-Options Structure

The SMPCloseOptionsDialog function (page 3-60) and the SMPDisposeMailer function (page 3-61) use the close-options structure to specify what actions the Standard Mail Package should take when the user closes a letter in the In Tray. The close-options structure is defined by the SMPCloseOptions data type.

```
struct SMPCloseOptions {
   Boolean              moveToTrash;
   Boolean              addTag;
   RString32            tag;
};
```

Field descriptions

`moveToTrash`	Move the letter from the In Tray to the Trash. You should not set this field to `true` if the `addTag` field is set to `true`.
`addTag`	Tag the letter with the value in the `tag` field. You should not set this field to `true` if the `moveToTrash` field is set to `true`.
`tag`	The tag to attach to the letter. This field must contain a valid tag if the `addTag` field is set to `true`. A tag can be any alphanumeric string up to 32 bytes in length.

Mailer-State Structure

The `SMPGetMailerState` function (page 3-69) uses the mailer-state structure to return information about a mailer in a specified window. The mailer-state structure is defined by the `SMPMailerState` data type.

```
struct SMPMailerState {
   short                mailerCount;
   short                currentMailer;
   Point                upperLeft;
   Boolean              hasBeenReceived;
   Boolean              isTarget;
   Boolean              isExpanded;
   Boolean              canMoveToTrash;
   Boolean              canTag;
   Byte                 padByte2;
   unsigned long        changeCount;
   SMPMailerComponent   targetComponent;
   Boolean              canCut;
   Boolean              canCopy;
   Boolean              canPaste;
   Boolean              canClear;
   Boolean              canSelectAll;
   Byte                 padByte3;
   SMPUndoState         undoState;
   Str63                undoWhat;
};
```

Field descriptions

`mailerCount` The number of mailers in the mailer set associated with the window. This number is incremented by 1 each time the letter is forwarded. You should enable the Reply item in the Mail menu if the `mailerCount` field is greater than 1.

`currentMailer` The number of the mailer that the user is currently looking at. The original mailer for the letter is number 1, and each forwarding mailer is numbered sequentially.

`upperLeft` The upper-left corner of the mailer in the window's local coordinates.

`hasBeenReceived`
A Boolean value that indicates whether the most recent mailer has been received (that is, it was sent to the current user). If set to `true` (that is, if the mailer has been received), then the user cannot edit the fields in the mailer but can forward or reply to the letter. You should enable the Forward and Reply items in the Mail menu. If it is set to `false`, the current user is the originator of the letter or has added a new mailer to forward the letter, and might still be working on the letter, so you should disable the Forward item.

`isTarget` A Boolean value that indicates whether the mailer is the target; that is, whether the user is working in the mailer so that key-down events apply to the mailer rather than to the portion of the window that you control. Note that the Event Manager sends all events that take place in your window— including in the mailer—to your application. If you pass every event to the `SMPMailerEvent` function (page 3-63), that function returns a value that tells you whether you have to handle the event.

`isExpanded` A Boolean value that indicates whether the mailer is in the expanded state or contracted state.

`canMoveToTrash`
A Boolean value that indicates whether to enable the Close and Delete item in the File menu. The standard interface is to enable this item for a letter that is in the In Tray, but not for one that has been saved to disk.

`canTag` A Boolean value that indicates whether to enable the Tag item in the Mail menu. The user can add a tag to a letter that is in the In Tray, but not to a letter that has been saved to disk. See the `SMPTagDialog` function (page 3-58) to see how to implement the Tag item in the Mail menu.

`changeCount` A value that indicates whether the mailer has been changed. If this field is set to a nonzero value, the mailer has been changed since the last time it was saved. If this number has changed since the last time you checked it, then the mailer has been changed during that period.

`targetComponent`
A constant that indicates which of the fields in the mailer the user is working in. Possible values for this field are listed immediately following these field descriptions.

`canCut`	A Boolean value that indicates whether you should enable the Cut item in the Edit menu. This field is significant only if the `isTarget` field is set to `true`.
`canCopy`	A Boolean value that indicates whether you should enable the Copy item in the Edit menu. This field is significant only if the `isTarget` field is set to `true`.
`canPaste`	A Boolean value that indicates whether you should enable the Paste item in the Edit menu. This field is significant only if the `isTarget` field is set to `true`.
`canClear`	A Boolean value that indicates whether you should enable the Clear item in the Edit menu. This field is significant only if the `isTarget` field is set to `true`.
`canSelectAll`	A Boolean value that indicates whether you should enable the Select All item in the Edit menu. This field is significant only if the `isTarget` field is set to `true`.
`undoState`	A constant that you can use to determine whether you should enable the Undo item in the Edit menu. See the description of the `SMPClearUndo` function on page 3-70 for information on clearing the undo buffer. The possible values for this field are described following these field descriptions.
`undoWhat`	A string that indicates the action that the reader should undo or redo. You should use this string in place of the word "Undo" or "Redo" in the Edit menu. For example, if the user just used the Edit menu to cut a word from the subject field in the mailer, the `undoWhat` field is set to the string `Undo Cut`. This field is significant only if the `undoState` field equals `kMailerUndo`.

Here are the possible values for the `targetComponent` field. These values are also used by the `SMPBecomeTarget` function (page 3-54), the `SMPGetComponentSize` function (page 3-110), the `SMPGetComponentInfo` function (page 3-111), and the `SMPGetListItemInfo` function (page 3-113).

```
enum {
   kSMPOther         = -1,
   kSMPFrom          = 32,
   kSMPTo            = 20,
   kSMPRegarding     = 22,
   kSMPSendDateTime  = 29,
   kSMPAttachments   = 26,
   kSMPAddressOMatic = 16
};

typedef unsigned long SMPMailerComponent;
```

Constant descriptions

`kSMPOther` No field, or some field other than those indicated by the other enumerated values (such as the Signature field).

`kSMPFrom` The From field in a mailer for a new letter, or the Forwarded By field in a mailer for a forwarded letter.

`kSMPTo` The Recipients field.

`kSMPRegarding` The Subject field.

`kSMPSendDateTime` The Sent field.

`kSMPAttachments` The Enclosures field.

`kSMPAddressOMatic` The addressing panel (see Figure 3-4 on page 3-6).

Your application and the mailer maintain independent undo buffers. The mailer keeps track of which undo buffer should currently be in use and passes this information to you in the `undoState` field of the mailer-state structure. You can use this information to determine which items in the Edit menu to enable and whether to clear your application's undo buffer. The possible values for the `undoState` field are as follows:

```
enum {
   kSMPUndoDisabled,
   kSMPAppMayUndo,
   kSMPMailerUndo
};

typedef unsigned short SMPUndoState;
```

Constant descriptions

`kSMPUndoDisabled` The Standard Mail Package has cleared its undo buffer after executing a command that the user cannot undo. Therefore, there is currently no action in the mailer or in your application that the user can undo. You should disable the Undo item in the Edit menu and clear your application's undo buffer.

`kSMPAppMayUndo` The Standard Mail Package has not executed a command that the user may undo. Therefore, there is no action in the mailer that the user can undo, but the Standard Mail Package can't tell whether there is an action in your application that the user can undo. You should enable the Undo item in the Edit menu only if your application has executed a command that the user may undo. If the user has taken an action in the content portion of the window that the user can undo or that should cause the undo buffer to be cleared, you must also call the `SMPClearUndo` function (page 3-70) to tell the Standard Mail Package to clear its undo buffer.

`kSMPMailerUndo`

The Standard Mail Package has executed a command that the user may undo. Therefore, the latest action that the user can undo was in the mailer, and there is no action in your application that the user can undo. You should enable the Undo item in the Edit menu and display the string returned in the `undoWhat` field of the `SMPMailerState` structure. You should also clear your application's undo buffer. If the user chooses the Undo item in the Edit menu, call the `SMPMailerEditCommand` function to allow the Standard Mail Package to handle the undo operation.

Send-Options Structure

The Standard Mail Package maintains a set of options for each letter. There is a default value for each option, but before you send a letter, you should give the user the opportunity to change the send options for that letter. You can call the `SMPSendOptionsDialog` function (page 3-73) to provide the user with a dialog box that sets these options. The `SMPSendOptionsDialog` function returns the send-options structure, defined by the `SMPSendOptions` data type.

```
struct SMPSendOptions {
   Boolean     signWhenSent;
   IPMPriority priority;
};
```

Field descriptions

`signWhenSent` A Boolean value that indicates whether a digital signature should be added to the letter when you send it. If this field is set to `true`, the Standard Mail Package prompts the user for a signature when you send the letter.

`priority` A constant that indicates the priority of the message. The Standard Mail Package includes the priority information in the In and Out Trays.

Send-Format Structure

The Standard Mail Package uses two standard formats and allows applications to send or open letters in any number of "native" formats known to the application. The two standard formats used by the Standard Mail Package are standard interchange format and image format. Native formats for the SurfWriter word processing program might be SurfWriter, TIFF, and SGML, for example.

Before you send a letter using the Standard Mail Package, you call the `SMPSendOptionsDialog` function (page 3-73). This function displays a dialog box that lets the user indicate which format or formats to use when sending the letter and returns the user's choices to you in a send-format structure.

The send-format structure includes a `whichFormats` field that indicates whether you should send the document in a format designed to be read by your application (`kSMPNativeBit`), as an image designed to be read by any application that reads AOCE image files (`kSMPImageBit`), or in standard interchange format (`kSMPStandardInterchangeBit`).

```
enum {
   kSMPNativeBit,
   kSMPImageBit,
   kSMPStandardInterchangeBit
};

/* values of SMPSendFormatMask */
enum {
   kSMPNativeMask               = 1<<kSMPNativeBit,
   kSMPImageMask                = 1<<kSMPImageBit,
   kSMPStandardInterchangeMask  = 1<<kSMPStandardInterchangeBit,
};
typedef unsigned long SMPSendFormatMask;
```

The send-format structure is defined by the `SMPSendFormat` data type.

```
struct SMPSendFormat {
   SMPSendFormatMask    whichFormats;
   short                whichNativeFormat;   /* 0 based */
};
```

The `whichNativeFormat` field is an index number (starting with 0) that indicates which one of your application's native formats has been selected by the user, or, in the case of a received letter, which native format is currently in the letter. The index number refers to the array of string pointers you pass to the `SMPSendOptionsDialog` function in the `nativeFormatNames` parameter. The `whichNativeFormat` field is significant only if the `whichFormats` field has the `kSMPNativeBit` set to 1.

Letter-Specification Structure

The letter-specification structure is a data structure that you receive from an `'aevt' 'odoc'` Apple event and pass to the `SMPOpenLetter` function (page 3-94). The content of this data structure is private to the AOCE toolbox.

```
struct LetterSpec
{
   unsigned long     spec[3];
};
```

Standard Mail Package Functions

The following sections describe the routines provided by the AOCE Standard Mail Package. Several Standard Mail Package routines require you to provide an authentication identity as input. The chapter "Standard Catalog Package" in this book describes a routine that prompts the user for a name and password, authenticates the user, and returns the authentication identity number to your application.

The routines in this chapter are divided into two main sections, reflecting the two parts of the Standard Mail Package:

- Send-letter functions, which provide a very simple way to send a letter or a file.
- Mailer functions, which provide a standard user interface for sending and opening your application's documents as letters.

A final section, "Application-Defined Functions," describes some callback routines that you can provide to support Standard Mail Package features.

Assembly-Language Interface

To call a Standard Mail Package routine from assembly language, you must do the following:

1. Push space for the function result and all routine parameters (in Pascal calling-convention order) on the stack.
2. Put in the D0 register a long word consisting of the parameter word count for the routine followed by the routine selector. The parameter word count indicates how many words of parameters you are placing on the stack; for example, if the function has two parameters and each is a pointer, the parameter word count for the function is $0004.
3. Call the Standard Mail Package trap, $AA5D.

Each routine description in the following sections lists the parameter word count and routine selector for that routine.

Authenticating a User

Before the first time you send a message, you must provide identification to prove that the caller is an authorized user of the system. The `SDPPromptForID` function described in the chapter "Standard Catalog Package" in this book provides dialog boxes that allow the user to identify himself or herself as one of the authorized users of the system and returns an identification number (the *authentication identity*) for the user. You can use the authentication identity in all subsequent calls to Standard Mail Package and other AOCE routines that require it.

However, the Standard Mail Package implements a special scheme to ease handling authentication identities for its routines. That is, you can pass a value of 0 for the authentication identity parameter to those functions requiring it. The effect of passing the 0 parameter value varies according to the situation. The first time you pass 0 after initializing the Standard Mail Package, the system uses the local identity (see the chapter "Standard Catalog Package" in this book for a description of local and specific identities).

If you forward or reply to a letter, the Standard Mail Package uses the identity for the mailbox the letter was in: a visitor's mailbox produces the visitor's identity; the main mailbox produces the local identity. In all other cases, if you pass 0 for the authentication identity parameter, the Standard Mail Package uses the last identity (local or specific) used by the user.

Send-Letter Functions

You can use the functions in this section to send a document as a letter with enclosures, as an image, or as a file. The `SMPSendLetter` function sends the document. If you want to send the document as an image, you must provide an image-drawing routine that calls the `SMPNewPage` function each time it images a page of the document.

You can obtain catalog system specification (`DSSpec`) structures for the recipients of the letter by using the dialog boxes or the Catalog-Browsing panel described in the chapter "Standard Catalog Package" in this book. You can use the `SMPResolveToRecipient` function described in this section to transform the `DSSpec` structures into a linked list of mail addresses, and you can use this linked list as input to the `SMPSendLetter` function.

The `SMPSendLetter` function includes as a parameter a pointer to a parameter block. The routine description includes a list of the parameter block fields for which you must provide values or that return values to you. Each parameter block field list in the routine description consists of four columns, as described in the Preface of this book.

SMPSendLetter

The `SMPSendLetter` function sends a letter, an image, or a file.

```
pascal OSErr SMPSendLetter(SMPLetterPBPtr theLetter);
```

`theLetter` Pointer to a parameter block.

Parameter block

←	result	OSErr	Result code
→	subject	RStringPtr	Subject of letter
→	senderIdentity	AuthIdentity	Identity of sender
→	toList	SMPRecipientDescriptorPtr	List of recipients
→	ccList	SMPRecipientDescriptorPtr	List of cc recipients
→	bccList	SMPRecipientDescriptorPtr	List of bcc recipients
→	script	ScriptCode	Script code
→	textSize	Size	Length of text
→	textBuffer	Ptr	Letter text
→	sendAs	SMPPSendAs	Letter, image, or file
→	enclosures	SMPEnclosureDescriptorPtr	Enclosed files
→	drawImageProc	SMPDrawImageProcPtr	Image-drawing routine
→	imageRefCon	long	For your use
→	supportsColor	Boolean	Set to true for a color graphics port

Field descriptions

result — The function result. This field contains the same result code as the function return value.

subject — The subject string for the letter.

senderIdentity — Authentication identity of the sender.

toList — A pointer to a linked list of recipient descriptors for the main addressees of the letter. You can use the SMPResolveToRecipient function to create this list.

ccList — A pointer to a linked list of recipient descriptors for the "carbon copy" (cc) addressees of the letter. You can use the SMPResolveToRecipient function to create this list.

bccList — A pointer to a linked list of recipient descriptors for the "blind carbon copy" (bcc) addressees of the letter. You can use the SMPResolveToRecipient function to create this list.

script — Language of letter text. This is a script code from the Script Manager. You cannot use the values smSystemScript or smCurrentScript for this parameter. The function ignores this field if you set the sendAs field to kSMPSendFileOnlyMask.

textSize — Number of bytes in the text of the letter. The function ignores this field if you set the sendAs parameter to kSMPSendFileOnlyMask.

textBuffer — A pointer to the buffer that contains the text of the letter. The function ignores this field if you set the sendAs field to kSMPSendFileOnlyMask.

sendAs	A constant that indicates whether to send the message as an image (kSMPSendAsImageMask), to send the message as a letter with enclosures (kSMPSendAsEnclosureMask), to send an enclosed file so that it appears in the In Tray as the file itself rather than as a letter (kSMPSendFileOnlyMask), or to send some combination of these formats. You cannot combine the send-file-only and send-as-enclosure formats.
enclosures	A pointer to a linked list of enclosure descriptors. If you specify kSMPSendFileOnlyMask for the sendAs field, you can include only one enclosure. In this case, the enclosure descriptor must provide values for the file creator and type that are appropriate for the file being sent in order for the Finder to display the file correctly.
drawImageProc	A pointer to your image-drawing routine. If you want to send a letter as an image, you must provide a routine to draw the image. The procedure declaration for this routine is described on page 3-123. The function ignores this field if you do not set the sendAs field to send the file as an image.
imageRefCon	A reference constant for your use. The function passes this value to your image-drawing routine.
supportsColor	A Boolean value that indicates whether the procedure pointed to by the drawImageProc parameter is capable of drawing in color. The Standard Mail Package provides a color graphics port to your image-drawing routine only if you specify true for the supportsColor field and the user has color QuickDraw.

DESCRIPTION

The SMPSendLetter function provides no user interface. Your application must determine the subject, text, enclosures, and addressees for the letter either by providing its own user interface or through some other means. You can use the SDPGetDirectories, SDPFindRecord, SDPNewPanel, or SDPGetNewPanel functions to provide a user interface for selecting an addressee.

If the SMPSendLetter function returns with a result code that indicates a bad recipient descriptor or a bad enclosure descriptor, you can check the result field of each descriptor in the linked list to determine which one was bad. Look in the filename field of the bad enclosure descriptor for the name of the file that caused the problem. The theRID field of the recipient descriptor contains the record ID containing the name of the addressee. For example, an RStringPtr structure pointing to the name of the addressee represented by the first recipient descriptor of the Recipients list is located in theLetter->toList->theRID.local.name.

You cannot specify the values smSystemScript or smCurrentScript for the script parameter. To obtain the system script, call the GetScriptManagerVariable function with a selector of smSysScript. To obtain the current script, call the FontScript function.

The SMPSendLetter function can send a letter as a note with optional enclosures, as an image of the note and enclosures, as the document file alone, or as some combination of

these formats. Use one or a combination of the following constants in the sendAs field to specify the format for the letter:

```
enum {
   kSMPSendAsEnclosureBit, /* appears as letter with enclosures */
   kSMPSendFileOnlyBit,    /* appears as a file in mailbox. */
   kSMPSendAsImageBit      /* letter includes image of content */
};

/* values of SMPPSendAs */
enum {
   kSMPSendAsEnclosureMask = 1<<kSMPSendAsEnclosureBit,
   kSMPSendFileOnlyMask    = 1<<kSMPSendFileOnlyBit,
   kSMPSendAsImageMask     = 1<<kSMPSendAsImageBit
};

typedef Byte SMPPSendAs;
```

Constant descriptions

kSMPSendAsEnclosureMask
: The SMPSendLetter function sends the letter as a note with the text pointed to by the textBuffer parameter and the enclosure specified by the enclosure descriptor.

kSMPSendFileOnlyMask
: The enclosed file appears directly in the recipient's In Tray as the file itself rather than as a letter with an enclosure. If you specify this value for the sendAs parameter, the letter can contain only one enclosure.

kSMPSendAsImageMask
: The SMPSendLetter function converts the note into an image and calls your image-drawing routine to convert the enclosures into an image.

To combine formats, perform a bitwise OR operation on the appropriate constants. For example, to send a document as both a note with enclosures and as an image, set the sendAs parameter to kSMPSendAsEnclosureMask PLUS kSMPSendAsImageMask in Pascal or kSMPSendAsEnclosureMask OR kSMPSendAsImageMask in assembly language or C. You cannot combine the send-as-file format (kSMPSendFileOnlyMask) with the note-with-enclosures format (kSMPSendAsEnclosureMask).

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

You cannot combine the document-only format (kSMPSendFileOnlyMask) with the note-with-enclosures format (kSMPSendAsEnclosureMask). If you attempt to do so, the function returns the paramErr result code.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$01F4

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in a parameter value

SEE ALSO

The procedure declaration for your image-drawing routine is described on page 3-123.

The enclosure descriptor is defined in "Enclosure Descriptor" on page 3-26.

The recipient descriptor is defined in "Recipient Descriptor" on page 3-25.

The `StGetScriptManagerVariable` function and `FontScript` function are described in *Inside Macintosh: Text* in the chapter "Script Manager."

You can obtain record IDs for the recipients of the letter by using the dialog boxes or the Catalog-Browsing panel described in the chapter "Standard Catalog Package" in this book.

You can create a linked list of record descriptors from the recipient record IDs by calling the `SMPResolveToRecipient` function described on page 3-44.

SMPNewPage

The `SMPNewPage` function creates a new page for use by your image-drawing routine.

```
pascal OSErr SMPNewPage(OpenCPicParams *newHeader);
```

`newHeader` Pointer to an `OpenCPicParams` structure (see the chapter "Color QuickDraw" in *Inside Macintosh: Imaging With QuickDraw*). The `SMPNewPage` function sets the size of your graphics port rectangle equal to the size of the source rectangle you specify in this structure, and sets the image's horizontal and vertical resolutions to those you specify in this structure. For the normal resolution of the Macintosh screen, use 72 pixels per inch for both the vertical and horizontal resolutions.

DESCRIPTION

The `SMPSendLetter` or `SMPImage` function calls your image-drawing routine when you add an image to a letter you are sending. Your image-drawing routine then calls the `SMPNewPage` function before it draws each new page of an image file.

Note
You use the `hRes` and `vRes` fields in the `OpenCPicParams` structure to specify the horizontal and vertical resolutions of the image. Both of these fields are of type `Fixed`, which is a long word that contains an integer part in the high-order word and a binary fraction in the low-order word. To set the horizontal resolution to 72 dpi, for example, you specify a value of 0x00480000 for the `hRes` field to indicate an integer part with a value of 72 and no fractional part. If by mistake you simply specified a value of 72 (that is, 0x00000048) for the `hRes` field, you would be indicating an integer part with a value of 0 and a fractional part of 9/8192. Note also that you can use the `FixRatio` routine to create a value of type `Fixed` from two integer values representing a numerator and denominator. u

SPECIAL CONSIDERATIONS

If you change the graphics port within your image-drawing routine, you must change it back before calling the `SMPNewPage` function.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$0834

RESULT CODES

`noErr`	0	No error
`kSMPTooManyPages`	–1927	Image is more than 127 pages

SEE ALSO

The `SMPSendLetter` function is described on page 3-37.

The procedure declaration for your image-drawing routine is described on page 3-123.

The `OpenCPicParams` structure is described in the chapter “Color QuickDraw” in *Inside Macintosh: Imaging With QuickDraw*. The `FixRatio` routine is described in *Inside Macintosh: Operating System Utilities*.

SMPImageErr

The `SMPImageErr` function returns result codes from image-drawing routines.

```
pascal OSErr SMPImageErr(void);
```

DESCRIPTION

The `SMPSendLetter` or `SMPImage` function calls your image-drawing routine when you add an image to a letter you are sending. Your image-drawing routine calls the `SMPImageErr` function instead of calling the `QDError` function after it calls each QuickDraw routine. The `SMPImageErr` function returns both QuickDraw errors and errors returned by the `SMPAddBlock` function.

SPECIAL CONSIDERATIONS

If you change the graphics port within your image-drawing routine, you must change it back before calling the `SMPImageErr` function.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0000	$0835

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	Disk full
`pixmapTooDeepErr`	–148	Pixel map structure is deeper than 1 bit per pixel
`mfStackErr`	–149	Insufficient stack
`rgnTooBigErr`	–500	Bitmap would convert to a region greater than 64 KB

SEE ALSO

The `SMPSendLetter` function is described on page 3-37.

The procedure declaration for your image-drawing routine is described on page 3-123.

The `QDError` function is described in the chapter “Color QuickDraw” in *Inside Macintosh: Imaging With QuickDraw*.

The `SMPImageErr` function returns both QuickDraw errors and errors returned by the `SMPAddBlock` function (page 3-91).

SMPResolveToRecipient

The `SMPResolveToRecipient` function takes a pointer to a `PackedDSSpec` structure and returns a pointer to a linked list of mail addresses.

```
pascal OSErr SMPResolveToRecipient(PackedDSSpecPtr dsSpec,
                        SMPRecipientDescriptorPtr *recipientList,
                        AuthIdentity identity);
```

`dsSpec` A pointer to a `PackedDSSpec` structure containing the record ID and location information for a user record or group record.

`recipientList`
A pointer to a linked list of recipients for a letter. You can use this parameter as input to the `SMPSendLetter` function, or you can use the `recipient` field of the recipient descriptor as input to the `SMPAddAddress` function.

`identity` The authentication identity of the caller. The catalog uses this identity to determine whether the caller has the access privileges necessary to resolve specific mail addresses.

DESCRIPTION

When the user selects a record from one of the standard dialog boxes or from the Catalog-Browsing panel, you can use a pointer to the `PackedDSSpec` structure for that record as input to the `SMPResolveToRecipient` function.

If the `PackedDSSpec` structure holds a single address, the function returns a linked list with only one item. If the record is for a group address (that is, if the type of the record is `Group`) and the record is in a personal catalog, then the function resolves it into a linked list of all the members of the group, including all the members of any personal catalog groups in that group. The function performs this service for group addresses in personal catalogs because the recipient is unlikely to have the same information in his or her personal catalog. The function does not expand groups that are not in personal catalogs, because the recipient is assumed to have access to the catalog server to expand those groups.

You can use the linked list returned by the `SMPResolveToRecipient` function as input to the `SMPSendLetter` function.

SPECIAL CONSIDERATIONS

The `SMPResolveToRecipient` function allocates each recipient descriptor in the current heap. To dispose of a recipient descriptor you must first call the `DisposePtr` function to deallocate the `recipient` field in the recipient descriptor, and then call the `DisposePtr` function again to dispose of the recipient descriptor itself.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0006	$044C

RESULT CODES

noErr	0	No error
memFullErr	–108	Out of memory

SEE ALSO

The routines for displaying and obtaining information from standard catalog dialog boxes and the Catalog-Browsing panel are described in the chapter "Standard Catalog Package" in this book.

Recipient descriptors are described in "Recipient Descriptor" on page 3-25.

The SMPSendLetter function is described on page 3-37. The SMPAddAddress function is described on page 3-118.

Providing Mailers in Your Windows

The routines in this section add a mailer to a window and help you to make the mailer appear to be an integral part of your application. You must call the SMPInitMailer function before calling any of the other mailer functions.

The SMPNewMailer function (page 3-46) adds a new mailer to a window. The SMPGetDimensions function (page 3-48) lets you determine the size of a mailer so you can decide how to fit it in your window. You can add a new mailer to the mailer set of a received letter with the SMPMailerForward function (page 3-49) or create a new mailer for a reply letter with the SMPMailerReply function (page 3-51).

The SMPExpandOrContract function (page 3-56) lets you expand or contract a mailer from within your application, and the SMPMoveMailer function (page 3-61) lets you move a mailer within your window.

You can use the SMPGetTabInfo (page 3-53) and SMPBecomeTarget (page 3-54) functions to let the user navigate seamlessly among fields in the mailer and your application window using the Tab key.

You can call the SMPPrepareToClose function (page 3-59) to determine whether you can close a window that contains a mailer. You use the SMPDisposeMailer function (page 3-61) to remove a mailer from a window and release the memory used by the mailer.

SMPInitMailer

The `SMPInitMailer` function initializes the mailer routines of the Standard Mail Package.

```
pascal OSErr SMPInitMailer(long mailerVersion);
```

`mailerVersion`
The version number of the Standard Mail Package.

DESCRIPTION

You must call this function before the first time you call any other Standard Mail Package function that applies to mailers. If you do not call this function, other mailer functions return the result code `kSMPMailerNotInitialized` when you call them.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$1285

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	Out of memory

SMPNewMailer

The `SMPNewMailer` function allocates a new mailer for a window you specify.

```
pascal OSErr SMPNewMailer(WindowPtr window,
                          Point upperLeft,
                          Boolean canContract,
                          Boolean initiallyExpanded,
                          AuthIdentity identity,
                          const PrepareMailerForDrawingProcPtr
                                        prepareMailerForDrawingCB,
                          long clientData);
```

`window` The window in which you want the mailer to appear.

`upperLeft` The upper-left corner of the mailer, in the window's local coordinates. This position is normally (0, 0).

`canContract`
A Boolean value that specifies whether it should be possible to contract and expand the mailer. Specify `true` if you want the mailer to have this ability; this parameter should always be set to `true` unless the mailer is in its own, separate window.

`initiallyExpanded`
A Boolean value that specifies whether the mailer is to be expanded or contracted when initially displayed. Specify `true` if you want it to be expanded initially. The function ignores this parameter if the `canContract` parameter is set to `false`.

`identity` The authentication identity of the sender of the letter. Specify 0 to use the identity of the most recently authenticated user. The `SMPNewMailer` function uses the identity to fill in the From field in the mailer.

`prepareMailerForDrawingCB`
A pointer to your drawing-preparation function. If you change the clip region, coordinates, or other aspects of your window's graphics port, you must provide this function to restore the graphics port to a standard state so that the Standard Mail Package can draw a mailer in your window. The drawing-preparation function is described on page 3-122. Specify `nil` for this parameter if you are not providing a drawing-preparation function.

`clientData`
Reserved for your use. The `SMPNewMailer` function passes this value unaltered to your drawing callback routine.

DESCRIPTION

You should call the `SMPNewMailer` function whenever you want to have a mailer appear in a window; for example, when the user chooses the Add Mailer item from the Mail menu in your application. When you call this function, the Standard Mail Package adds a mailer to the window you specify. The next time the user chooses the Save or Save As commands, you should save the document in the letter file format rather than in your application's file format.

If you want the mailer to appear in a modeless, movable dialog box, or for some other reason do not want to provide the user with the ability to expand and contract the mailer, set the `canContract` parameter to `false`.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$000C	$125D

RESULT CODES

noErr	0	No error
memFullErr	–108	Out of memory
kSMPMailerNotInitialized	–1902	The mailer has not been initialized
kSMPMailerAlreadyInWindow	–1911	A mailer was previously allocated

SEE ALSO

Use the SMPBeginSave function (page 3-77) to save a document in the letter file format.

Use the SMPDisposeMailer function (page 3-61) to dispose of a mailer.

Use the SMPMailerForward function (page 3-49) to add a mailer to a letter that you want to forward.

Use the SMPMailerReply function (page 3-51) to add a reply mailer to a window.

SMPGetDimensions

The SMPGetDimensions function returns the standard dimensions of a mailer.

```
pascal OSErr SMPGetDimensions(short *width,
                              short *contractedHeight,
                              short *expandedHeight);
```

width A pointer to the minimum width, in QuickDraw coordinates, that bounds all of the fields in a mailer.

contractedHeight
A pointer to the height, in QuickDraw coordinates, of a mailer in the contracted state.

expandedHeight
A pointer to the height, in QuickDraw coordinates, of a mailer in the expanded state.

DESCRIPTION

The SMPGetDimensions function lets you determine the standard dimensions of a mailer from within your program so that your application will continue to work correctly if Apple ever changes the size of a mailer. When the user expands or contracts a mailer, it is up to you to update the content part of your document's window appropriately. You can use the heights returned by the SMPGetDimensions function to determine how large an area of your window is affected. You can use the width returned

by this function to help determine the size to make a window when the user clicks the zoom box. Clicking the zoom box should never make a window with a mailer in it smaller than the minimum size of the mailer.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0006	$125C

RESULT CODES

`noErr`	0	No error
`kSMPMailerNotInitialized`	–1902	The mailer has not been initialized

SEE ALSO

You use the `SMPMailerEvent` function (page 3-63) to determine when the user has contracted or expanded the mailer.

SMPMailerForward

The `SMPMailerForward` function creates a new mailer for a letter that is to be forwarded.

```
pascal OSErr SMPMailerForward(WindowPtr window,
                              AuthIdentity from);
```

`window` A pointer to the window containing the letter you want to forward.

`from` The authentication identity of the sender of the letter. Specify 0 to use the identity of the user whose mailbox contains the received letter. The `SMPMailerForward` function uses the identity to fill in the From field in the mailer.

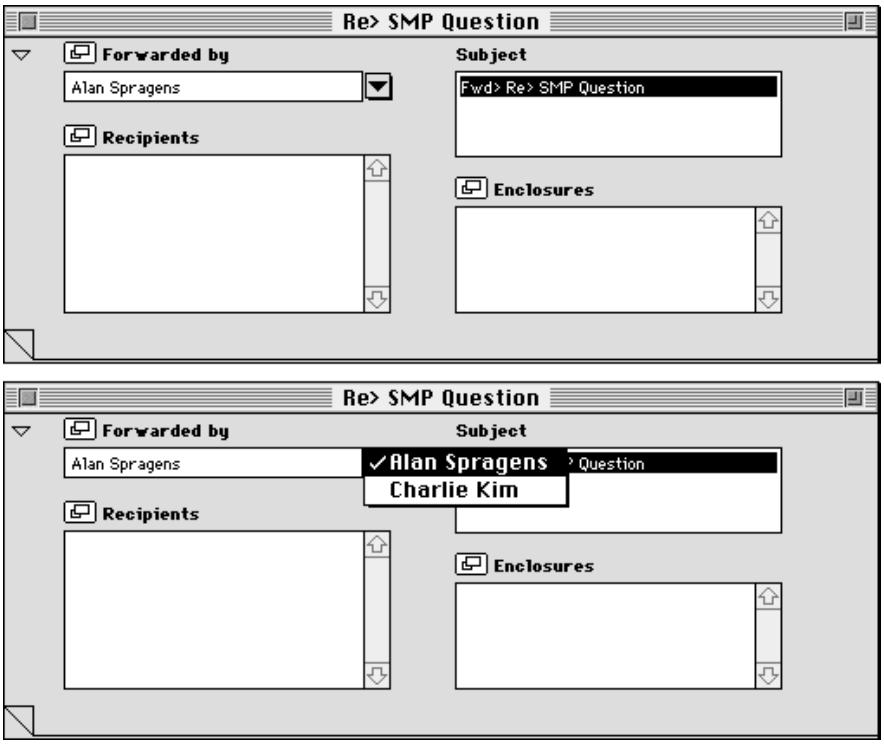
DESCRIPTION

When the user has received a letter and chooses the Forward item in the Mail menu, you call the `SMPMailerForward` function to add a new mailer to the mailer set. The function superimposes the new mailer on the existing mailers in the specified window and, if this is only the second mailer in the mailer set, adds a pop-up menu to the From field in the mailer. If there are already two or more mailers in the mailer set, the function

adds the new mailer to the existing pop-up menu. The user can use this menu to view any of the mailers in the mailer set. Figure 3-6 shows the top mailer and the pop-up menu for a letter that has been forwarded once.

You can call the `SMPMailerForward` function to add a new mailer to a mailer set only if the top mailer in the set is a received mailer. You can use the `hasBeenReceived` field of the `SMPMailerState` structure to get this information.

Figure 3-6 Mailer for a forwarded letter



SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$1261

RESULT CODES

noErr	0	No error
kOCEUnknownID	–1567	Authentication identity passed is not valid
kSMPNoMailerInWindow	–1909	No mailer is in the specified window

SEE ALSO

Use the SMPNewMailer function (page 3-46) to add a new mailer to a window that has no mailer.

Use the SMPMailerReply function (described next) to add a new mailer to a letter to which you want to reply.

Use the SMPGetMailerState function (page 3-69) to obtain an SMPMailerState structure. The SMPMailerState structure is described in "Mailer-State Structure" on page 3-30.

SMPMailerReply

The SMPMailerReply function helps you reply to a letter by adding a new mailer to a window you specify and addressing the reply mailer by copying information from the original mailer.

```
pascal OSErr SMPMailerReply(WindowPtr originalLetter,
                            WindowPtr newLetter,
                            Boolean replyToAll,
                            Point upperLeft,
                            Boolean canContract,
                            Boolean initiallyExpanded,
                            AuthIdentity identity,
                            const PrepareMailerForDrawingProcPtr
                                        prepareMailerForDrawingCB,
                            long clientData);
```

originalLetter
A pointer to the window containing the mailer for the original letter to which the user wishes to reply.

newLetter A pointer to the window that you are providing for the reply. The function adds a mailer to this window; the window must not already contain a mailer.

replyToAll
A Boolean value that indicates whether all the original "To" and "cc" recipients should be included as addressees for the reply.

upperLeft The upper-left corner of the mailer in the window's local coordinates. This position is normally (0, 0).

`canContract`
A Boolean value that specifies whether it should be possible to contract and expand the mailer. Specify `true` if you want the mailer to have this ability; this parameter should always be set to `true` unless the mailer is in its own, separate window.

`initiallyExpanded`
A Boolean value that specifies whether the mailer is to be expanded or contracted when initially displayed. Specify `true` if you want it to be expanded initially. The function ignores this parameter if the `canContract` parameter is set to `false`.

`identity` The authentication identity of the sender of the letter. Specify 0 to use the identity of the user whose mailbox contains the received letter. The `SMPMailerReply` function uses the identity to fill in the From field in the mailer.

`prepareMailerForDrawingCB`
A pointer to your drawing-preparation function. If you change the clip region, coordinates, or other aspects of your window's graphics port, you must provide this function to restore the graphics port to a standard state so that the Standard Mail Package can draw a mailer in your window. The drawing-preparation function is described on page 3-122. Specify `nil` for this field if you are not providing a drawing-preparation function.

`clientData`
Reserved for your use. The `SMPMailerReply` function passes this value unaltered to your callback routine.

DESCRIPTION

When the user chooses the Reply or Reply to All items in the Mail menu, you should create a new document window and call the `SMPMailerReply` function. This function places a mailer in the window, copies the subject from the original letter, places the string "Re>" in front of it, and places it in the Subject field of the new mailer. Then it copies the From address from the original letter and places it in the Recipients field of the reply mailer. If the user chose the Reply to All item, you should set the parameter `replyToAll` to `true`, and the `SMPMailerReply` function also copies all the recipients in the Recipients field— including the "cc" recipients—of the received mailer and places them in the corresponding fields of the reply mailer.

If the original letter has been forwarded, the `SMPMailerReply` function takes the subject and addresses from the original letter's most recent mailer; that is, from the mailer that was added the last time the letter was forwarded.

You should call the `SMPMailerReply` function only if the top mailer in the mailer set is a received mailer. You can use the `hasBeenReceived` field of the `SMPMailerState` structure to get this information.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$000F	$1262

RESULT CODES

noErr	0	No error
kOCEUnknownID	–1567	Authentication identity is not valid
kSMPNoMailerInWindow	–1909	No mailer is in the specified window
kSMPMailerAlreadyInWindow	–1911	Specified window already has a mailer

SEE ALSO

Use the SMPNewMailer function (page 3-46) to add a new mailer to a window that has no mailer.

Use the SMPMailerForward function (page 3-49) to add a mailer to a letter that you want to forward.

Use the SMPGetMailerState function (page 3-69) to obtain an SMPMailerState structure. The SMPMailerState structure is described in "Mailer-State Structure" on page 3-30.

SMPGetTabInfo

The SMPGetTabInfo function tells you which fields in the mailer are the first and last to be highlighted when the user presses the Tab key repeatedly to move from one field to another.

```
pascal OSErr SMPGetTabInfo(SMPMailerComponent *firstTab,
                           SMPMailerComponent *lastTab);
```

firstTab	The first field highlighted.
lastTab	The last field highlighted.

DESCRIPTION

When the user first clicks in a mailer, the Standard Mail Package makes one field the target for user actions and highlights that field. If the user presses the Tab key, the Standard Mail Package makes another field the target, and so on, eventually returning to the first field. You can intercept this sequence and make a field in your window active when the user presses the Tab key, returning to the mailer after you have given the user the opportunity to modify one or more fields in your window. The SMPGetTabInfo function tells you which field is the first one to be made a target by the Standard Mail Package and which is the last, so you know where to intercept the sequence and where

to return to it. This sequence is not dependent on the state of the mailer; you need call it only once.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$1274

RESULT CODES

`noErr`	0	No error
`kSMPMailerNotInitialized`	–1902	Mailer has not been initialized

SEE ALSO

The possible values for the `SMPMailerComponent` data type are shown on page 3-32.

You use the `SMPBecomeTarget` function (described next) to return the Tab sequence to the mailer.

You use the `SMPGetMailerState` function (page 3-69) to determine which field is currently the target.

SMPBecomeTarget

The `SMPBecomeTarget` function specifies whether your window or the mailer is the target of user action and, if the mailer is the target, specifies which field in the mailer is active.

```
pascal OSErr SMPBecomeTarget(WindowPtr window,
                             Boolean becomeTarget,
                             SMPMailerComponent whichField);
```

`window` The window containing the mailer.

`becomeTarget`
A Boolean value that specifies whether the mailer in this window should become the target of the user's actions. If this parameter is set to `true`, the mailer becomes the target. If it is set to `false`, the Standard Mail Package does not highlight any field, and the `SMPMailerEvent` function assumes that key-down events are intended for your application.

`whichField`

If the `becomeTarget` parameter is set to `true`, this parameter specifies which field should be active. If the `becomeTarget` parameter is set to `false`, the function ignores this field. Possible values for this field are shown on page 3-32.

DESCRIPTION

The user can use the Tab key to cycle through the fields in the mailer. Each time the user presses the Tab key, you receive a key-down event. In most cases, you would call the `SMPMailerEvent` function to handle the event. However, if you want one or more fields in your application's window to be included in the set of fields that the user can select with the Tab key, you must determine the nature of the key-down event yourself. If the user pressed the Tab key, you can call the `SMPGetMailerState` function (page 3-69) to determine which field in the mailer is currently the target. You can then check the results of the `SMPGetTabInfo` function to find out which field is the last one in the sequence. If the current field is the one returned in the `lastTab` parameter of the `SMPGetTabInfo` function, you can call the `SMPBecomeTarget` function with the `becomeTarget` parameter set to `false`. You can then activate and highlight whichever field in your window you wish.

When you have finished cycling through the fields in your window, call the `SMPBecomeTarget` function again, this time with the `becomeTarget` parameter set to `true` and the `whichField` parameter set to the value returned in the `firstTab` parameter of the `SMPGetTabInfo` function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0005	$1273

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in this window
`kSMPIllegalComponent`	–1918	Bad field name parameter
`kSMPMailerAlreadyNotTarget`	–1919	This mailer is not the target
`kSMPComponentIsAlreadyTarget`	–1920	The selected field is the target

SEE ALSO

The possible values for the `SMPMailerComponent` data type are shown on page 3-32.

You can call the `SMPGetMailerState` function (page 3-69) to determine which field in the mailer is currently the target.

You can call the `SMPGetTabInfo` function (page 3-53) to find out which fields are the first and last in the selection sequence.

You can call the `SMPMailerEvent` function (page 3-63) each time you receive an event. This function returns a value telling you how the Standard Mail Package handled the event and whether your application has to process it as well.

SMPExpandOrContract

The `SMPExpandOrContract` function expands or contracts a mailer.

```
pascal OSErr SMPExpandOrContract(WindowPtr window,
                                 Boolean expand);
```

`window` The window containing the mailer.

`expand` A Boolean value that specifies whether the mailer in this window should be expanded (`true`) or contracted (`false`).

DESCRIPTION

The user indicates a desire to expand or contract a mailer by clicking the triangle in the upper-left corner of the mailer (see Figure 3-2 and Figure 3-3 on page 3-5). If the user wants to expand the mailer, the `SMPMailerEvent` function returns the flag `kExpanded`. You must update your window to make room for the expanded mailer and then call the `SMPExpandOrContract` function to expand the mailer. (When the user contracts the mailer, by contrast, you have to update the content portion of your window but do not have to call the `SMPExpandOrContract` function.)

The `SMPExpandOrContract` function also lets you expand or contract the mailer entirely from within your application, for example, to implement an Expand or Contract menu command.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0003	$1272

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPMailerCannotExpandOrContract`	–1916	Mailer created with `canContract` parameter set to `false`
`kSMPMailerAlreadyExpandedOrContracted`	–1917	Mailer is already in requested state

SEE ALSO

You set the initial state of the mailer (expanded or contracted) with the `SMPNewMailer` function (page 3-46), the `SMPMailerReply` function (page 3-51), or the `SMPOpenLetter` function (page 3-94).

You can call the `SMPGetMailerState` function (page 3-69) to determine whether the mailer is currently expanded or contracted.

You call the `SMPGetDimensions` function (page 3-48) to determine the size of an expanded or contracted mailer.

SMPMoveMailer

The `SMPMoveMailer` function moves a mailer within your window.

```
pascal OSErr SMPMoveMailer(WindowPtr window,
                           short dh,
                           short dv);
```

`window`	The window containing the mailer you want to move.
`dh`	The horizontal distance, in QuickDraw coordinates, by which you want to move the mailer. Use a positive number to move the mailer to the right and a negative number to move the mailer to the left.
`dv`	The vertical distance, in QuickDraw coordinates, by which you want to move the mailer. Use a positive number to move the mailer down and a negative number to move the mailer up.

DESCRIPTION

You set the initial location of a mailer in your window when you call the `SMPNewMailer` function or the `SMPMailerReply` function. You can use the `SMPMoveMailer` function to move a mailer if, for example, you need to make space for a tool palette at the top or left edge of your window.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$126A

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

You set the initial location of the mailer with the `SMPNewMailer` function (page 3-46), the `SMPMailerReply` function (page 3-51), or the `SMPOpenLetter` function (page 3-94).

SMPTagDialog

The `SMPTagDialog` function displays a dialog box that allows a user to add a tag to a letter that was opened from the mailbox.

```
pascal OSErr SMPTagDialog(WindowPtr window,
                          RString32 *theTag);
```

`window` The window containing the mailer.

`theTag` A pointer to the tag to be associated with the letter. If you specify a tag when you call the function, it is displayed as the default value in the dialog box. The function uses this parameter to return the tag specified by the user.

DESCRIPTION

The PowerTalk mailbox allows the user to sort and display letters according to tags that the user has specified for each letter. Your application can provide a Tag item in the Mail menu. If the user chooses this item, you should call the `SMPTagDialog` function to let the user specify the tag. You should call the `SMPGetMailerState` function to determine whether to enable the Tag command.

The AOCE software stores the tag with the letter and displays it for the user in the In and Out Trays. It is not necessary for you to specify this tag in the close-options structure when you call the `SMPCloseOptionsDialog` or `SMPDisposeMailer` functions. When you save the letter to disk, the letter becomes an HFS object and no longer has a tag.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$128B

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in a parameter value
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

Call the `SMPGetMailerState` function (page 3-69) to determine whether to enable the Tag command.

SMPPrepareToClose

The `SMPPrepareToClose` function tells you whether a mailer can be closed.

```
pascal OSErr SMPPrepareToClose(WindowPtr window);
```

`window` The window containing the mailer that you would like to close.

DESCRIPTION

In certain circumstances—for instance, when an enclosure is open—you can't dispose of a mailer. The `SMPPrepareToClose` function returns an error when you can't dispose of a mailer, so you can display a dialog box informing the user of the situation rather than closing the window containing the mailer.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$1287

RESULT CODES

noErr	0	No error
kSMPCopyInProgress	–1901	Finder is copying an enclosure
kSMPHasOpenAttachments	–1906	One or more enclosures are open
kSMPNoMailerInWindow	–1909	No mailer is in specified window

SEE ALSO

You should call the SMPPrepareToClose function before closing a window containing a mailer or attempting to call the SMPDisposeMailer function (page 3-61).

SMPCloseOptionsDialog

The SMPCloseOptionsDialog function displays a dialog box that allows a user to delete letters or add tags to letters that were opened from the mailbox.

```
pascal OSErr SMPCloseOptionsDialog(WindowPtr window,
                                   SMPCloseOptionsPtr closeOptions);
```

window The window containing the mailer.

closeOptions
A pointer to a close-options structure that specifies the initial settings to be displayed in the dialog box. After the function call returns, this structure contains the new settings entered by the user in the close-options dialog box.

DESCRIPTION

Your application should provide a user preference option that specifies whether attempting to close a letter should cause the close-options dialog box to appear. If the user elects to see the dialog box, you should call the SMPCloseOptionsDialog function whenever a user closes a window containing a mailer and before you call the SMPDisposeMailer function. If the user opened the letter from the mailbox, the SMPCloseOptionsDialog function displays the close-options dialog box; otherwise, the function does nothing.

You can use the closeOptions parameter to provide default settings for the dialog box. If you provide a Close and Delete item in the File menu and the user chooses this item, you should specify true for the moveToTrash field of the structure pointed to by the closeOptions parameter.

You can use the tag field of the structure pointed to by the closeOptions parameter to specify a default tag for the letter. If the letter already has a tag value, either because the user added it the last time the letter was closed or because you called the SMPTagDialog function, the Standard Mail Package puts that tag in the tag field of the dialog box. It is not necessary for you to specify this tag in the close-options structure when you call the SMPCloseOptionsDialog function.

The Standard Mail Package stores the options the user selects and executes them when you call the `SMPDisposeMailer` function (if the `closeOptions` parameter in the `SMPDisposeMailer` function is not set to `nil`).

SPECIAL CONSIDERATIONS

If you specify `true` for both the `moveToTrash` field and the `addTag` field of the close-options structure, the `SMPCloseOptionsDialog` function returns the `paramErr` result code.

If you specify `true` for the `addTag` field of the close-options structure, you must also specify a valid tag for the letter. If you specify `true` for the `addTag` field and you specify a zero-length string for the `tag` field, the function returns the `paramErr` result code.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$1288

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in a parameter value
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

The close-options structure is described in “Close-Options Structure” on page 3-29.

You can call the `SMPTagDialog` function (page 3-58) to display a dialog box that allows the user to add a tag to a letter in the In Tray.

Call the `SMPDisposeMailer` function (described next) to execute the close options.

SMPDisposeMailer

The `SMPDisposeMailer` function deallocates the mailer set in the specified window and erases the mailer set.

```
pascal OSErr SMPDisposeMailer(WindowPtr window,
                              SMPCloseOptionsPtr closeOptions);
```

`window` The window containing the mailer set you want to deallocate.

`closeOptions`
A pointer to a close-options structure specifying actions the Standard Mail Package should take in addition to disposing of the mailer set. If you specify nil for this parameter, the function disposes of the mailer set without taking any other action.

DESCRIPTION

You should call the `SMPDisposeMailer` function when the user chooses the Remove Mailer item from a menu or when you close a window that contains a mailer. This function removes the mailer set from the window you specify and deallocates all the data structures associated with that mailer set. If the user removes the mailer from the window, the next time the user chooses the Save or Save As commands, you should save the document in your application's file format rather than the letter file format.

You use the `closeOptions` parameter to specify close options. For example, you can provide a Close and Delete item in the File menu. If the user chooses this item, you should specify `true` for the `moveToTrash` field of the structure pointed to by the `closeOptions` parameter.

Your application may provide a user preference option that specifies whether attempting to close a letter should cause the close-options dialog box to appear. If the user elects to see the dialog box, you should call the `SMPCloseOptionsDialog` function whenever the user closes a window containing a mailer. Then use the pointer to the close-options structure that you provided to the `SMPCloseOptionsDialog` function as the value of the `closeOptions` parameter of the `SMPDisposeMailer` function.

Before you close a window that contains a mailer, call the `SMPPrepareToClose` function to make sure that you can dispose of the mailer.

SPECIAL CONSIDERATIONS

If you specify `true` for both the `moveToTrash` field and the `addTag` field of the close-options structure, the `SMPDisposeMailer` function returns the `paramErr` result code.

If you specify `true` for the `addTag` field of the close-options structure, you must also specify a valid tag for the letter. If you specify `true` for the `addTag` field and you specify a zero-length string for the `tag` field, the function returns the `paramErr` result code.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$125E

RESULT CODES

noErr	0	No error
paramErr	–50	Error in a parameter value
kSMPHasOpenAttachments	–1906	One or more enclosures are open
kSMPNoMailerInWindow	–1909	No mailer is in specified window

SEE ALSO

The SMPPrepareToClose function (page 3-59) tells you whether it's possible to dispose of a mailer.

The close-options structure is described in "Close-Options Structure" on page 3-29.

The SMPCloseOptionsDialog function (page 3-60) displays a dialog box that lets the user select close options for a letter that was opened from the mailbox.

Handling Events in Mailers

Whenever you receive an event for a window that contains a mailer, you can pass that event directly to the SMPMailerEvent function (described next). The Standard Mail Package handles the event if it applied to the mailer and returns a value that tells you what action it took and whether you have to take any further action. You can also use the SMPMailerEditCommand function (page 3-67) to handle events related to standard items in the Edit menu.

When the user is working in the mailer, you must enable and disable items in the Edit menu as appropriate. The SMPGetMailerState function (page 3-69) lets you determine which items should be enabled or disabled. You must ensure that the Undo command works consistently whether the user is working in the mailer or in your application. You use the SMPGetMailerState function to determine when to clear your application's undo buffer, and you use the SMPClearUndo function (page 3-70) to tell the mailer when to clear its Undo buffer.

Finally, you can use the SMPDrawMailer function (page 3-72) to redraw the mailer if you want to handle update events yourself or if you need to redraw the mailer for some other reason.

SMPMailerEvent

The SMPMailerEvent function processes events that you pass to it, gives you information about how the Standard Mail Package responded to the event, and informs you of further action that you must take.

```
pascal OSErr SMPMailerEvent(const EventRecord *event,
                            SMPMailerResult *whatHappened,
                            const FrontWindowProcPtr frontWindowCB,
                            long clientData);
```

`event` A pointer to the event record of an event returned to your application by the `WaitNextEvent` function.

`whatHappened`
A pointer to a set of flags informing you what action the `SMPMailerEvent` function took.

`frontWindowCB`
A pointer to your front-window routine. This routine, described on page 3-124, returns a pointer to the window that your application wants the Standard Mail Package to consider as the front window. Specify `nil` for this field if you do not want to provide a front-window routine. If you do not provide a front-window routine, the Standard Mail Package uses the Window Manager's `FrontWindow` routine.

`clientData`
Reserved for your use. The `SMPMailerEvent` function passes this value unaltered to your callback routine.

DESCRIPTION

Each time your application calls the `WaitNextEvent` function, it can pass the event record immediately to the `SMPMailerEvent` function. The `SMPMailerEvent` function determines whether the Standard Mail Package should handle the event, your application should handle the event, or action is required by both the Standard Mail Package and your application. If the `SMPMailerEvent` function has to take any further action, it does so before returning control to your application. In any case, the `whatHappened` parameter returns a set of flags that tell you what action, if any, the function took, and whether your application must handle the event.

If the event record does not include a window pointer, the `SMPMailerEvent` function uses your front-window callback routine to determine to which window the event applies. If you do not provide a front-window callback routine, the `SMPMailerEvent` function uses the Window Manager's `FrontWindow` routine.

If you decide instead to check the event record first and pass to the `SMPMailerEvent` function only events that the Standard Mail Package must handle, call the `SMPBecomeTarget` function when the mailer is no longer the target (for example, when the user clicks in the content region of the window). In that case, you must still pass null events to the `SMPMailerEvent` function frequently so that the Standard Mail Package can control the appearance of the cursor, implement Balloon Help, and pass null events to the Catalog-Browsing panel and Find-Record panel.

IMPORTANT

To use the Standard Mail Package, your application must be aware of high-level events. You must pass all high-level events (including Apple events) to the `SMPMailerEvent` function before calling the `AEProcessAppleEvent` or `AcceptHighLevelEvent` routines. If you do not do so, some Standard Mail Package features, such as enclosing files and folders, may not work correctly. s

The flags returned by the whatHappened parameter are as follows:

```
enum {
   kSMPAppMustHandleEventBit,
   kSMPAppShouldIgnoreEventBit,
   kSMPContractedBit,
   kSMPExpandedBit,
   kSMPMailerBecomesTargetBit,
   kSMPAppBecomesTargetBit,
   kSMPCursorOverMailerBit,
   kSMPCreateCopyWindowBit,
   kSMPDisposeCopyWindowBit
};
```

You can use the following masks to test for these bits:

```
enum {
   kSMPAppMustHandleEventMask    = 1<<kSMPAppMustHandleEventBit,
   kSMPAppShouldIgnoreEventMask  = 1<<kSMPAppShouldIgnoreEventBit,
   kSMPContractedMask            = 1<<kSMPContractedBit,
   kSMPExpandedMask              = 1<<kSMPExpandedBit,
   kSMPMailerBecomesTargetMask   = 1<<kSMPMailerBecomesTargetBit,
   kSMPAppBecomesTargetMask      = 1<<kSMPAppBecomesTargetBit,
   kSMPCursorOverMailerMask      = 1<<kSMPCursorOverMailerBit,
   kSMPCreateCopyWindowMask      = 1<<kSMPCreateCopyWindowBit,
   kSMPDisposeCopyWindowMask     = 1<<kSMPDisposeCopyWindowBit
   };

typedef unsigned long SMPMailerResult;
```

Bit descriptions

kSMPAppMustHandleEventBit
: The application must process the event. The event was either an event the Standard Mail Package couldn't process, or it was one that both the application and the Standard Mail Package must process (such as activate and update events). The function always sets either this flag or the kSMPAppShouldIgnoreEventBit flag.

kSMPAppShouldIgnoreEventBit
: The application should ignore the event. It was handled by the Standard Mail Package. The function always sets either this flag or the kSMPAppMustHandleEventBit flag.

`kSMPContractedBit`
: The user clicked the triangle at the left edge of the mailer (see Figure 3-3 on page 3-5), switching the mailer to the contracted state. Because your application does not have to read the event record, the function also sets the `kSMPAppShouldIgnoreEventBit` flag. However, you must update the content portion of your application's frontmost window when you receive this flag.

`kSMPExpandedBit`
: The user clicked the triangle at the left edge of the mailer (see Figure 3-2 on page 3-5), indicating a desire to switch the mailer to the expanded state. Because your application does not have to read the event record, the function also sets the `kSMPAppShouldIgnoreEventBit` flag. However, you must update the content portion of your application's frontmost window and call the `SMPExpandOrContract` function (page 3-56) to finish expanding the mailer when you receive this flag.

`kSMPMailerBecomesTargetBit`
: The user had been working in the application's part of the window but has now clicked in the mailer or contracted the mailer. Because your application does not have to read the event record, the function also sets the `kSMPAppShouldIgnoreEventBit` flag. However, you might want to take other action, such as removing highlighting from the content portion of the window or stopping an insertion-point caret from blinking.

`kSMPAppBecomesTargetBit`
: The user had been working in the mailer but has now clicked in the application's part of the window. Because you must handle this event, the function also sets the `kSMPAppMustHandleEventBit` flag.

`kSMPCursorOverMailerBit`
: When this flag is set, the cursor is in the mailer in the frontmost window, so the Standard Mail Package is controlling Balloon Help and the appearance of the cursor. When this flag is cleared, the cursor is not in the mailer, so you must control the appearance of the cursor. The function also sets the `kSMPAppMustHandleEventBit` flag when it sets or clears the `kSMPCursorOverMailerBit` flag. Because the Standard Mail Package can detect the position of the cursor only when you give it some processing time, the function sets or clears this flag only when you pass it a null event.

`kSMPCreateCopyWindowBit`
: The Standard Mail Package is using the Finder to copy files and has displayed a modal dialog box showing the status of the copy operation. You should continue to send events to the `SMPMailerEvent` function.

`kSMPDisposeCopyWindowBit`
: The Standard Mail Package has removed the copy status dialog box. Your application should resume normal operation.

SPECIAL CONSIDERATIONS

The Standard Mail Package reserves all high-level events of class `'cwin'` for its own use. Do not install an event handler for events of this class. (There is no problem if you have installed an Apple event handler with class and ID of `typeWildCard`, because the Standard Mail Package removes that handler before calling the `AEProcessAppleEvent` routine and reinstalls it afterward.)

The `SMPMailerEvent` function may move or purge memory; you should not call this function at interrupt time.

The `SMPMailerEvent` function preserves your application's A5 world when it calls your front-window routines. Therefore, you have access to your application's global variables from these routines.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0008	$125F

RESULT CODES

`noErr`	0	No error

SEE ALSO

The front-window callback function is described on page 3-124.

If you check the event record first and pass to the `SMPMailerEvent` function only events that the Standard Mail Package must handle, call the `SMPBecomeTarget` function (page 3-54) when the mailer is no longer the target.

Call the `SMPGetMailerState` function (page 3-69) to determine the current state of the mailer.

SMPMailerEditCommand

The `SMPMailerEditCommand` function handles Edit menu commands when they apply to fields in the mailer.

```
pascal OSErr SMPMailerEditCommand(WindowPtr window,
                        SMPEditCommand command,
                        SMPMailerResult *whatHappened);
```

`window` A pointer to the window containing the mailer.

`command` The Edit menu command that the user chose.

`whatHappened`
: A pointer to a set of flags that indicate what action the function took and whether you must take any action. This function sets either the `kSMPAppMustHandleEventBit` or the `kSMPAppShouldIgnoreEventBit` flag.

DESCRIPTION

When the user chooses one of the standard Edit menu commands (Undo, Cut, Copy, Paste, Clear, or Select All), you can call the `SMPMailerEditCommand` function immediately. If the user has selected something in the mailer, the `SMPMailerEditCommand` function handles the requested action, sets the `kSMPAppShouldIgnoreEventBit` flag in the `whatHappened` parameter, and returns. If the user has not selected anything in the mailer, the function sets the `kSMPAppMustHandleEventBit` flag and returns to you immediately. Use the `SMPGetMailerState` function to determine which of the Edit menu commands to enable.

The possible values for the `command` parameter are as follows:

```
enum {
   kSMPUndoCommand,
   kSMPCutCommand,
   kSMPCopyCommand,
   kSMPPasteCommand,
   kSMPClearCommand,
   kSMPSelectAllCommand
};

typedef unsigned short SMPEditCommand;
```

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0005	$1260

RESULT CODES

noErr	0	No error
kSMPNoMailerInWindow	–1909	No mailer is in specified window
kSMPIllegalComponent	–1918	Bad command parameter value

SEE ALSO

The flag field for the whatHappened parameter is completely defined on page 3-65.

You can use the SMPGetMailerState function (page 3-69) to determine whether the user is working in the mailer and, if so, which of the Edit menu commands to enable.

You can call the SMPMailerEvent function (page 3-63) each time you receive an event. This function returns a value telling you how the Standard Mail Package handled the event and whether your application has to process it as well. The SMPMailerEvent function returns the value kSMPAppMustHandleEventBit when the event is an Edit menu command.

SMPGetMailerState

The SMPGetMailerState function returns the state of the specified mailer.

```
pascal OSErr SMPGetMailerState(windowPtr window,
                        SMPMailerState *itsState);
```

window	The window containing the mailer whose state you want to know.
itsState	A pointer to a structure containing the state of the mailer. The SMPMailerState data type is defined and all of its fields are described in "Mailer-State Structure" on page 3-30.

DESCRIPTION

The SMPGetMailerState function lets you determine whether the user is working in the mailer, and if so, which Edit menu and Mail menu commands you should enable. For example, if both the isTarget and canCut fields are set to true, then you should enable the Cut item in the Edit menu. This function also returns a value that helps you determine whether to clear your application's undo buffer. You should call this function when you need to display the Edit menu (that is, before calling the MenuSelect function) or the Mail menu or when the user presses a keyboard equivalent for an Edit menu command or Mail menu command.

This function also returns other information about the mailer, such as its current state (contracted or expanded), its location in the window, and the number of mailers in the mailer set.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$1263

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

The `SMPMailerState` data type is defined and all of its fields are described in "Mailer-State Structure" on page 3-30.

You can call the `SMPMailerEvent` function (page 3-63) each time you receive an event. This function returns a value telling you how the Standard Mail Package handled the event and whether your application has to process it as well.

SMPClearUndo

The `SMPClearUndo` function tells the Standard Mail Package to clear its undo buffer.

```
pascal OSErr SMPClearUndo(WindowPtr window);
```

`window` A pointer to the window containing the mailer.

DESCRIPTION

The *Macintosh Human Interface Guidelines* call for an Undo item in the Edit menu and specify that only the latest action can be undone. Furthermore, certain actions that cannot be undone should cause you to disable the Undo item and some should not; for example, you should disable the Undo item after the user saves a file but not after the user scrolls through the window. Even though the Standard Mail Package maintains its own undo buffer, you are responsible for enabling and disabling the Undo item in the Edit menu whether the user is working in the content portion of your window or in the mailer. The `SMPClearUndo` function lets you coordinate the Standard Mail Package's undo facility with that of your application so that it appears to the user that there is only one undo buffer for the entire window.

You should call the `SMPGetMailerState` function when you need to display the Edit menu (that is, before calling the `MenuSelect` routine) or when the user presses a keyboard equivalent for an Edit menu command. If the `SMPGetMailerState` function indicates that the user is working in the content portion of the window, you must determine whether the user can undo the action, and you must enable or disable the Undo item in the Edit menu accordingly. If the action can be undone or if it causes you to disable the Undo item, you must call the `SMPClearUndo` function to tell the Standard Mail Package to clear its undo buffer. If you fail to do so, the Standard Mail Package will not update the mailer state correctly.

Conversely, if the user's last action was in the mailer, you must call the `SMPGetMailerState` function to find out whether to enable or disable the Undo item in the Edit menu and whether to clear your application's undo buffer.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$1275

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

You can call the `SMPMailerEvent` function (page 3-63) each time you receive an event. This function returns a value telling you how the Standard Mail Package handled the event and whether your application has to process it as well.

The `SMPGetMailerState` function (page 3-69) returns a mailer-state structure; see "Mailer-State Structure" on page 3-30. The `undoState` field of the mailer-state structure, described on page 3-33, returns a value that tells you whether to clear your application's undo buffer and whether to disable the Undo item in the Edit menu.

You can use the `SMPMailerEditCommand` function (page 3-67) to handle edit commands when they apply to the mailer.

SMPDrawMailer

The `SMPDrawMailer` function redraws the mailer in the window you specify.

```
pascal OSErr SMPDrawMailer(WindowPtr window);
```

`window` A pointer to the window containing the mailer you want to draw.

DESCRIPTION

You use the `SMPDrawMailer` function to redraw a mailer when you need to do so but have not received an update event, or if you want to handle an update event yourself rather than passing it to the `SMPMailerEvent` function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$1269

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

You can have the Standard Mail Package draw the mailer by passing update events to the `SMPMailerEvent` function (page 3-63).

Sending and Saving Mail

The `SMPBeginSend` function (page 3-81) starts the process of creating a letter that is to be mailed. Immediately before you call the `SMPBeginSend` function, you should call the `SMPSendOptionsDialog` function (page 3-73) to let the user set send options.

The `SMPSendOptionsDialog` function returns the user's choice: whether to send the letter as an image, as standard interchange format, in one of your application's native formats, or in some combination of these three options. To send the letter as an image, you must call the `SMPImage` function (page 3-88) to put an image block in the letter. Call the `SMPAddContent` function (page 3-85) to add a standard interchange format block to the letter and the `SMPAddMainEnclosure` function (page 3-90) to add a document in

one of its native formats to the letter. You can also call the SMPAddBlock function (page 3-91) to add one or more blocks of your own design to the letter.

You can enclose files or folders in a letter by calling the SMPAddAttachment function (page 3-119) or the SMPAttachDialog function (page 3-119).

When you are ready to send the letter, call the SMPEndSend function (page 3-84).

The process of saving a letter to disk is similar to the process of sending one. First you call the SMPBeginSave function (page 3-77) to create the letter. Then you can use the SMPAddContent, SMPAddMainEnclosure, and SMPAddBlock functions to add content to the letter. To save the letter, call the SMPEndSave function (page 3-80). You can use the SMPOpenLetter function (page 3-94) to open a letter on disk for reading.

SMPSendOptionsDialog

The SMPSendOptionsDialog function displays the send-options dialog box and returns the user's selections.

```
pascal OSErr SMPSendOptionsDialog(WindowPtr window,
                                  Str255 documentName,
                                  StringPtr nativeFormatNames[],
                                  unsigned short nameCount,
                                  SMPSendFormatMask canSend,
                                  SMPSendFormat *currentFormat,
                                  SendOptionsFilterProc filterProc,
                                  long clientData,
                                  SMPSendFormat *shouldSend,
                                  SMPSendOptionsPtr sendOptions);
```

`window` A pointer to the window containing the mailer.

`documentName`
The name of the document. This name is displayed at the top of the send-options dialog box.

`nativeFormatNames`
An array of string pointers containing the names of the "native" formats your application can use for the letter. These names should be the same as the formats listed in the dialog box you display when the user chooses Save As from the File menu. If your application can write data in only one format, use the name of the application. The Standard Mail Package displays the names you list here in a pop-up menu in the send-options dialog box.

`nameCount` The number of string pointers in the `nativeFormatNames` parameter.

`canSend` A set of flags indicating which types of format your application can send for this letter. You can use any combination of the mask values `kSMPNativeMask`, `kSMPImageMask`, and `kSMPStandardInterchangeMask`.

`currentFormat`
A pointer to a send-format structure. If the user opened this letter from the mail box or from disk, you should use this structure to indicate which formats the letter currently contains. If this is a new letter, you should indicate which formats you prefer to use for this letter. Do not include any format you did not include in the canSend parameter.

`filterProc`
A pointer to a routine you can provide to add additional items to the send-options dialog box. This routine is described on page 3-125.

`clientData`
A constant reserved for your use. The Standard Mail Package passes this value to the routine you provide in the `filterProc` parameter.

`shouldSend`
A pointer to a send-format structure, allocated by your application, in which the `SMPSendOptionsDialog` function returns the formats the user has selected for sending the letter.

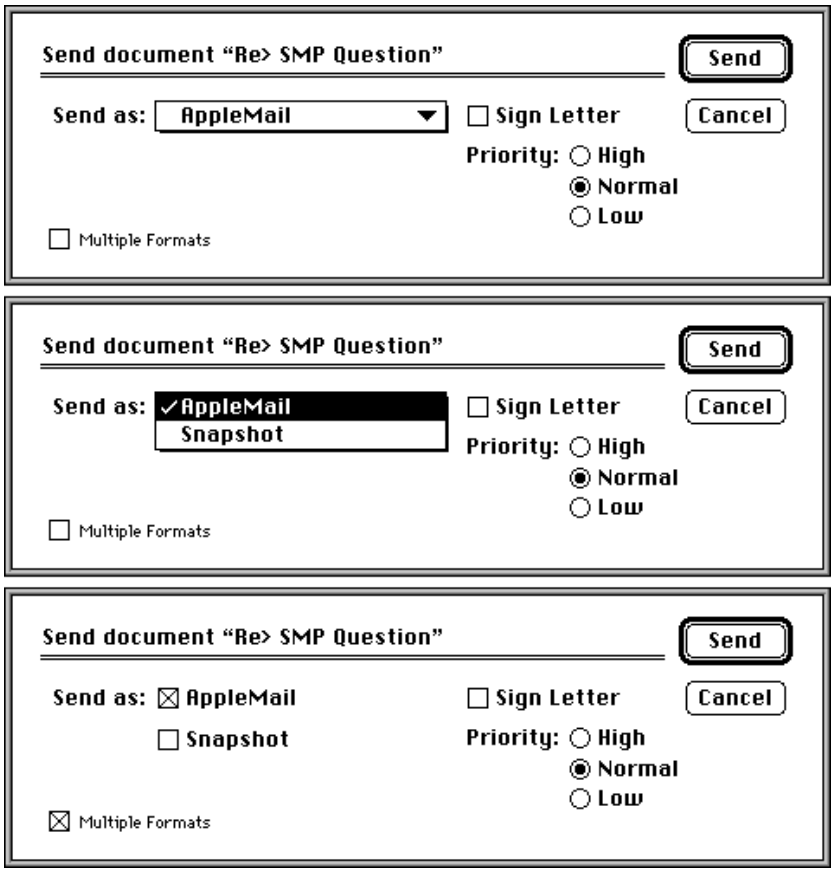
`sendOptions`
A pointer to a send-options structure. Pass this pointer to the `SMPBeginSend` function when you are ready to send the letter. The function only returns information in this structure; it ignores any values in this structure that are set at the time you call the function.

DESCRIPTION

The send-options dialog box lets the user specify whether letters should be signed and whether documents should be sent as application documents, images, or both.

Figure 3-7 shows a send-options dialog box.

Figure 3-7 Send-options dialog box



To make it possible for any user to read letters sent by your application, even if the recipient doesn't have your application, you should be able to add either a standard interchange format version of your document, an image version, or both to the letter. If the standard interchange format does not completely describe your application's documents, you can also send a document in one of its native formats either as a main enclosure to the letter or as a block or blocks that you add to the letter. You use the `canSend` parameter to specify which formats you are prepared to add to a letter.

You should call the `SMPSendOptionsDialog` function before you use the `SMPBeginSend` function to initiate the process of sending a letter. The `SMPBeginSend` function returns a send-format structure that indicates whether the user wants to send an image, standard interchange format, one of the document formats supported by your application, or some combination of the three. If the reader wants to send an image, you must call the `SMPImage` function so that the Standard Mail Package can provide the image. If the reader wants to send standard interchange format, call the `SMPAddContent` function. If the user wants to send the letter as a document, the send-format structure also indicates which of your application's document formats to use.

To add your own items to the send-options dialog box, provide a send-options filter routine.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

The `SMPSendOptionsDialog` function only returns the user's selections. You cannot use this function to set default values for the fields in the send-options dialog box.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0013	$1388

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in a parameter value
`userCanceledErr`	–128	User clicked Cancel button
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPIllegalSendFormats`	–1923	Format not in `canSend` parameter

SEE ALSO

See the descriptions of the `SMPBeginSend` (page 3-81) and `SMPEndSend` (page 3-84) functions for more information about sending mail.

The send-format structure is defined in "Send-Format Structure" on page 3-34. The send-options structure is defined in "Send-Options Structure" on page 3-34.

You can specify a routine to add items to the send-options dialog box. See the description of the `MySendOptionsFilterProc` routine on page 3-125 for more information.

Call the `SMPAddMainEnclosure` function (page 3-90) to add a main enclosure to a letter. Call the `SMPAddContent` function (page 3-85) to add standard interchange format content to a letter. Call the `SMPImage` function (page 3-88) to add an image to the letter.

SMPContentChanged

The `SMPContentChanged` function informs the Standard Mail Package that the content of the letter has changed.

```
pascal OSErr SMPContentChanged(WindowPtr window);
```

`window` A pointer to the window containing the mailer.

DESCRIPTION

You must call the `SMPContentChanged` function to inform the Standard Mail Package when the user changes the content of a letter. The Standard Mail Package can then indicate to the user that the signature is not valid. The Standard Mail Package also needs this information to determine whether it can save or send a forwarded letter without requiring you to rebuild the content of the letter.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$126F

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

To forward a letter, see the `SMPMailerForward` function (page 3-49).

To save a letter, see the `SMPBeginSave` function (page 3-77). To send a letter, see the `SMPBeginSend` function (page 3-81).

Before you allow the user to change the content of a letter, call the `SMPPrepareToChange` function (page 3-83).

SMPBeginSave

You must call the `SMPBeginSave` function before you save a letter.

```
pascal OSErr SMPBeginSave(WindowPtr window,
                          const FSSpec *diskLetter,
                          OSType creator,
                          OSType filetype,
                          SMPSaveType saveType,
                          Boolean *mustAddContent);
```

`window` The window containing the letter to be saved.

`diskLetter`
: A pointer to a file system specification structure indicating the name and location you want to use for the file.

`creator`
: The creator for the file, which also becomes the letter's creator. Use the same creator that you use for all of your application's documents.

`filetype`
: The file type, which also becomes the letter type. Letters containing only AOCE standard content should be of type `'lttr'`.

`saveType`
: The type of save: `kSMPSave`, `kSMPSaveAs`, or `kSMPSaveACopy`.

`mustAddContent`
: A pointer to a Boolean value returned by the function that tells you whether you have to add any blocks or enclosures to the letter. If this parameter is set to `false`, call the `SMPEndSave` function immediately without adding blocks or enclosures to the letter.

DESCRIPTION

When you save a document to which you have added a mailer, you must save it in letter file format rather than the document format normally used by your application. A letter consists of a header, data blocks, and enclosures. Every block has a block creator and type, and every letter has a letter creator and type. When you save the letter, the Standard Mail Package assigns the file the same creator and type as the letter. Your application should provide icon resources and a file reference resource for your application's letters so that users can distinguish them from standard documents.

To begin the process of saving a letter, you call the `SMPBeginSave` function. This function prepares the letter file into which you can save your document. You can create a new file format for your application's documents that takes advantage of the block structure of a letter file, or you can save your document in one of your application's native formats to a temporary file on disk and then add that file as the main enclosure to the letter. The letter is not actually saved to disk until you call the `SMPEndSave` function.

If neither your application nor the user has changed the content of a received letter, the `SMPBeginSave` function returns a value of `false` for the `mustAddContent` parameter. In that case you can call the `SMPEndSave` function immediately to save the letter. If you have changed the content of the letter, however, the `mustAddContent` parameter returns `true` and you must build the letter (adding the appropriate combination of blocks, main enclosure, standard interchange format block, and image block) just as if it were a new letter. The Standard Mail Package handles enclosures added by the user. Note that, if you make changes to the enclosed original letter, you invalidate any digital signature.

IMPORTANT

Be sure to call the `SMPContentChanged` function whenever the user changes the content of a letter. If you do not call the `SMPContentChanged` function, then the `SMPBeginSave` function doesn't know that the letter has been changed and won't return `true` as the value of `mustAddContent`. s

Note
The SMPBeginSave and SMPEndSave function pair perform a "safe save"; that is, they save the document into a temporary file and then, if the document has been saved before, replace the original file with the temporary file. u

When you call the SMPBeginSave function you must specify the type of save, as follows:

```
enum {kSMPSave, kSMPSaveAs, kSMPSaveACopy};

typedef unsigned short SMPSaveType;
```

Constant descriptions

kSMPSave	Save an existing file, overwriting the older version, and keeping the file open.
kSMPSaveAs	Save the file with a new name, close the original file (if any) without changing it, and open the new file.
kSMPSaveACopy	Save a copy of the file with a new name, leaving the original file open.

SPECIAL CONSIDERATIONS

Any time the user changes the content of a letter, you must call the SMPContentChanged function immediately. The SMPBeginSave function needs this information to operate correctly.

The SMPBeginSave function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$000B	$1266

RESULT CODES

noErr	0	No error
dskFulErr	–34	Disk is full
fnfErr	–43	File not found
fBsyErr	–47	File is busy
kSMPNoMailerInWindow	–1909	No mailer is in specified window

SEE ALSO

Before letting the user change the content of a letter, call the SMPPrepareToChange function (page 3-83). If the user changes the content of a letter, you must call the SMPContentChanged function (page 3-76) immediately.

When you have finished building your letter, you call the SMPEndSave function (described next) to save it. To cancel a save operation any time after you call the SMPBeginSave function, call the SMPEndSave function with the okToSave parameter set to false.

Letter file format is briefly discussed in "The Mailer Functions" beginning on page 3-4.

Use the SMPAddContent function (page 3-85) to add a standard interchange format block to a letter.

Use the SMPAddMainEnclosure function (page 3-90) to add an application document as a main enclosure to a letter.

Use the SMPImage function (page 3-88) to add an image block to a letter.

Use the SMPAddBlock function (page 3-91) to add a block to a letter.

SMPEndSave

The SMPEndSave function saves a letter to disk.

```
pascal OSErr SMPEndSave(WindowPtr window,
                        Boolean okToSave);
```

window	The window containing the letter to be saved.
okToSave	A Boolean value that you can use to cancel the process of saving a letter. Specify false to cancel the save operation.

DESCRIPTION

After you have used the SMPBeginSave function to initiate the process of saving a letter and have added content or a main enclosure to the letter by calling the SMPAddContent, SMPAddBlock, or SMPAddMainEnclosure functions, you call the SMPEndSave function to save the letter to disk. You use the saveType parameter in the SMPBeginSave function to specify which File menu operation this represents: Save, Save As, or Save A Copy. In any case, some version of the file remains open after the file has been saved to disk.

To cancel a save operation any time after you call the SMPBeginSave function, call the SMPEndSave function with the okToSave parameter set to false.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$1270

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	Disk is full
`kSMPNoMailerInWindow`	–1909	No mailer is in window
`kSMPNoMatchingBegin`	–1913	`SMPBeginSave` was not called

SEE ALSO

You begin the process of saving a letter by calling the `SMPBeginSave` function (page 3-77).

Call the `SMPReadContent` function (page 3-98) to read the standard interchange contents of a letter.

Call the `SMPGetMainEnclosureFSSpec` function (page 3-103) to obtain the file system specification for the main enclosure of a letter.

SMPBeginSend

You must call the `SMPBeginSend` function before you send a letter.

```
pascal OSErr SMPBeginSend(WindowPtr window,
                          OSType creator,
                          OSType fileType,
                          SMPSendOptionsPtr sendOptions,
                          Boolean *mustAddContent);
```

`window` The window containing the letter to be sent.

`creator` The creator for the file, which also becomes the letter's creator for a new letter. Use the same creator that you use for all of your application's documents.

`filetype` The file type, which also becomes the letter type for a new letter. Letters containing only AOCE standard content should be of type `'lttr'`.

`sendOptions`
The pointer to a send-options structure that was returned by the `SMPSendOptionsDialog` function.

`mustAddContent`
A pointer to a Boolean value returned by the function that tells you whether you have to add any blocks or enclosures to the letter. If this parameter is set to `false`, call the `SMPEndSave` function immediately without adding blocks or enclosures to the letter.

DESCRIPTION

When you send a letter, it is in letter file format rather than the document format normally used by your application. A letter consists of a header, data blocks, and enclosures. Every block has a creator and type, and every letter has a creator and type.

Before you send a letter, you should call the `SMPSendOptionsDialog` function to let the user set the send options for that letter. To begin the process of sending a letter, you call the `SMPBeginSend` function. This function prepares a letter file into which you can place your document.

If the user elected to send the letter as a document, you can create a new file format for the document that takes advantage of the block structure of a letter file, or you can save the document in one of your application's native formats to a temporary file on disk and then add that file as the main enclosure to the letter. You should also add a standard interchange format block to the letter. If the user elected to send the letter as an image, you must also add an image block to the letter. The Standard Mail Package does not actually send the letter until you call the `SMPEndSend` function.

If neither your application nor the user has changed the content of a received letter, the `SMPBeginSend` function returns a value of `false` for the `mustAddContent` parameter. In that case you can call the `SMPEndSend` function immediately to send the letter. If you have changed the content of the letter, however, the `mustAddContent` parameter returns `true` and you must build the letter (adding the appropriate combination of blocks, main enclosure, standard interchange format block, and image block) just as if it were a new letter. The Standard Mail Package handles enclosures added by the user. Note that, if you make changes to the enclosed original letter, you invalidate any digital signature.

IMPORTANT

Be sure to call the `SMPContentChanged` function whenever the user changes the content of a letter. If you do not call the `SMPContentChanged` function, then the `SMPBeginSend` function does not know that the letter has been changed and won't return `true` as the value of `mustAddContent`. s

SPECIAL CONSIDERATIONS

Any time the user changes the content of a letter, you must call the `SMPContentChanged` function immediately. The `SMPBeginSend` function needs this information to operate correctly.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$000A	$1267

RESULT CODES

noErr	0	No error
kSMPNoMailerInWindow	–1909	No mailer is in window
kMSPCannotSendReceivedLetter	–1914	Letter is received; cannot send

SEE ALSO

Call the SMPSendOptionsDialog function (page 3-73) before calling the SMPBeginSend function to let the user set send options.

If the user changes the content of a letter, you must call the SMPContentChanged function (page 3-76) immediately.

When you have finished building your letter, you call the SMPEndSend function (page 3-84) to send it. To cancel a send operation any time after you call the SMPBeginSend function, call the SMPEndSend function with the okToSend parameter set to false.

Letter file format is briefly discussed in "The Mailer Functions" beginning on page 3-4.

Use the SMPAddContent function (page 3-85) to add a standard interchange format block to a letter.

Use the SMPAddMainEnclosure function page 3-90) to add an application document as a main enclosure to a letter.

Use the SMPImage function (page 3-88) to add an image block to a letter.

Use the SMPAddBlock function (page 3-91) to add other blocks to a letter.

SMPPrepareToChange

The SMPPrepareToChange function checks whether the letter has any digital signatures that might be invalidated if the user changes the content.

```
pascal OSErr SMPPrepareToChange(WindowPtr window)
```

window A pointer to the window containing the mailer.

DESCRIPTION

Before making a change in the content of a letter, call the SMPPrepareToChange function. If the letter has any digital signatures that might be invalidated by the change, the function displays a dialog box alerting the user and providing a chance to cancel the change. If the user does not cancel the change, you should implement the change and then call the SMPContentChanged function. If the user cancels the change, the function returns the userCanceledErr result code.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$1289

RESULT CODES

`noErr`	0	No error
`userCanceledErr`	–128	User clicked Cancel in dialog box
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

After changing the content of a letter, call the `SMPContentChanged` function (page 3-76).

SMPEndSend

The `SMPEndSend` function sends a letter.

```
pascal OSErr SMPEndSend(WindowPtr window,
                        Boolean okToSend);
```

`window` The window containing the letter to be sent.

`okToSend` A Boolean value that you can use to cancel the process of sending a letter. Specify `false` to cancel the send operation.

DESCRIPTION

After you have used the `SMPBeginSend` function to initiate the process of sending a letter and have created the letter by calling some or all of the `SMPAddContent`, `SMPAddBlock`, `SMPImage`, and `SMPAddMainEnclosure` functions, you call the `SMPEndSend` function to send the letter.

To cancel a send operation any time after you call the `SMPBeginSend` function, call the `SMPEndSend` function with the `okToSend` parameter set to `false`.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$1271

RESULT CODES

noErr	0	No error
userCanceledErr	–128	User clicked Cancel button
kSMPNoMailerInWindow	–1909	No mailer is in specified window
kSMPNoMatchingBegin	–1913	SMPBeginSend was not called

SEE ALSO

You begin the process of sending a letter by calling the SMPBeginSend function (page 3-81).

To read the standard interchange contents of a letter you call the SMPOpenLetter function (page 3-94) and SMPReadContent function (page 3-98).

SMPAddContent

The SMPAddContent function adds a segment of a standard interchange format block to a letter.

```
pascal OSErr SMPAddContent(WindowPtr window,
                MailSegmentType segmentType,
                Boolean appendFlag,
                void *buffer,
                unsigned long bufferSize,
                StScrpRec *textScrap,
                Boolean startNewScript,
                ScriptCode script);
```

`window` The window containing the letter.

`segmentType`
A constant that indicates the type of data segment that you want to add to the letter. Letter segments may be text, picture, sound, styled text, or QuickTime movies. You can specify only one segment type in this parameter each time you call the SMPAddContent function. It may be any of the following constants:

`kMailTextSegmentType`
Text segment

`kMailPictSegmentType`
Picture segment

kMailSoundSegmentType
: Sound segment

kMailStyledTextSegmentType
: Styled text segment

kMailMovieSegmentType
: Movie segment

The Standard Mail Package also defines another MailSegmentType constant, kMailInvalidSegmentType, which you can use to initialize a variable, for example, in a type-safe manner without indicating that a valid segment has been passed.

appendFlag
: A Boolean value that indicates whether you want the SMPAddContent function to write the data in your buffer to a new segment or append it to the current segment. Set this parameter to false when you first call the SMPAddContent function. On subsequent calls to the function, set this parameter to false if you want to start a new segment. Set this parameter to true if you want to append the data in your buffer to the segment currently being written by the SMPAddContent function.

buffer
: A pointer to the data you want to add to the letter.

bufferSize
: The number of bytes of data you want to add to the letter.

textScrap
: A pointer to an StScrpRec structure that contains style information. You must provide this style information when your buffer contains styled text. Set this parameter to nil if you are not passing styled text data to the function.

startNewScript
: A Boolean value that indicates whether the text in your buffer uses a new character set. Set this parameter to true each time you call the SMPAddContent function to start a text segment. After that, set this parameter to true only if the data in your buffer is in a different character set than the data you previously provided to the function for that segment. The function ignores this parameter when you set the segmentType parameter to any value other than kMailTextSegmentType or kMailStyledTextSegmentType.

script
: A value that indicates the character set (Roman, Arabic, Kanji, etc.) of the data in your buffer. If you set the startNewScript parameter to true, set this parameter to the code for the text segment's character set. You cannot use the values smSystemScript or smCurrentScript for this parameter. The SMPAddContent function ignores this parameter when you set the segmentType parameter to any value other than kMailTextSegmentType or kMailStyledTextSegmentType.

DESCRIPTION

After you have called the SMPBeginSend or SMPBeginSave function, you can call the SMPAddContent function to add standard interchange format data to the letter that is in the window that you specify. The first time you call the function for a given letter, it

creates a new block and puts the data into the block. You then call the function repeatedly until you have finished adding standard interchange format data to the letter. Each time you call the `SMPAddContent` function, it adds data to that same block.

A standard interchange format 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 `SMPAddContent` function. The function adds the segments in the order that you provide them. A single letter may contain more than one segment of a given type.

A text segment contains one or more script runs. A script run is a string of text in the same character set. The `SMPAddContent` function can accommodate only one script at a time. Therefore, if you want to create a segment that contains several script runs, you must call this function once for each script run in the segment. Use the `script` parameter to specify the character set of the script run. Set the `startNewScript` parameter to `true` when you start a new text segment and to begin a new script run in the current text segment. To append text to the current script run, set the `startNewScript` parameter to `false` and the `appendFlag` parameter to `true`. If you add a segment of styled text, you must provide the style information in the `textScrap` parameter.

You cannot specify the values `smSystemScript` or `smCurrentScript` for the `script` parameter. To obtain the system script, call the `GetScriptManagerVariable` function with a selector of `smSysScript`. To obtain the current script, call the `FontScript` function.

Because font numbers are local to a given Macintosh computer, the fonts originally used in a letter might be different from those with the same font numbers on the receiving computer. For this reason, the `SMPAddContent` function creates a font table that associates a font name with each font number in the standard interchange format block of the letter. When you receive a letter, you can use the `SMPGetFontNameFromLetter` function to recover the names of the fonts originally used in a letter.

Once you begin creating a letter's standard interchange format block, you must not call other Standard Mail Package functions until you finish writing that block.

The data for picture segments must be in PICT format.

The data for sound segments must be in Audio Interchange File Format (AIFF).

The data for text and styled text segments must consist of 1-byte or 2-byte character codes, depending on the value in the `script` parameter. For styled text you must also provide a pointer to an `StScrpRec` structure in the `textScrap` parameter.

The data for QuickTime movie segments must be in the QuickTime movie format (`'MooV'`).

ASSEMBLY LANGUAGE INFORMATION

Parameter count	Routine selector
$000D	$127A

RESULT CODES

noErr	0	No error
dskFulErr	–34	Disk is full
memFullErr	–108	Not enough room in heap zone
kSMPShouldNotAddContent	–1903	You cannot add content to this letter
kSMPNoMailerInWindow	–1909	No mailer is in specified window

SEE ALSO

See "Summary of the Script Manager" at the end of the "Script Manager" chapter of *Inside Macintosh: Text*, for a list of script code constants.

See *Inside Macintosh: Imaging With QuickDraw* for more information about PICT images.

See *Inside Macintosh: Sound* for more information about AIFF.

The `StScrpRec` structure is described in *Inside Macintosh: Text* in the chapter "TextEdit."

The `StGetScriptManagerVariable` function and `FontScript` function are described in *Inside Macintosh: Text* in the chapter "Script Manager."

The `SMPReadContent` function is described on page 3-98.

You can use the `SMPGetFontNameFromLetter` function (page 3-102) to recover the names of the fonts originally used in a letter.

Call the `SMPAddMainEnclosure` function (page 3-90) to add a main enclosure to a letter. Call the `SMPImage` function (described next) to add an image to the letter. Use the `SMPAddBlock` function (page 3-91) to add other blocks to a letter.

SMPImage

The `SMPImage` function adds an image of a document to a letter.

```
pascal OSErr SMPImage (WindowPtr window,
                       SMPDrawImageProcPtr drawImageProc,
                       long imageRefCon,
                       Boolean supportsColor);
```

`window` The window containing the letter.

`drawImageProc`
A pointer to your image-drawing routine. If you want to send a letter as an image, you must provide a routine to draw the image. The procedure declaration for this routine is described on page 3-123.

`imageRefCon`
A reference constant for your use. The function passes this constant to your image-drawing routine.

`supportsColor`
A Boolean value that indicates whether the procedure pointed to by the `drawImageProc` parameter is capable of drawing in color. The Standard Mail Package provides a color graphics port to your image-drawing routine only if you specify `true` for the `supportsColor` field and the user has color QuickDraw.

DESCRIPTION

You can use the Standard Mail Package to send a letter as an image. You use the `SMPImage` function to create an image from your document and add it to a letter. When you call the `SMPImage` function, you provide a pointer to your drawing routine. The `SMPImage` function calls the drawing routine to draw the image of your document.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$1282

RESULT CODES

`noErr`	0	No error
`dskFulErr`	–34	Disk is full
`memFullErr`	–108	Not enough room in heap zone
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

The procedure declaration for your image-drawing routine is on page 3-123.

Mailers are described in "The Mailer Functions" beginning on page 3-4.

Call the `SMPSendOptionsDialog` function (page 3-73) to let the user set send options.

Call the `SMPBeginSend` function (page 3-81) to start the process of sending a letter that contains a mailer.

Call the `SMPAddMainEnclosure` function (described next) to add a main enclosure to a letter. Call the `SMPAddContent` function (page 3-85) to add standard interchange format content to a letter. Use the `SMPAddBlock` function (page 3-91) to add other blocks to a letter.

When you have finished building your letter, you call the `SMPEndSend` function (page 3-84) to send it.

SMPAddMainEnclosure

The `SMPAddMainEnclosure` function adds a main enclosure to a letter.

```
pascal OSErr SMPAddMainEnclosure(WindowPtr window,
                                 const FSSpec *enclosure);
```

`window` The window containing the letter.

`enclosure` A pointer to a file system specification structure that identifies the file that you want to enclose.

DESCRIPTION

When you are creating a letter, you can include a document in one of your application's native formats by first saving the document to disk and then calling the `SMPAddMainEnclosure` function to add the document to the letter as a main enclosure. If you must create a temporary file for this operation, create it in the Temporary Items folder at the root level of the startup volume. The main enclosure is not listed as an enclosure in the mailer.

SPECIAL CONSIDERATIONS

When you save the letter, the file system specification for the main enclosure changes so that the `FSSpec` structure you specified in the `enclosure` parameter is no longer valid. Use the `SMPGetMainEnclosureFSSpec` function to obtain the file system specification of the main enclosure of a letter.

ASSEMBLY LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$127D

RESULT CODES

`noErr`	0	No error
`fnfErr`	–43	File not found
`memFullErr`	–108	Not enough room in heap zone
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

Call the `SMPAddContent` function (page 3-85) to add standard interchange format content to a letter. Call the `SMPImage` function (page 3-88) to add an image to the letter. Use the `SMPAddBlock` function (page 3-91) to add other blocks to a letter.

You can use the `SMPGetMainEnclosureFSSpec` function (page 3-103) to obtain the `FSSpec` structure for the main enclosure of a letter.

The Temporary Items folder is described in the chapter "Finder Interface" in *Inside Macintosh: Macintosh Toolbox Essentials*.

SMPAddBlock

The `SMPAddBlock` function adds data to a block in a letter.

```
pascal OSErr SMPAddBlock(WindowPtr window,
                         const OCECreatorType *blockType,
                         Boolean append,
                         void *buffer,
                         unsigned long bufferSize,
                         MailBlockMode mode,
                         unsigned long offset);
```

`window` The window containing the letter.

`blockType` A pointer to a data structure that specifies the creator and type of the block you want to add. You may specify any value in the `msgCreator` field of the structure; usually your application signature. The `msgType` field identifies the type of block. You can define your own block types to serve your purposes. Apple Computer, Inc., reserves all block types consisting entirely of lowercase letters.

`append` A Boolean value that indicates whether you want the `SMPAddBlock` function to append the data in your buffer to the current block. Set this parameter to `false` when you call the function to start a new block. If you set this parameter to `true`, the function uses the `mode` and `offset` parameters to determine where to start writing.

`buffer` A pointer to your data buffer.

`bufferSize`
The number of bytes of data to write to the block.

`mode` The mode in which the offset parameter is to be interpreted. The function uses this field to determine whether to begin writing data relative to the end of the last data written, to the beginning of the message, or to the end of the block. See the discussion following these parameter descriptions for details. The function ignores this parameter if you set the `append` parameter to `false`.

`offset` An offset that the function uses when it calculates the starting point of the write operation. Set this value to 0 when you start a new block. See the following discussion for details. The function ignores this parameter if you set the `append` parameter to `false`.

DESCRIPTION

You call the SMPAddBlock function to write data into a block whose type you specify in the blockType field.

You can write data to a block that is too large to be written all at once by setting the append parameter to true after the first time you call the function and then calling the function repeatedly until you have written the entire block.

The Standard Mail Package uses a *mark* to point to the current location within a block that you are writing. After the SMPAddBlock function completes, the mark points to the end of the last byte written.

You use the mode and offset parameters to specify the point in the block at which the SMPAddBlock function starts writing. You can set the mode parameter 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 parameter as an offset from the beginning of the block. When you use this mode, you cannot set the offset parameter to a negative value.

kMailFromLEOB
: The function interprets the value in the offset parameter 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 parameter as an offset from the current position of 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.

To use the SMPAddBlock function to write data in several pieces sequentially into a block, call the function as many times as necessary, setting the mode parameter to kIPMFromMark and the offset parameter to 0 each time.

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

SPECIAL CONSIDERATIONS

Once you begin writing a block in a letter, you must finish writing the block, calling the SMPAddBlock function as many times as necessary to complete the block before starting another block or calling the SMPAddContent or SMPAddMainEnclosure functions.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$000C	$127F

RESULT CODES

noErr	0	No error
kSMPNoMailerInWindow	–1909	No mailer is in specified window

SEE ALSO

The OCECreatorType structure is described in "Creator Type Structure" on page 3-28.

Call the SMPAddMainEnclosure function (page 3-90) to add a main enclosure to a letter. Call the SMPAddContent function (page 3-85) to add standard interchange format content to a letter. Call the SMPImage function (page 3-88) to add an image to the letter.

Reading Mail

If you are retrieving mail using the Standard Mail Package, use the SMPOpenLetter function (page 3-94) to gain access to an existing letter. If the letter is in the In Tray, you can use the use the SMPGetLetterInfo function before you call SMPOpenLetter. The SMPGetLetterInfo function returns the name and type of the letter. Once you have opened a letter in the In Tray, you can use the SMPGetNextLetter function (page 3-97) to open the next or preceding letter in the tray.

While a letter is open, you can examine the standard interchange contents of the letter by calling the SMPReadContent function (page 3-98) and can examine the letter's main enclosure by calling the SMPGetMainEnclosureFSSpec function (page 3-103). You can use the SMPGetFontNameFromLetter function (page 3-102) to determine the original fonts used in the standard interchange content block of a letter. You can use the SMPEnumerateBlocks function (page 3-104) to list all the blocks in a letter and the SMPReadBlock function (page 3-106) to read any block in a letter, including an image block.

SMPGetLetterInfo

The SMPGetLetterInfo function returns information about a letter in the In Tray.

```
pascal OSErr SMPGetLetterInfo(LetterSpec *mailboxSpec,
                              SMPLetterInfo *info);
```

mailboxSpec
: A pointer to a letter-specification structure. The LetterSpec structure is defined on page 3-35.

`info` A pointer to a letter information structure. The `SMPLetterInfo` structure is defined on page 3-27.

DESCRIPTION

The `SMPGetLetterInfo` function lets you determine the creator and letter type of a letter in the In Tray, together with the subject and sender of the letter. You can use this information to title windows or in a dialog box if you cannot open the letter for some reason.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$128A

RESULT CODES

`noErr`	0	No error
`kSMPMailboxNotFound`	–1904	Cannot find mailbox

SEE ALSO

Use the `SMPOpenLetter` function (described next) to open a letter for reading.

SMPOpenLetter

The `SMPOpenLetter` function opens a letter so you can read the contents.

```
pascal OSErr SMPOpenLetter(const LetterDescriptor *letter,
                           WindowPtr window,
                           Point upperLeft,
                           Boolean canContract,
                           Boolean initiallyExpanded,
                           const PrepareMailerForDrawingProcPtr
                                          prepareMailerForDrawingCB,
                           long clientData);
```

`letter` A pointer to the letter descriptor of the letter you want to open. The letter descriptor specifies whether the letter is on disk or in the In Tray and provides the file system specification structure or letter-specification structure of the letter.

`window` The window in which you want to display the opened letter. This window must not contain a mailer at the time you call the `SMPOpenLetter` function.

`upperLeft` The upper-left corner of the mailer in your window's local coordinates. This position is normally (0, 0).

`canContract`
A Boolean value that specifies whether it should be possible to contract and expand the mailer. Specify `true` if you want the mailer to have this ability.

`initiallyExpanded`
A Boolean value that indicates whether you want the mailer displayed initially in its expanded or contracted state. Specify `true` to display the mailer initially expanded.

`prepareMailerForDrawingCB`
A pointer to your drawing-preparation function. If you change the clip region, coordinates, or other aspects of your window's graphics port, you must provide this function to restore the graphics port to a standard state so that the Standard Mail Package can draw a mailer in your window. The drawing-preparation function is described on page 3-122. Specify `nil` for this parameter if you are not providing a drawing-preparation function.

`clientData`
Reserved for your use. The `SMPOpenLetter` function passes this value unaltered to your callback routine.

DESCRIPTION

You call the `SMPOpenLetter` function when you receive an Apple event to open a letter or when the user chooses the Open command. This function displays a mailer in the window you specify and opens the letter so you can read its contents.

The user can double-click an icon in the In Tray to open a letter or double-click the icon for a letter file on disk. Your Open item in the File menu should also open letter files on disk. There is no way to open a letter in the In Tray from a menu in your application; the user must use the Finder to open the letter. The Finder determines the owner of the letter and sends an `'aevt' 'odoc'` Apple event to the appropriate application, which can then open it. The Apple event contains the letter-specification structure, which you put in the letter descriptor and pass to the `SMPOpenLetter` function. If the user chooses the Open command to open the letter, you can get the file system specification structure from the Standard File Package.

Whether a letter is on disk or not, the Standard Mail Package treats the letter's enclosures as if they are stored in a folder in an external file system. That folder contains all of the enclosures added by the user through the mailer or by the `SMPAddAttachment` function, and might contain the letter's main enclosure, if any.

(You can use the `SMPAddMainEnclosure` function to add a main enclosure to a letter. The main enclosure is not displayed in the Enclosures field of a mailer.)

The Standard Mail Package displays the enclosures added by the user in the Enclosures field of the mailer and handles the user interface for those enclosures. To obtain the file system specification structure of the main enclosure to a letter, use the `SMPGetMainEnclosureFSSpec` function. Use the `SMPGetListItemInfo` function to get the file system specification for the enclosures added by the user.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$000C	$1268

RESULT CODES

`noErr`	0	No error
`kSMPMailerAlreadyInWindow`	–1911	Specified window has a mailer
`kMailInvalidSeqNum`	–15041	Invalid letter sequence number

SEE ALSO

The `LetterDescriptor` data type is described in “Letter Descriptor” on page 3-27.

Use the `SMPGetMainEnclosureFSSpec` function (page 3-103) to obtain the file system specification structure of the main enclosure of a letter.

Use the `SMPGetListItemInfo` function (page 3-113) to get the file system specification for the enclosures added by the user.

You can use the `SMGetComponentInfo` function (page 3-111) to obtain the file system specification structure of the enclosures folder for the letter.

You can use the `SMPGetNextLetter` function (described next) to determine which letter in the In Tray is the oldest unread letter.

SMPGetNextLetter

The `SMPGetNextLetter` function returns the letter descriptor of the In Tray item to be opened next.

```
pascal OSErr SMPGetNextLetter(
                            OSType *typesList,
                            short numTypes,
                            LetterDescriptor *adjacentLetter);
```

`typesList` A pointer to a list of letter types and file types. The function returns letter descriptors only for items with the letter types and file types specified in this list. Letters containing only AOCE standard content are of type `'lttr'`. Use the wildcard letter type `'ltr*'` if you want the function to return letter descriptors for all the items in the In Tray.

`numTypes` The number of letter types and file types in the types list pointed to by the `typesList` parameter.

`adjacentLetter`
The letter descriptor returned by the function identifying the next item to open.

DESCRIPTION

The letter descriptor returned by the `SMPGetNextLetter` function is that of the oldest unread letter in the In Tray. You can use this function to open letters in age sequence.

You can use the `typesList` parameter to specify what letter types and file types the function should include when it returns a letter descriptor.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0008	$1286

RESULT CODES

`noErr`	0	No error
`kSMPMailboxNotFound`	–1904	Cannot find mailbox
`kSMPNoNextLetter`	–1905	There is no next letter in the In Tray

SEE ALSO

Use the `SMPOpenLetter` function (page 3-94) to open a letter for reading.

The letter descriptor (`LetterDescriptor` data type) is described in "Letter Descriptor" on page 3-27.

SMPReadContent

The `SMPReadContent` function reads a segment from a letter's standard interchange format block.

```
pascal OSErr SMPReadContent(WindowPtr window,
                MailSegmentMask segmentTypeMask,
                void *buffer,
                unsigned long bufferSize,
                unsigned long *dataSize,
                StScrpRec *textScrap,
                ScriptCode *script,
                MailSegmentType *segmentType,
                Boolean *endOfScript,
                Boolean *endOfSegment,
                Boolean *endOfContent,
                long *segmentLength,
                long *segmentID);
```

`window` The window containing the letter.

`segmentTypeMask`
The type of segment that you want to retrieve. You can request a combination of segment types by performing a bitwise OR operation on the following constants:

`kMailTextSegmentMask`
Text segment

`kMailPictSegmentMask`
Picture segment

`kMailSoundSegmentMask`
Sound segment

`kMailStyledTextSegmentMask`
Styled text segment

`kMailMovieSegmentMask`
Movie segment

You can request any combination of segment types, except that you cannot combine the `kMailTextSegmentMask` and `kMailStyledTextSegmentMask` constants in the same request. 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.

The `SMPReadContent` function reads this parameter only the first time you call it for a given letter. The next and subsequent times you call this function for the same letter, it ignores this parameter. The function returns data of a single segment type each time you call it.

`buffer` A pointer to the buffer you are providing to hold the data read from the letter.

`bufferSize`
The size of the buffer you are providing.

`dataSize` A pointer to the amount of data returned in your buffer.

`textScrap` A pointer to an `StScrpRec` structure. You must allocate this pointer. Set the first field of this structure (`scrpNStyles`) to the number of styles your buffer can hold. When the function writes styled text to your buffer, it returns style information in this structure and sets the `scrpNStyles` field to the actual number of styles returned.

`script` A pointer to the script code, which indicates the character set (Roman, Arabic, Kanji, etc.) of the text that the function placed in your buffer. If the function placed nontext data in your buffer, it does not set this parameter and you should ignore it.

`segmentType`
A constant that indicates the type of data segment that the `SMPReadContent` function returned in your buffer. It may be any of the following constants:

`kMailTextSegmentType`
Text segment

`kMailPictSegmentType`
Picture segment

`kMailSoundSegmentType`
Sound segment

`kMailStyledTextSegmentType`
Styled text segment

`kMailMovieSegmentType`
Movie segment

The Standard Mail Package also defines another `MailSegmentType` constant, `kMailInvalidSegmentType`, which you can use to initialize a variable, for example, in a type-safe manner without indicating that a valid segment has been passed.

`endOfScript`
A pointer to a Boolean value returned by the function that indicates whether the text placed in your buffer is the end of a script run. A script run is a sequence of text in a single character set. If there is more text in the current script run, the function sets this parameter to `false`.

endOfSegment
: A pointer to a Boolean value returned by the function that indicates whether you have received all of the data in the segment. The SMPReadContent function sets this parameter to true when it has returned all of the data in a given segment. It sets this parameter to false when there is more data in that segment to return.

endOfContent
: A pointer to a Boolean value returned by the function that indicates if there is more data in the standard interchange format block of the letter to be read. If the SMPReadContent function has returned the entire contents of the block, it sets the endOfContent parameter to true; otherwise, it sets this parameter to false.

segmentLength
: A pointer to the size of the current segment. The SMPReadContent function returns a valid value for this parameter the first time you call the function to read a particular segment.

segmentID
: A pointer to the ID of this segment. If you specify 0 for this parameter, the SMPReadContent function reads the next segment sequentially, returning the segment ID in this parameter. If you specify a value, SMPReadContent reads the segment with the specified ID. Note that this number is not an index number; it is an ID that is unique for the beginning of each segment. The number returned by the function in this parameter when you continue reading in the middle of a segment is undefined. You must set this parameter to 0 the first time you call the function.

DESCRIPTION

You call the SMPReadContent function to read some or all of a letter's standard interchange format block. You must call the SMPOpenLetter function before the first time you call the SMPReadContent function for a given letter. You then call the SMPReadContent function repeatedly to read all of the segments of the types you specified the first time you called the function. Once the SMPReadContent function has returned true for the endOfContent parameter, you must call the SMPOpenLetter function again before you can call the SMPReadContent function again.

The SMPReadContent function examines the value of the segmentTypeMask parameter the first time you call it for a given letter and uses that same value until you start the sequence over by calling the SMPOpenLetter function again. The SMPReadContent function returns segments in the order that they are stored in the letter.

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 SMPReadContent function returns text data (that is, when the function sets the segmentType parameter to kMailTextSegmentType), it

indicates the character set by setting the `script` parameter. The function identifies the end of a script run by setting the `endOfScript` parameter to `true`.

The `SMPReadContent` function returns, in the `dataSize` parameter, a pointer to the actual number of bytes written to your buffer. If your buffer is not large enough to hold all of the data in a segment, the function sets the `endOfSegment` parameter to `false`. You can call the function again to continue reading data from that segment.

If a single segment of styled text contains more styles than your `StScrpRec` structure can hold, the `SMPReadContent` function stops writing data to your buffer and sets the `endOfSegment` parameter to `false`. You can use the `dataSize` parameter to determine how many bytes of text were written to your buffer. The next time to call the function, it continues writing text from the same segment into your buffer and putting text styles in your `StScrpRec` structure. In this case, the offsets in the `scrpStartChar` field of the script table of the `StScrpRec` structure apply only to the data currently in your data 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 that is 120 bytes long and contains 12 different styles. The 11th 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 `StScrpRec` structure can hold only 10 styles. In this case, the `SMPReadContent` function stops writing data to your buffer after it has placed 10 styles in your `StScrpRec` structure. Because these 10 styles applied to the first 90 bytes of text, the `dataSize` parameter indicates that 90 bytes of data were written to your buffer and the `endOfSegment` parameter is `false`.

The next time you call the `SMPReadContent` function, it writes the last 30 bytes of text into your buffer and puts the last two styles into your `StScrpRec` structure. It returns a value of 2 in the `scrpNStyles` field of your `StScrpRec` structure and sets the `endOfSegment` parameter to `true`. In this case, the first offset in the `scrpStartChar` field of the script table of the `StScrpRec` 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 `StScrpRec` structure been able to hold all of the styles at once.)

The data for picture segments is in PICT format.

The data for sound segments is in Audio Interchange File Format (AIFF).

The data for text and styled text segment consists of 1-byte or 2-byte character codes, depending on the value in the `script` parameter. For styled text the function also returns a pointer to an `StScrpRec` structure in the `textScrap` parameter.

The data for QuickTime movie segments must be in the QuickTime movie format (`'MooV'`).

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0019	$127B

RESULT CODES

noErr	0	No error
kSMPNoMailerInWindow	–1909	No mailer is in specified window
kMailMalformedContent	–15061	A mailed structure is malformed

SEE ALSO

See *Inside Macintosh: Imaging With QuickDraw* for more information about PICT images.

See *Inside Macintosh: Sound* for more information about AIFF.

The StScrpRec structure is described in the chapter "TextEdit" of *Inside Macintosh: Text*.

SMPGetFontNameFromLetter

The SMPGetFontNameFromLetter function converts the font numbers in the standard interchange format block of a letter into font names.

```
pascal OSErr SMPGetFontNameFromLetter(WindowPtr window,
                                      short fontNum,
                                      str255 fontName,
                                      Boolean doneWithFontTable);
```

window The window containing the letter.

fontNum The font number you read from the text scrap (the TextEdit structure) for the text.

fontName The name of the font associated with the font number.

doneWithFontTable
A Boolean value that you set to indicate that this is the last request for a font name.

DESCRIPTION

You can use the SMPGetFontNameFromLetter function to recover the names of the fonts originally used in a letter. Because font numbers are local to a given Macintosh computer, the fonts originally used in a received letter might be different from those with the same font numbers on the local computer. For this reason, when the Standard Mail Package sends a letter, it creates a font table that associates a font name with each font number in the standard interchange format block of the letter.

To recover the font names, you must first call the SMPReadContent function to read the standard content block of the letter, and then read the font numbers from the StScrpRec structure associated with each styled-text segment in that block. Then you can call the SMPGetFontNameFromLetter function once for each font number.

Set the `doneWithFontTable` parameter to `true` the last time you call the `SMPGetFontNameFromLetter` function. Doing so signals the Standard Mail Package to release the memory it has reserved for the font table.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0006	$127C

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

The `StScrpRec` structure is described in the chapter "TextEdit" of *Inside Macintosh: Text*.

SMPGetMainEnclosureFSSpec

The `SMPGetMainEnclosureFSSpec` function returns the file specification of the main enclosure file for a letter.

```
pascal OSErr SMPGetMainEnclosureFSSpec (WindowPtr window,
                                        FSSpec *enclosureDir);
```

`window` The window containing the letter.

`enclosureDir`
A pointer to the file system specification structure of the main enclosure.

DESCRIPTION

You can call the `SMPGetMainEnclosureFSSpec` function to get the file system specification for the main enclosure file for a letter. The main enclosure contains the letter's content, usually in your application's native document format. You can then use standard File Manager routines to open and read the main enclosure. The file system specification returned by this function is valid until the mailer is disposed of or until the next time the user saves the letter. You must call the `SMPOpenLetter` function before you call the `SMPGetMainEnclosureFSSpec` function.

If the letter does not contain a main enclosure, the function returns the result code `fnfErr` (file not found).

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$127E

RESULT CODES

noErr	0	No error
fnfErr	–43	File not found
kSMPNoMailerInWindow	–1909	No mailer is in specified window

SEE ALSO

You use the SMPAddMainEnclosure function (page 3-90) to add a main enclosure to a letter.

File Manager routines are described in *Inside Macintosh: Files.*

SMPEnumerateBlocks

The SMPEnumerateBlocks function returns information about the blocks in a letter.

```
pascal OSErr SMPEnumerateBlocks (WindowPtr window,
                                 unsigned short startIndex,
                                 void *buffer,
                                 unsigned long bufferSize,
                                 unsigned long *dataSize,
                                 unsigned short *nextIndex,
                                 Boolean *more);
```

`window` The window containing the letter.

`startIndex`
The sequence number of the next block for which you want the function to return information. Sequence numbers start with 1. When you call the SMPEnumerateBlocks function and there is insufficient space in the buffer you provide to hold information about all of the remaining blocks, the function returns, in the nextIndex parameter, the sequence number of the next block. Use that number as the value of the startIndex parameter the next time you call the function.

`buffer` A pointer to a buffer you provide to hold the information returned by the function. The block information is in the form of a count byte, indicating the number of blocks in the letter, followed by a block information structure for each block.

`bufferSize`
The length, in bytes, of the buffer you are providing.

`dataSize` The address at which the function places the number of bytes written to your buffer.

`nextIndex` The address at which the function places 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 function sets the sequence number to 0.

`more` A Boolean value returned by the function indicating whether there is more block information to be returned. If your buffer is too small to hold all of the information that you requested, the `SMPEnumerateBlocks` function sets this parameter to `true` and returns, in the `nextIndex` parameter, the sequence number of the next item to be returned.

DESCRIPTION

You can use the `SMPEnumerateBlocks` function to determine the number of blocks that are contained in a letter and each block's type and size. You can use this information to read specific blocks in the letter. You must call the `SMPOpenLetter` function before the first time you call the `SMPEnumerateBlocks` function for a given letter.

Apple Computer, Inc., reserves all block types that consist of all lowercase letters for its own use. Use the `SMPReadBlock` function to read image blocks and blocks of types that you define. Use the `SMPReadContent` function to read the standard interchange format block.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$000D	$1281

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

Use the `SMPReadBlock` function (described next) to read the contents of a block.

Use the `SMPReadContent` function (page 3-98) to read the standard interchange format block in a letter.

SMPReadBlock

The SMPReadBlock function reads a block from a letter that you specify.

```
pascal OSErr SMPReadBlock (WindowPtr window,
                           const OCECreatorType *blockType,
                           unsigned short blockIndex,
                           void *buffer,
                           unsigned long bufferSize,
                           unsigned long dataOffset,
                           unsigned long *dataSize,
                           Boolean *endOfBlock,
                           unsigned long *remaining);
```

`window` The window containing the letter.

`blockType` A pointer to a structure that specifies the creator and the type of the block that you want to read.

`blockIndex`
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 `SMPReadBlock` function and increment it by 1 each subsequent time you call the function until you have read all blocks of that type in the letter.

`buffer` A pointer to your data buffer. The `SMPReadBlock` function writes the information that you request into your buffer and sets the `dataSize` field to the number of bytes written.

`bufferSize`
The length, in bytes, of the buffer you are providing.

`dataOffset`
The offset relative to the beginning of the block of the byte at which you want the `SMPReadBlock` function to begin reading. Set this field to 0 to read from the beginning of the block.

`dataSize` A pointer to the number of bytes written to your buffer.

`endOfBlock`
A pointer to a Boolean value that indicates if the `SMPReadBlock` function has reached the end of the block. If the buffer that you provide is not large enough to contain the data remaining in the block, the `SMPReadBlock` function sets this parameter to `false`. You can call the function again with an updated value in the `dataOffset` parameter to retrieve additional data.

`remaining` A pointer to the number of bytes of data remaining in the block. You can use the value returned by this parameter to adjust the size of your data buffer before the next time you call the function. When the function sets the `endOfBlock` parameter to `true`, it sets the number of bytes remaining to 0.

DESCRIPTION

You call the `SMPReadBlock` function to read data from a specific block in a letter. You identify the block that you want to read by the values of the `blockType` and `blockIndex` parameters.

You can use this function to read an image block (a block with creator type `'apml'` and block type `'imag'`) or any block of a type you define.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0012	$1280

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

You can use the `SMPEnumerateBlocks` function (page 3-104) to list the block types and sizes of blocks in a letter.

The `OCECreatorType` data structure is described in “Creator Type Structure” on page 3-28.

The section “Image Block Information Structure” on page 3-28 describes how to read an image block.

Printing Mailers

If you are printing or imaging a letter, you should print or image the mailers as cover pages. You use the `SMPPrepareCoverPages` function (described next) to determine the total number of cover pages and the `SMPDrawNthCoverPage` function (page 3-108) to draw each cover page.

SMPPrepareCoverPages

The `SMPPrepareCoverPages` function prepares cover pages for a letter and returns the number of cover pages that are needed to print all of the mailers for the letter.

```
pascal OSErr SMPPrepareCoverPages(windowPtr window,
                                  short *pageCount);
```

`window`	The window for which you want the number of cover pages.
`pageCount`	A pointer to the number of cover pages necessary to print all the mailers in the specified window.

DESCRIPTION

When you print or image a letter, you can print or image the mailers as cover pages. You must call the `SMPPrepareCoverPages` function from within your printing or imaging routine to prepare the cover pages and to determine the number of cover pages before calling the `SMPDrawNthCoverPage` function.

SPECIAL CONSIDERATIONS

The `SMPPrepareCoverPages` function makes a number of calculations that are used by the `SMPDrawNthCoverPage` function. You must make sure that the mailer does not change between the time you call these two functions.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$1264

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

You call the `SMPDrawNthCoverPage` function, described next, to draw a cover page to the current graphics port.

SMPDrawNthCoverPage

The `SMPDrawNthCoverPage` function draws a cover page for a letter.

```
pascal OSErr SMPDrawNthCoverPage(WindowPtr window,
                                 short pageNumber,
                                 Boolean doneDrawingCoverPages);
```

`window` A pointer to the window for which you want to draw a cover page.

`pageNumber`
The number of the cover page you want to print.

`doneDrawingCoverPages`
A Boolean value that you set to `true` when you call the function for your last cover page. Doing so allows the Standard Mail Package to release the memory that it uses for drawing cover pages.

DESCRIPTION

Before you print or image a letter, you should include the mailers for that letter as cover pages. The `SMPDrawNthCoverPage` function draws or images one cover page. You must use the `SMPPrepareCoverPages` function first to prepare the cover pages and to determine the total number of cover pages for a given letter. You call these functions from within your drawing or imaging routine, and they draw to whatever graphics port you provide. You can use these routines for printing, preparing an image of your letter to be sent as electronic mail, or for display on the screen.

SPECIAL CONSIDERATIONS

The `SMPDrawNthCoverPage` function uses a number of calculations that are made by the `SMPPrepareCoverPages` function. You must make sure that the mailer does not change between the time you call these two functions.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$1265

RESULT CODES

`noErr`	0	No error
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window

SEE ALSO

The `SMPPrepareCoverPages` function is described on page 3-107.

For the sequence of routines you must call to image a letter, see the description of the image-drawing callback routine on page 3-123.

Getting and Setting Information in the Mailer

You can use the functions in this section to determine the contents of the fields of a mailer and to change the information in those fields without user interaction.

The SMPGetListItemInfo function (page 3-113) returns information from the Recipients or Enclosures fields of any mailer. The SMPGetComponentInfo function (page 3-111) returns information from any other field. Before calling either of these functions, you call the SMPGetComponentSize function (described next) to determine the size of the buffer to allocate.

You can put information into the fields of a mailer with the functions SMPSetSubject (page 3-116), SMPSetFromIdentity (page 3-117), SMPAddAddress (page 3-118), and SMPAddAttachment (page 3-119).

SMPGetComponentSize

The SMPGetComponentSize function returns the size of the buffer that would be required to hold all of the information in a specific field of the mailer you specify.

```
pascal OSErr SMPGetComponentSize(WindowPtr window,
                                 unsigned short whichMailer,
                                 SMPMailerComponent whichField,
                                 unsigned short *size);
```

window The window containing the mailer from which you want information.

whichMailer
The sequence number of the mailer from which you want information. The original mailer for the letter is number 1, and each forwarding mailer is numbered sequentially.

whichField
The field from which you want to extract the information.

size A pointer to the number of bytes of data in the field you specified. For all fields except the Recipients and Enclosures fields, you should allocate a buffer of this size and call the SMPGetComponentInfo function to obtain the contents of that field. The Recipients and Enclosures fields might contain more data than it is practical to retrieve all at once; see the description of the SMPGetListItemInfo function (page 3-113) for more information.

DESCRIPTION

The SMPGetComponentSize function returns the number of bytes of data in any of the fields in a mailer. You specify which field by using one of the following constants for the whichField parameter: kSMPFrom, kSMPTo, kSMPRegarding, kSMPSendDateTime, or kSMPAttachments.

If you specify any other value for the `whichField` parameter, the function returns the `kSMPIllegalComponent` result code.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0007	$1277

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in user parameter list
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPIllegalComponent`	–1918	Illegal value for `whichField` parameter

SEE ALSO

You can use the `SMPGetMailerState` function (page 3-69) to get the total number of mailers for a given letter.

Use the `SMPGetComponentInfo` function (described next) to get data from any field except the Recipients and Enclosures fields. Use the `SMPGetListItemInfo` function (page 3-113) to get data from the Recipients and Enclosures fields.

All possible values for the `SMPMailerComponent` data type are shown on page 3-32.

SMPGetComponentInfo

The `SMPGetComponentInfo` function returns information from the From, Subject, and Sent fields of a mailer.

```
pascal OSErr SMPGetComponentInfo(WindowPtr window,
                                 unsigned short whichMailer,
                                 SMPMailerComponent whichField,
                                 void *buffer);
```

`window` The window containing the mailer from which you want information.

`whichMailer`
The sequence number of the mailer from which you want information. The original mailer for the letter is number 1, and each forwarding mailer is numbered sequentially.

`whichField`
: The field from which you want to extract the information.

`buffer`
: A pointer to a buffer you provide into which the function places the information you requested. Use the `SMPGetComponentSize` function to determine what size to make this buffer.

DESCRIPTION

The `SMPGetComponentInfo` function returns information from a mailer. You specify which field by using one of the following constants for the `whichField` parameter: `kSMPFrom` (for the From field), `kSMPRegarding` (for the Subject field), or `kSMPSendDateTime` (for the Sent field).

If you specify any other value for the `whichField` parameter, the function returns the `kSMPIllegalComponent` result code.

If you request information from the Subject field, then the function returns an `RString` structure containing the text of the field.

If you request information from the From field of a draft mailer, the function returns an `AuthIdentity` structure identifying the sender, followed by an `RString` structure containing the text in the From field. If you request information from the From field of a received mailer, the function returns an `OCEPackedRecipient` structure containing the address of the sender. Only the top mailer in a mailer set can be a draft mailer; use the `hasBeenReceived` field of the `SMPMailerState` structure to determine whether the top mailer has been received.

If you request information from the Date field of the mailer, then the function returns a `MailTime` structure. The time is defined with respect to the local computer that records it. The `offset` field in the `MailTime` structure corrects UTC time (also known as Greenwich Mean Time) for the local time zone. The `offset` field is in seconds; it is positive if east of Greenwich and negative if west of Greenwich.

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

typedef struct MailTime MailTime;

typedef  unsigned long  UTCTime;    /* seconds since 1/1/1904 */
typedef  long           UTCOffset;  /* correct for local time */
```

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0007	$1278

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in user parameter list
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPIllegalComponent`	–1918	Illegal value for `whichField` parameter

SEE ALSO

You can use the `SMPGetMailerState` function (page 3-69) to get the total number of mailers for a given letter and to determine if the top mailer has been received.

Use the `SMPGetComponentSize` function (page 3-110) to determine the size of the buffer to provide. Use the `SMPGetListItemInfo` function (described next) to get data from the Recipients and Enclosures fields.

All possible values for the `SMPMailerComponent` data type are shown on page 3-32.

SMPGetListItemInfo

The `SMPGetListItemInfo` function returns information from the Recipients or Enclosures fields of a mailer.

```
pascal OSErr SMPGetListItemInfo(WindowPtr window,
                                unsigned short whichMailer,
                                SMPMailerComponent whichField,
                                void *buffer,
                                unsigned long bufferLength,
                                unsigned short startItem,
                                unsigned short *itemCount,
                                unsigned short *nextItem,
                                Boolean *more);
```

`window` The window containing the mailer from which you want information.

`whichMailer`
The sequence number of the mailer from which you want information. The original mailer for the letter is number 1, and each forwarding mailer is numbered sequentially.

`whichField`
The field from which you want information; either `kSMPAttachments` or `kSMPTo`.

`buffer` A pointer to a buffer you provide to hold the information returned by the function.

`bufferLength` The length, in bytes, of the buffer you are providing.

`startItem` The sequence number of the first address or enclosure that the function should return. Sequence numbers start with 0. When you call the `SMPGetListItemInfo` function and there is insufficient space in the buffer you provide to hold all of the remaining items, the function returns, in the `nextItem` parameter, the sequence number of the next item. Use that number for the `startItem` parameter the next time you call the function. If there is insufficient space in the buffer to hold even one item, the number the function returns in the `nextItem` parameter is the same as the number you put in the `startItem` parameter. In that case, you must increase the buffer size before calling the function again.

`itemCount` A pointer to the number of items that the function has placed in the buffer. If the buffer is too small to hold the item you specify in the `startItem` parameter, then the `itemCount` parameter returns 0, and the `more` parameter returns `true`. If you specify `nil` for the `buffer` parameter and 0 for the `bufferLength` parameter, the `itemCount` parameter returns a pointer to the total number of items in the mailer field.

`nextItem` A pointer to the sequence number of the next item to be returned. If the `more` parameter returns `true`, set the `startItem` parameter to the number returned in the `nextItem` parameter and call the function again.

`more` A pointer to a Boolean value returned by the function indicating whether there is more information to be returned. If your buffer was not large enough to hold all of the requested data, the function sets this parameter to `true` and returns, in the `nextItem` parameter, the sequence number of the next item to be returned.

DESCRIPTION

Before you call the `SMPGetListItemInfo` function, call the `SMPGetComponentSize` function with a value of `kSMPTo` or `kSMPAttachments` for the `whichField` parameter. The `SMPGetComponentSize` function returns the total number of bytes of storage space required to hold all of the information you requested. You can then allocate a buffer to hold the data returned by the `SMPGetListItemInfo` function. If you can't (or don't want to) provide a buffer large enough to hold all of the information at once, you can allocate a smaller buffer.

If you cannot allocate a buffer large enough to hold all of the items at once, the function returns the sequence number of the next item in the `nextItem` parameter and it returns `true` for the `more` parameter. If the buffer is not large enough to hold even one item, the sequence number returned in the `nextItem` parameter is the same as the number you passed in the `startItem` parameter. You can then increase the size of your buffer if necessary, set the `startItem` parameter to the number just returned in the `nextItem` parameter, and call the function again.

You can specify either `kSMPTo` or `kSMPAttachments` for the `whichField` parameter.

If you request information from the Recipients field, for each address the function returns a short value indicating the type of address followed by an `OCEPackedRecipient` structure containing the address. The address type can be any of the following values:

```
enum {
   kSMPToAddress =      kMailToBit,
   kSMPCCAddress =      kMailCcBit,
   kSMPBCCAddress =     kMailBccBit
};

typedef MailAttributeID SMPAddressType;
```

If you request information from the Enclosures field, the function returns a file system specification structure (`FSSpec` data type) identifying the letter's enclosure folder. You can then use File Manager routines to determine the contents of that folder.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0010	$1279

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in user parameter list
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPIllegalComponent`	–1918	Illegal value for `whichField` parameter

SEE ALSO

You can use the `SMPGetMailerState` function (page 3-69) to get the total number of mailers for a given letter.

Use the `SMPGetComponentSize` function (page 3-110) to determine the size of the buffer to provide.

Use the `SMPGetComponentInfo` function (page 3-111) to get data from mailer fields other than the Recipients and Enclosures fields.

SMPSetSubject

The `SMPSetSubject` function specifies the subject string for the top mailer in the window you specify.

```
pascal OSErr SMPSetSubject(WindowPtr window,
                           const RString *text);
```

`window` The window containing the mailer.

`text` A pointer to the subject string you want to place in the mailer.

DESCRIPTION

The Standard Mail Package provides a user interface that lets a user enter a subject string in a mailer. You can use the `SMPSetSubject` function to set the subject string directly from your application. You can use this function, for example, to place an initial, default subject string in the subject field of a new mailer.

The `SMPSetSubject` function sets only the string in the most recent mailer for the window you specify, and then only if it is a draft mailer (that is, if it is not a received mailer). You can use the `hasBeenReceived` field of the `SMPMailerState` structure (a parameter of the `SMPGetMailerState` function) to determine whether the mailer is a draft mailer.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$126B

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in user parameter list
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPMailerUneditable`	–1912	Mailer cannot be edited
`kSMPSubjectTooBig`	–1925	Subject string exceeds 127 characters

SEE ALSO

You can use the `SMPGetMailerState` function (page 3-69) to determine if the top mailer is for a received letter.

SMPSetFromIdentity

The `SMPSetFromIdentity` function sets the authentication identity for the sender of a letter.

```
pascal OSErr SMPSetFromIdentity(WindowPtr window,
                                AuthIdentity from);
```

`window` The window containing the mailer.

`from` The authentication identity you want to use for that mailer. Specify 0 to use the identity of the most recently authenticated user.

DESCRIPTION

The `SMPSetFromIdentity` function lets you change the contents of the From field of a mailer from within your application. The `SMPSetFromIdentity` function modifies only the most recent mailer in the specified window, and then only if it is not a received mailer.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$126C

RESULT CODES

`noErr`	0	No error
`kOCEUnknownID`	–1567	Identity passed is not valid
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPMailerUneditable`	–1912	Mailer cannot be edited

SEE ALSO

Use the `SDPPromptForID` function in the chapter “Standard Catalog Package” in this book to obtain an authentication identity.

SEE ALSO

You can use the `SMPGetMailerState` function (page 3-69) to determine if the top mailer is for a received letter.

SMPAddAddress

The SMPAddAddress function adds an address to the Recipients field of a mailer.

```
pascal OSErr SMPAddAddress(WindowPtr window,
                SMPAddressType addrType,
                OCEPackedRecipient *address);
```

window	The window containing the mailer.
addrType	The type of address you want to add. You can specify the value kSMPToAddress to add a primary addressee, or kSMPCCAddress to add a "copy to" addressee.
address	The address that you want to add to the mailer.

DESCRIPTION

The Standard Mail Package provides a user interface that lets a user enter an address in the Recipients field of a mailer. You can use the SMPAddAddress function to add an address directly from your application. You can use this function, for example, to place an initial, default address in the Recipients field of a reply mailer. If you specify an address type of kSMPCCAddress, the mailer flags the address as a "copy to" address (see Figure 3-3 on page 3-5). The values of the SMPAddressType data type are defined as follows:

```
enum {
   kSMPToAddress =      kMailToBit,
   kSMPCCAddress =      kMailCcBit,
   kSMPBCCAddress =     kMailBccBit
};

typedef MailAttributeID SMPAddressType;
```

The SMPAddAddress function adds addresses only to the most recent mailer in the specified window, and then only if it is not a received mailer. You can use the hasBeenReceived field of the SMPMailerState structure (a parameter of the SMPGetMailerState function) to determine whether the top mailer has been received.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0005	$126D

RESULT CODES

noErr	0	No error
kSMPNoMailerInWindow	–1909	No mailer is in specified window
kSMPMailerUneditable	–1912	Mailer cannot be edited
kSMPAddressAlreadyInList	–1922	Specified address is in Recipients field

SEE ALSO

You can use the SMPGetMailerState function (page 3-69) to determine if the top mailer is for a received letter.

You can use the SMPGetListItemInfo function (page 3-113) to get the addresses that the user entered in the Recipients field of a mailer.

SMPAddAttachment

The SMPAddAttachment function adds a disk file as an enclosure to a letter.

```
pascal OSErr SMPAddAttachment(WindowPtr window,
                              const FSSpec *attachment);
```

window The window containing the mailer.

attachment
A pointer to the file system specification structure of the disk file that you want to add as an enclosure.

DESCRIPTION

The Standard Mail Package provides a user interface that lets a user add a disk file or folder as an enclosure to a letter. You can use the SMPAddAttachment function to add an enclosure directly from your application in case you want to provide an Add Enclosures command. The SMPAddAttachment function adds enclosures only to the most recent mailer in the specified window, and then only if it is not a received mailer. You can use the hasBeenReceived field of the SMPMailerState structure (a parameter of the SMPGetMailerState function) to determine whether the top mailer has been received.

Use the SMPAddMainEnclosure function to add a main enclosure to the letter. The mailer does not display the contents of the letter's main enclosure in the Enclosures field.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

The enclosure is not actually added until well after this function has returned. Therefore, after calling the SMPAddAttachment function, you should call the WaitNextEvent

routine so that you yield time to the Finder to process Apple events while in the background. You must wait until the Standard Mail Package has finished copying the enclosure into the letter before you add anything else to the letter or try to send or save the letter. The `SMPMailerEvent` function uses the `kSMPCreateCopyWindowBit` and `kSMPDisposeCopyWindowBit` status bits to inform you of the progress of the copy operation.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$126E

RESULT CODES

`noErr`	0	No error
`fnfErr`	–43	File not found
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPMailerUneditable`	–1912	Mailer cannot be edited
`kSMPTooManyEnclosures`	–1928	More than 50 total files and folders

SEE ALSO

You can use the `SMPAttachDialog` function (described next) to display a dialog box that lets the user add an enclosure to a mailer.

You can use the `SMPGetMailerState` function (page 3-69) to determine if the top mailer is for a received letter.

You can use the `SMPGetListItemInfo` function (page 3-113) to list the enclosures that the user entered in the Enclosures field of a mailer.

Call the `SMPMailerEvent` function (page 3-63) to handle mailer events and to determine the status of the copy operation that occurs when you call the `SMPAddMainEnclosure` function.

Use the `SMPAddMainEnclosure` function (page 3-90) to add a main enclosure to a letter.

SMPAttachDialog

The `SMPAttachDialog` function displays a dialog box that lets the user add a disk file as an enclosure to a letter.

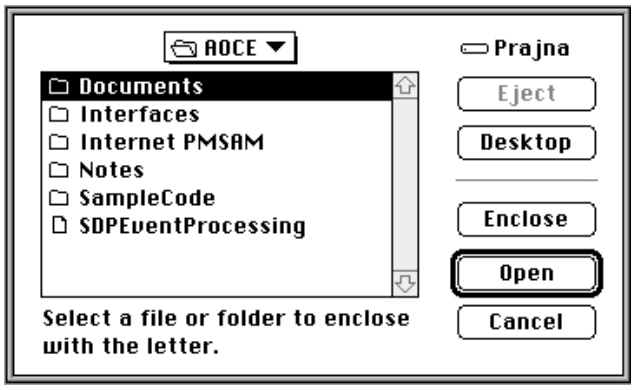
```
pascal OSErr SMPAttachDialog (WindowPtr window);
```

`window` The window containing the mailer.

DESCRIPTION

The Standard Mail Package provides a dialog box that lets a user add a disk file or folder as an enclosure to a letter (Figure 3-8). You can use the `SMPAttachDialog` function to display this same dialog box as an easy way to provide an Add Enclosures command. The `SMPAttachDialog` function adds enclosures only to the most recent mailer in the specified window, and then only if it is not a received mailer. You can use the `hasBeenReceived` field of the `SMPMailerState` structure (a parameter of the `SMPGetMailerState` function) to determine whether the top mailer is editable.

Figure 3-8 Add Enclosure dialog box



Use the `SMPAddMainEnclosure` function to add a main enclosure to the letter. The mailer does not display the contents of the letter's main enclosure in the Enclosures field.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

The enclosure is not actually added until well after this function has returned. Therefore, after calling the `SMPAttachDialog` function, you should call the `WaitNextEvent` routine so that you yield time to the Finder to process Apple events while in the background. You must wait until the Standard Mail Package has finished copying the enclosure into the letter before you add anything else to the letter or try to send or save the letter. The `SMPMailerEvent` function uses the `kSMPCreateCopyWindowBit` and `kSMPDisposeCopyWindowBit` status bits to inform you of the progress of the copy operation.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$1276

RESULT CODES

noErr	0	No error
userCanceledErr	–128	User clicked Cancel button
kSMPNoMailerInWindow	–1909	No mailer is in specified window
kSMPMailerUneditable	–1912	Mailer cannot be edited
kSMPTooManyEnclosures	–1928	More than 50 total files and folders

SEE ALSO

You can use the SMPAddAttachment function (page 3-119) if you want to provide your own interface that lets the user add an enclosure to a mailer.

You can use the SMPGetMailerState function (page 3-69) to determine if the top mailer is for a received letter and is therefore uneditable.

Call the SMPMailerEvent function (page 3-63) to handle mailer events and to determine the status of the copy operation that occurs when you call the SMPAddMainEnclosure function.

Use the SMPAddMainEnclosure function (page 3-90) to add a main enclosure to a letter.

Application-Defined Functions

This section describes the callback routines that you may provide for Standard Mail Package functions. Your MyPrepareMailerForDrawing routine restores your window port to a standard state so the Standard Mail Package can draw into it. Your MyDrawImage routine (page 3-123) images a document for the Standard Mail Package. The Standard Mail Package calls your MyFrontWindowCB routine (page 3-124) to determine which window is active when processing a key-down event.

MyPrepareMailerForDrawing

You may need to provide a MyPrepareMailerForDrawing routine to the SMPNewMailer function to make sure that the Standard Mail Package can draw a mailer in your window.

```
pascal void MyPrepareMailerForDrawing (WindowPtr window,
                                       long clientData);
```

window A pointer to the window into which the Standard Mail Package wants to draw.

clientData
Reserved for your use. You specify this value when you call the SMPNewMailer function, and that function passes the value unaltered to your callback routine.

DESCRIPTION

If you ever change the clip region, coordinates, or other aspects of your window's graphics port, you must provide a drawing-preparation routine that restores the window to its original state. The Standard Mail Package calls this routine before it draws into your window to add a new mailer or alter an existing mailer. You provide a pointer to your drawing-preparation routine when you call the `SMPNewMailer` function, the `SMPMailerReply` function, and the `SMPOpenLetter` function.

SPECIAL CONSIDERATIONS

You must make sure that your code for this routine is in a locked segment.

The `SMPNewMailer` function preserves your application's A5 world when it calls your drawing-preparation routine. Therefore, you have access to your application's global variables from this routine.

SEE ALSO

The `SMPNewMailer` function is described on page 3-46.

The `SMPMailerReply` function is described on page 3-51.

The `SMPOpenLetter` function is described on page 3-94.

MyDrawImage

The `MyDrawImage` function is a callback routine you must provide if you call the `SMPImage` function or if you specify `kSMPSendAsImageMask` for the `sendAs` field of the parameter block used by the `SMPSendLetter` function.

```
pascal void MyDrawImage (long refcon, Boolean inColor);
```

`refcon` A reference constant that you can use for your own purposes.

`inColor` A Boolean value that indicates whether the Standard Mail Package is providing a color graphics port to your image-drawing routine. This parameter is significant only in image-information structures passed to image-drawing routines.

DESCRIPTION

You provide a pointer to your image-drawing routine in the `drawImageProc` parameter when you call the `SMPImage` function and in the `drawImageProc` field of the parameter block when you call the `SMPSendLetter` function. Your image-drawing routine must call the `SMPNewPage` function before it draws each page of the document. You should call the `SMPImageErr` function rather than the `QDError` function after each QuickDraw routine you call. When you are finished imaging the document, just return.

If the user has color QuickDraw and you specified `true` for the `supportsColor` parameter of the `SMPImage` function or the `supportsColor` field of the parameter block used by `SMPSendLetter`, then the Standard Mail Package provides you with a color graphics port when it calls your image-drawing routine.

If you are imaging a letter that includes one or more mailers, you should image the mailers as cover pages before imaging the document. To do so, your image-drawing routine should first call the `SMPPrepareCoverPages` function to prepare the cover pages and to determine the total number of cover pages. Then for each cover page, you should call the `SMPNewPage` function and then the `SMPDrawNthCoverPage` function.

SPECIAL CONSIDERATIONS

If you change the graphics port within your image-drawing routine, you must change it back before calling the `SMPNewPage` or `SMPImageErr` functions.

SEE ALSO

The `SMPSendLetter` function is described on page 3-37. The `SMPImage` function is described on page 3-88.

You must call the `SMPNewPage` function (page 3-41) before you draw each page.

You should call the `SMPImageErr` function (page 3-41) after each QuickDraw routine you call.

To prepare cover pages for a mailer, you must call the `SMPPrepareCoverPages` function (page 3-107). To draw each cover page, you call the `SMPDrawNthCoverPage` function (page 3-108).

You can call the `GetPort` routine to determine the current graphics port. The `GetPort` routine is described in *Inside Macintosh: Imaging With QuickDraw*.

MyFrontWindowCB

The `MyFrontWindowCB` function is a callback routine you can provide with the `SMPMailerEvent` function. If you provide this function, the Standard Mail Package calls your `MyFrontWindowCB` function to determine which is the active window when processing a key-down event.

```
pascal WindowPtr MyFrontWindowCB (long clientData);
```

`clientData`
: Reserved for your use. You specify this value when you call the `SMPMailerEvent` function, and that function passes the value unaltered to your callback routine.

DESCRIPTION

You can provide a pointer to your front-window routine when you call the `SMPMailerEvent` function. Your front-window routine returns a pointer to the window that you want the `SMPMailerEvent` function to treat as the frontmost window. You might use this callback routine, for example, if your application displays a status dialog box in front of your application's main window on the screen, but you want any key-down events to apply to your application's main window. If, as is the case with most applications, you do not have any windows in front of your main application window, specify `nil` for the `frontWindowCB` parameter of the `SMPMailerEvent` function. In that case the Standard Mail Package uses the Window Manager's `FrontWindow` routine to determine the frontmost window.

SPECIAL CONSIDERATIONS

The `SMPMailerEvent` function preserves your application's A5 world when it calls your front-window routine. Therefore, you have access to your application's global variables from this routine.

SEE ALSO

The `SMPMailerEvent` function is described on page 3-63.

MySendOptionsFilterProc

The send-options filter procedure is a routine you can provide when you call the `SMPSendOptionsDialog` function. This routine extends the send-options dialog box.

```
pascal Boolean MySendOptionsFilterProc (DialogPtr theDialog,
                                        EventRecord* theEvent,
                                        short itemHit,
                                        long clientData);
```

`theDialog` A pointer to the dialog structure for the send-options dialog box.

`theEvent` The event that was just received by the send-options dialog box.

`itemHit` If the dialog box has just received a mouse-down event, this parameter indicates the number of the dialog item in which the mouse-down event occurred.

`clientData` A constant reserved for your use. You specify this value when you call the `SMPSendOptionsDialog` function.

DESCRIPTION

If you provide a filter routine when you call the `SMPSendOptionsDialog` function, the Standard Mail Package calls your filter routine each time it receives an event for the send-options dialog box. If your filter routine returns `true`, the Standard Mail Package assumes you handled the event. If your filter routine returns `false`, the Standard Mail Package handles the event normally. You can alter the event before returning `false`.

To allow your filter routine to add new items to the send-options dialog box and to clean up before it removes the dialog box, the Standard Mail Package sends your function two pseudoevents:

```
enum {
   kSMPSendOptionsStart       = -1,
   kSMPSendOptionsEnd         = -2
};
```

When your filter routine receives the `kSMPSendOptionsStart` event, you can call the `CountDITL` routine to determine the number of items already in the dialog box. You can then call the `AppendDITL` function to add new items to the dialog box, as follows:

```
AppendDITL(theDialog, myDITL, appendDITLBottom)
```

The parameter `myDITL` describes the new items you wish to add to the dialog box. When you begin numbering new items, increment by 1 the number returned by the `CountDITL` routine. When your filter routine receives the `kSMPSendOptionsStart` event, you can also allocate memory, initialize menus, and so forth.

Immediately before closing the send-options dialog box, the Standard Mail Package sends a `kSMPSendOptionsEnd` event to your filter routine. You should then deallocate any memory that you allocated earlier.

SPECIAL CONSIDERATIONS

Do not make any assumptions about the number or position of the standard items in the send-options dialog box, as Apple Computer, Inc., reserves the right to change this dialog box at any time.

SEE ALSO

The `SMPSendOptionsDialog` function is described on page 3-73.

Summary of the Standard Mail Package

C Summary

Constants and Data Types

```
#define gestaltSMPMailerVersion'malr'
#define gestaltSMPSPSendLetterVersion'spsl'
#define kSMPNativeFormatName'natv'

#define typeLetterSpec'lttr'

/* wildcard used for filtering letter types */

enum {
   FilterAnyLetter='ltr*',
   FilterAppleLetterContent='ltc*',
   FilterImageContent='lti*'
};

/* SMPPSendAs values.  You may add the following values together to determine
how the file is sent, but you may not set both kSMPSendAsEnclosureMask and
kSMPSendFileOnlyMask. */

enum {
   kSMPSendAsEnclosureBit, /* appears as letter with enclosures */
   kSMPSendFileOnlyBit,    /* appears as a file in mailbo. */
   kSMPSendAsImageBit      /* letter includes image of content */
};

/* values of SMPPSendAs */

enum {
   kSMPSendAsEnclosureMask = 1<<kSMPSendAsEnclosureBit,
   kSMPSendFileOnlyMask    = 1<<kSMPSendFileOnlyBit,
   kSMPSendAsImageMask     = 1<<kSMPSendAsImageBit
};

typedef Byte SMPPSendAs;
```

```
enum {
   kSMPAppMustHandleEventBit,
   kSMPAppShouldIgnoreEventBit,
   kSMPContractedBit,
   kSMPExpandedBit,
   kSMPMailerBecomesTargetBit,
   kSMPAppBecomesTargetBit,
   kSMPCursorOverMailerBit,
   kSMPCreateCopyWindowBit,
   kSMPDisposeCopyWindowBit
};

/* values of SMPMailerResult */

enum {
   kSMPAppMustHandleEventMask    = 1<<kSMPAppMustHandleEventBit,
   kSMPAppShouldIgnoreEventMask  = 1<<kSMPAppShouldIgnoreEventBit,
   kSMPContractedMask            = 1<<kSMPContractedBit,
   kSMPExpandedMask              = 1<<kSMPExpandedBit,
   kSMPMailerBecomesTargetMask   = 1<<kSMPMailerBecomesTargetBit,
   kSMPAppBecomesTargetMask      = 1<<kSMPAppBecomesTargetBit,
   kSMPCursorOverMailerMask      = 1<<kSMPCursorOverMailerBit,
   kSMPCreateCopyWindowMask      = 1<<kSMPCreateCopyWindowBit,
   kSMPDisposeCopyWindowMask     = 1<<kSMPDisposeCopyWindowBit
};

typedef unsigned long SMPMailerResult;

/* values of SMPMailerComponent*/

enum {
   kSMPOther         =  -1,
   kSMPFrom          =  32,
   kSMPTo            =  20,
   kSMPRegarding     =  22,
   kSMPSendDateTime  =  29,
   kSMPAttachments   =  26,
   kSMPAddressOMatic =  16
};

typedef unsigned long SMPMailerComponent;
```

```
enum {
   kSMPToAddress  =  kMailToBit,
   kSMPCCAddress  =  kMailCcBit,
   kSMPBCCAddress =  kMailBccBit
};

typedef MailAttributeID SMPAddressType;

enum {
   kSMPUndoCommand,
   kSMPCutCommand,
   kSMPCopyCommand,
   kSMPPasteCommand,
   kSMPClearCommand,
   kSMPSelectAllCommand
};

typedef unsigned short SMPEditCommand;

enum {
   kSMPUndoDisabled,
   kSMPAppMayUndo,
   kSMPMailerUndo
};

typedef unsigned short SMPUndoState;

/* SMPSendFormatMask: Bitfield indicating which combinations of formats are
included in, should be included in, or can be included in a letter. */

enum {
   kSMPNativeBit,
   kSMPImageBit,
   kSMPStandardInterchangeBit
};

/* values of SMPSendFormatMask */

enum {
   kSMPNativeMask               = 1<<kSMPNativeBit,
   kSMPImageMask                = 1<<kSMPImageBit,
   kSMPStandardInterchangeMask  = 1<<kSMPStandardInterchangeBit
};

typedef unsigned long SMPSendFormatMask;
```

```
/* pseudo-events passed to the client's filter proc for initialization and
cleanup */

enum {
   kSMPSendOptionsStart = -1,
   kSMPSendOptionsEnd   = -2
};

enum {
   kSMPSave,
   kSMPSaveAs,
   kSMPSaveACopy
};

typedef unsigned short SMPSaveType;

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

typedef unsigned short MailSegmentType;

/* values of MailBlockMode */
enum {
   kMailFromStart = 1,     /* offset calculated from start of block */
   kMailFromLEOB  = 2,     /* offset calculated from end of block */
   kMailFromMark  = 3      /* offset calculated from current mark */
};

typedef short MailBlockMode;

struct SMPRecipientDescriptor
{
   struct SMPRecipientDescriptor *next;      /* pointer to next element */
   OSErr                         result;     /* result code */
   OCEPackedRecipient            *recipient; /* packed recipient address */
   unsigned long                 size;       /* size of recipient address */
```

```
   MailRecipient                theAddress; /* unpacked recipient address */
   RecordID                     theRID;     /* record ID of recipient */
};

typedef struct SMPRecipientDescriptor SMPRecipientDescriptor;
typedef SMPRecipientDescriptor *SMPRecipientDescriptorPtr;

struct SMPEnclosureDescriptor
{
   struct SMPEnclosureDescriptor *next;         /* pointer to next element */
   OSErr                         result;        /* result code */
   FSSpec                        fileSpec;      /* file specifier of
                                                   enclosure */
   OSType                        fileCreator;   /* creator of enclosure */
   OSType                        fileType;      /* file type of enclosure */
};

typedef struct SMPEnclosureDescriptor SMPEnclosureDescriptor;
typedef SMPEnclosureDescriptor *SMPEnclosureDescriptorPtr;

struct LetterDescriptor {
   Boolean onDisk;
   union {
      FSSpec fileSpec;
      LetterSpec mailboxSpec;
   }u;
};

typedef struct LetterDescriptor LetterDescriptor;

struct SMPLetterPB
{
   OSErr                      result;        /* function result */
   RStringPtr                 subject;       /* subject of letter */
   AuthIdentity               senderIdentity;/* identity of sender */
   SMPRecipientDescriptorPtr  toList;        /* list of addressees */
   SMPRecipientDescriptorPtr  ccList;        /* list of cc addressees */
   SMPRecipientDescriptorPtr  bccList;       /* list of bcc addressees */
   ScriptCode                 script;        /* script code for language */
   Size                       textSize;      /* length of body data */
   Ptr                        textBuffer;    /* body of the letter */
   SMPPSendAs                 sendAs;        /* file, enclosure, or image */
   Byte                       padByte;       /* reserved */
   SMPEnclosureDescriptorPtr  enclosures;    /* files to be enclosed */
   SMPDrawImageProcPtr        drawImageProc; /* your imaging routine */
```

```
   long                     imageRefCon;   /* for your use */
   Boolean                  supportsColor; /* true for a color grafPort */
};

typedef struct SMPLetterPB SMPLetterPB;
typedef SMPLetterPB *SMPLetterPBPtr;

struct SMPCloseOptions {
   Boolean              moveToTrash;
   Boolean              addTag;
   RString32            tag;
};

typedef struct SMPCloseOptions SMPCloseOptions;
typedef SMPCloseOptions *SMPCloseOptionsPtr;

struct SMPMailerState {
   short                mailerCount;
   short                currentMailer;
   Point                upperLeft;
   Boolean              hasBeenReceived;
   Boolean              isTarget;
   Boolean              isExpanded;
   Boolean              canMoveToTrash;
   Boolean              canTag;
   Byte                 padByte2;
   unsigned long        changeCount;
   SMPMailerComponent   targetComponent;
   Boolean              canCut;
   Boolean              canCopy;
   Boolean              canPaste;
   Boolean              canClear;
   Boolean              canSelectAll;
   Byte                 padByte3;
   SMPUndoState         undoState;
   Str63                undoWhat;
};

typedef struct SMPMailerState SMPMailerState;

struct SMPSendOptions {
   Boolean     signWhenSent;
   IPMPriority priority;
};
```

```
typedef struct SMPSendOptions SMPSendOptions;
typedef SMPSendOptions *SMPSendOptionsPtr, **SMPSendOptionsHandle;

/* SMPSendFormat: Structure describing the format of a letter.  If
kSMPNativeMask bit is set, the whichNativeFormat field indicates which of the
client-defined formats to use. */

struct SMPSendFormat {
   SMPSendFormatMask whichFormats;
   short whichNativeFormat;         /* zero-based */
};

typedef struct SMPSendFormat SMPSendFormat;

struct LetterSpec
{
   unsigned long     spec[3];
};

struct SMPLetterInfo {
      OSType      letterCreator;
      OSType      letterType;
      RString32   subject;
      RString32   sender;
};

typedef struct SMPLetterInfo SMPLetterInfo;

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

typedef struct MailTime MailTime;

typedef  unsigned long  UTCTime;    /* seconds since 1/1/1904 */
typedef  long           UTCOffset;  /* correct for local time */

/* pointers to functions for application-defined callback functions */

typedef pascal void (*SMPDrawImageProcPtr)(long refcon, Boolean inColor);

typedef pascal WindowPtr (*FrontWindowProcPtr) (long clientData);

typedef pascal void (*PrepareMailerForDrawingProcPtr) (WindowPtr window,
                                                       long clientData);
```

```
typedef pascal Boolean (*SendOptionsFilterProc) (DialogPtr theDialog,
                                                 EventRecord* theEvent,
                                                 short itemHit,
                                                 long clientData);
```

Standard Mail Package Functions

Send-Letter Functions

```
pascal OSErr SMPSendLetter  (SMPLetterPBPtr theLetter);
pascal OSErr SMPNewPage     (OpenCPicParams *newHeader);
pascal OSErr SMPImageErr    (void);
pascal OSErr SMPResolveToRecipient
                            (PackedDSSpecPtr dsSpec,
                             SMPRecipientDescriptorPtr *recipientList,
                             AuthIdentity identity);
```

Providing Mailers in Your Windows

```
pascal OSErr SMPInitMailer  (long mailerVersion);
pascal OSErr SMPNewMailer   (WindowPtr window,
                             Point upperLeft,
                             Boolean canContract,
                             Boolean initiallyExpanded,
                             AuthIdentity identity,
                             const PrepareMailerForDrawingProcPtr
                              prepareMailerForDrawingCB,
                             long clientData);
pascal OSErr SMPGetDimensions
                            (short *width,
                             short *contractedHeight,
                             short *expandedHeight);
pascal OSErr SMPMailerForward
                            (WindowPtr window,
                             AuthIdentity from);
```

```
pascal OSErr SMPMailerReply
                            (WindowPtr originalLetter,
                             WindowPtr newLetter,
                             Boolean replyToAll,
                             Point upperLeft,
                             Boolean canContract,
                             Boolean initiallyExpanded,
                             AuthIdentity identity,
                             const PrepareMailerForDrawingProcPtr
                             prepareMailerForDrawingCB,
                             long clientData);
pascal OSErr SMPGetTabInfo  (SMPMailerComponent *firstTab,
                             SMPMailerComponent *lastTab);
pascal OSErr SMPBecomeTarget
                            (WindowPtr window,
                             Boolean becomeTarget,
                             SMPMailerComponent whichField);
pascal OSErr SMPExpandOrContract
                            (WindowPtr window,
                             Boolean expand);
pascal OSErr SMPMoveMailer  (WindowPtr window,
                             short dh,
                             short dv);
pascal OSErr SMPTagDialog   (WindowPtr window,
                             RString32 *theTag);
pascal OSErr SMPPrepareToClose
                            (WindowPtr window);
pascal OSErr SMPCloseOptionsDialog
                            (WindowPtr window,
                             SMPCloseOptionsPtr closeOptions);
pascal OSErr SMPDisposeMailer
                            (WindowPtr window,
                             SMPCloseOptionsPtr closeOptions);
```

Handling Events in Mailers

```
pascal OSErr SMPMailerEvent
                            (const EventRecord *event,
                             SMPMailerResult *whatHappened,
                             const FrontWindowProcPtr frontWindowCB,
                             long clientData);
pascal OSErr SMPMailerEditCommand
                            (WindowPtr window,
                             SMPEditCommand command,
                             SMPMailerResult *whatHappened);
```

```
pascal OSErr SMPGetMailerState
                            (windowPtr window,
                             SMPMailerState *itsState);
pascal OSErr SMPClearUndo   (WindowPtr window);
pascal OSErr SMPDrawMailer  (WindowPtr window);
```

Sending and Saving Mail

```
pascal OSErr SMPSendOptionsDialog
                            (WindowPtr window,
                             Str255 documentName,
                             StringPtr nativeFormatNames[],
                             unsigned short nameCount,
                             SMPSendFormatMask canSend,
                             SMPSendFormat *currentFormat,
                             SendOptionsFilterProc filterProc,
                             long clientData,
                             SMPSendFormat *shouldSend,
                             SMPSendOptionsPtr sendOptions);
pascal OSErr SMPContentChanged
                            (WindowPtr window);
pascal OSErr SMPBeginSave   (WindowPtr window,
                             const FSSpec *diskLetter,
                             OSType creator,
                             OSType filetype,
                             SMPSaveType saveType,
                             Boolean *mustAddContent);
pascal OSErr SMPEndSave     (WindowPtr window,
                             Boolean okToSave);
pascal OSErr SMPBeginSend   (WindowPtr window,
                             OSType creator,
                             OSType fileType,
                             SMPSendOptionsPtr sendOptions,
                             Boolean *mustAddContent);
pascal OSErr SMPPrepareToChange
                            (WindowPtr window);
pascal OSErr SMPEndSend     (WindowPtr window,
                             Boolean okToSend);
```

```
pascal OSErr SMPAddContent  (WindowPtr window,
                             MailSegmentType segmentType,
                             Boolean appendFlag,
                             void *buffer,
                             unsigned long bufferSize,
                             StScrpRec *textScrap,
                             Boolean startNewScript,
                             ScriptCode script);
pascal OSErr SMPImage       (WindowPtr window,
                             SMPDrawImageProcPtr drawImageProc,
                             long imageRefCon,
                             Boolean supportsColor);
pascal OSErr SMPAddMainEnclosure
                            (WindowPtr window,
                             const FSSpec *enclosure);
pascal OSErr SMPAddBlock    (WindowPtr window,
                             const OCECreatorType *blockType,
                             Boolean append,
                             void *buffer,
                             unsigned long bufferSize,
                             MailBlockMode mode,
                             unsigned long offset);
```

Reading Mail

```
pascal OSErr SMPGetLetterInfo
                            (LetterSpec *mailboxSpec,
                             SMPLetterInfo *info);
pascal OSErr SMPOpenLetter  (const LetterDescriptor *letter,
                             WindowPtr window,
                             Point upperLeft,
                             Boolean canContract,
                             Boolean initiallyExpanded,
                             const PrepareMailerForDrawingProcPtr
                              prepareMailerForDrawingCB,
                             long clientData);
pascal OSErr SMPGetNextLetter
                            (OSType *typesList,
                             short numTypes,
                             LetterDescriptor *adjacentLetter);
```

```
pascal OSErr SMPReadContent
                            (WindowPtr window,
                             MailSegmentMask segmentTypeMask,
                             void *buffer,
                             unsigned long bufferSize,
                             unsigned long *dataSize,
                             StScrpRec *textScrap,
                             ScriptCode *script,
                             MailSegmentType *segmentType,
                             Boolean *endOfScript,
                             Boolean *endOfSegment,
                             Boolean *endOfContent,
                             long *segmentLength,
                             long *segmentID);
pascal OSErr SMPGetFontNameFromLetter
                            (WindowPtr window,
                             short fontNum,
                             str255 fontName,
                             Boolean doneWithFontTable);
pascal OSErr SMPGetMainEnclosureFSSpec
                            (WindowPtr window,
                             FSSpec *enclosureDir);
pascal OSErr SMPEnumerateBlocks
                            (WindowPtr window,
                             unsigned short startIndex,
                             void *buffer,
                             unsigned long bufferSize,
                             unsigned long *dataSize,
                             unsigned short *nextIndex,
                             Boolean *more);
pascal OSErr SMPReadBlock   (WindowPtr window,
                             const OCECreatorType *blockType,
                             unsigned short blockIndex,
                             void *buffer,
                             unsigned long bufferSize,
                             unsigned long dataOffset,
                             unsigned long *dataSize,
                             Boolean *endOfBlock,
                             unsigned long *remaining);
```

Printing Mailers

```
pascal OSErr SMPPrepareCoverPages
                            (windowPtr window,
                             short *pageCount);
```

```
pascal OSErr SMPDrawNthCoverPage
                            (WindowPtr window,
                             short pageNumber,
                             Boolean doneDrawingCoverPages);
```

Getting and Setting Information in the Mailer

```
pascal OSErr SMPGetComponentSize
                            (WindowPtr window,
                             unsigned short whichMailer,
                             SMPMailerComponent whichField,
                             unsigned short *size);
pascal OSErr SMPGetComponentInfo
                            (WindowPtr window,
                             unsigned short whichMailer,
                             SMPMailerComponent whichField,
                             void *buffer);
pascal OSErr SMPGetListItemInfo
                            (WindowPtr window,
                             unsigned short whichMailer,
                             SMPMailerComponent whichField,
                             void *buffer,
                             unsigned long bufferLength,
                             unsigned short startItem,
                             unsigned short *itemCount,
                             unsigned short *nextItem,
                             Boolean *more);
pascal OSErr SMPSetSubject  (WindowPtr window,
                             const RString *text);
pascal OSErr SMPSetFromIdentity
                            (WindowPtr window,
                             AuthIdentity from);
pascal OSErr SMPAddAddress  (WindowPtr window,
                             SMPAddressType addrType,
                             OCEPackedRecipient *address);
pascal OSErr SMPAddAttachment
                            (WindowPtr window,
                             const FSSpec *attachment);
pascal OSErr SMPAttachDialog
                            (WindowPtr window);
```

Application-Defined Functions

```
pascal void MyPrepareMailerForDrawing
                            (WindowPtr window,
                             long clientData);
pascal void MyDrawImage     (long refcon, Boolean inColor);
pascal WindowPtr MyFrontWindowCB
                            (long clientData);
pascal Boolean MySendOptionsFilterProc
                            (DialogPtr theDialog,
                             EventRecord* theEvent,
                             short itemHit,
                             long clientData);
```

Pascal Summary

Constants

```
CONST
gestaltSMPMailerVersion       = 'malr';
gestaltSMPSPSendLetterVersion = 'spsl';
kSMPNativeFormatName          = 'natv';

typeLetterSpec                = 'lttr';

{ wildcard used for filtering letter types }

FilterAnyLetter               =  'ltr*';
FilterAppleLetterContent      =  'ltc*';
FilterImageContent            =  'lti*';

{ SMPPSendAs values.  You may add the following values together to
determine how the file is sent, but you may not set both
kSMPSendAsEnclosureMask and kSMPSendFileOnlyMask. }

kSMPSendAsEnclosureBit  = 0;     { appears as letter with enclosures }
kSMPSendFileOnlyBit     = 1;     { appears as a file in mailbox }
kSMPSendAsImageBit      = 2;     { letter includes image of content }
```

```
{ values of SMPPSendAs }
kSMPSendAsEnclosureMask    = $01;  {1<<kSMPSendAsEnclosureBit}
kSMPSendFileOnlyMask       = $02;  {1<<kSMPSendFileOnlyBit}
kSMPSendAsImageMask        = $04;  {1<<kSMPSendAsImageBit}

kSMPAppMustHandleEventBit     = 0;
kSMPAppShouldIgnoreEventBit   = 1;
kSMPContractedBit             = 2;
kSMPExpandedBit               = 3;
kSMPMailerBecomesTargetBit    = 4;
kSMPAppBecomesTargetBit       = 5;
kSMPCursorOverMailerBit       = 6;
kSMPCreateCopyWindowBit       = 7;
kSMPDisposeCopyWindowBit      = 8;

{ values of SMPMailerResult }
kSMPAppMustHandleEventMask    = $00000001; {1<<kSMPAppMustHandleEventBit}
kSMPAppShouldIgnoreEventMask  = $00000002; {1<<kSMPAppShouldIgnoreEventBit}
kSMPContractedMask            = $00000004; {1<<kSMPContractedBit}
kSMPExpandedMask              = $00000008; {1<<kSMPExpandedBit}
kSMPMailerBecomesTargetMask   = $00000010; {1<<kSMPMailerBecomesTargetBit}
kSMPAppBecomesTargetMask      = $00000020; {1<<kSMPAppBecomesTargetBit}
kSMPCursorOverMailerMask      = $00000040; {1<<kSMPCursorOverMailerBit}
kSMPCreateCopyWindowMask      = $00000080; {1<<kSMPCreateCopyWindowBit}
kSMPDisposeCopyWindowMask     = $00000100; {1<<kSMPDisposeCopyWindowBit}

{ values of SMPMailerComponent }
kSMPOther            =  -1;
kSMPFrom             =  32;
kSMPTo               =  20;
kSMPRegarding        =  22;
kSMPSendDateTime     =  29;
kSMPAttachments      =  26;
kSMPAddressOMatic    =  16;

kSMPToAddress  =  kMailToBit;
kSMPCCAddress  =  kMailCcBit;
kSMPBCCAddress =  kMailBccBit;

kSMPUndoCommand = 0;
kSMPCutCommand = 1;
kSMPCopyCommand = 2;
```

```
kSMPPasteCommand = 3;
kSMPClearCommand = 4;
kSMPSelectAllCommand = 5;

kSMPUndoDisabled = 0;
kSMPAppMayUndo = 1;
kSMPMailerUndo = 2;

{ SMPSendFormatMask: Bitfield indicating which combinations of formats are
included in, should be included in, or can be included in a letter. }

kSMPNativeBit = 0;
kSMPImageBit = 1;
kSMPStandardInterchangeBit = 2;

{ values of SMPSendFormatMask }
kSMPNativeMask                = $00000001;  {1<<kSMPNativeBit}
kSMPImageMask                 = $00000002;  {1<<kSMPImageBit}
kSMPStandardInterchangeMask   = $00000004;  {1<<kSMPStandardInterchangeBit}

{ pseudo-events passed to the client's filter proc for initialization and
cleanup }
kSMPSendOptionsStart= -1;
kSMPSendOptionsEnd= -2;

kSMPSave = 0;
kSMPSaveAs = 1;
kSMPSaveACopy = 2;

{ values of MailSegmentType }
kMailInvalidSegmentType= 0;
kMailTextSegmentType= 1;
kMailPictSegmentType= 2;
kMailSoundSegmentType= 3;
kMailStyledTextSegmentType= 4;
kMailMovieSegmentType= 5;

{ values of MailBlockMode }
kMailFromStart= 1;{ offset calculated from start of block }
kMailFromLEOB= 2; { offset calculated from end of block }
kMailFromMark= 3; { offset calculated from the current mark }
```

Data Types

```
TYPE
SMPSendFormatMask = LONGINT;

SMPSaveType = INTEGER;

SMPMailerResult = LONGINT;

SMPMailerComponent = LONGINT;

SMPAddressType = MailAttributeID;

SMPEditCommand = INTEGER;

SMPUndoState = INTEGER;

SMPPSendAs = Byte;

MailBlockMode = INTEGER;

MailSegmentType = INTEGER;

SMPRecipientDescriptor = RECORD
   next:          ^SMPRecipientDescriptor;   { pointer to next element }
   result:        OSErr;                     { result code }
   recipient:     ^OCEPackedRecipient;       { packed recipient address }
   size:          LONGINT;                   { size of recipient address }
   theAddress:    MailRecipient;             { unpacked recipient address }
   theRID:        RecordID;                  { record ID of recipient }
   END;

SMPRecipientDescriptorPtr = ^SMPRecipientDescriptor;

SMPEnclosureDescriptor = RECORD
   next:          ^SMPEnclosureDescriptor;   { pointer to next element }
   result:        OSErr;                     { result code }
   fileSpec:      FSSpec;                    { file specifier of enclosure }
   fileCreator:   OSType;                    { creator of enclosure }
   fileType:      OSType;                    { file type of enclosure }
   END;

SMPEnclosureDescriptorPtr = ^SMPEnclosureDescriptor;
```

```
LetterDescriptor = RECORD
   onDisk: BOOLEAN;
   CASE INTEGER OF
      1: (fileSpec: FSSpec);
      2: (mailboxSpec: LetterSpec);
   END;

SMPLetterPB = PACKED RECORD
   result:        OSErr;                    { function result }
   subject:       RStringPtr;               { subject of letter }
   senderIdentity:AuthIdentity;             { identity of sender }
   toList:        SMPRecipientDescriptorPtr; { list of addressees }
   ccList:        SMPRecipientDescriptorPtr; { list of cc addressees }
   bccList:       SMPRecipientDescriptorPtr; { list of bcc addressees }
   script:        ScriptCode;               { script code for language }
   textSize:      Size;                     { length of body data }
   textBuffer:    Ptr;                      { body of the letter }
   sendAs:        SMPPSendAs;               { letter,enclosure, or image }
   padByte:       Byte;                     { reserved }
   enclosures:    SMPEnclosureDescriptorPtr; { files to be enclosed }
   drawImageProc: SMPDrawImageProcPtr;      { your imaging routine }
   imageRefCon:   LONGINT;                  { for your use }
   supportsColor: BOOLEAN;                  { true for a color grafPort }
   END;

SMPLetterPBPtr = ^SMPLetterPB;

SMPCloseOptions = RECORD
   moveToTrash: BOOLEAN;
   addTag: BOOLEAN;
   tag: RString32;
   END;

SMPCloseOptionsPtr = ^SMPCloseOptions;

SMPMailerState = RECORD
   mailerCount: INTEGER;
   currentMailer: INTEGER;
   upperLeft: Point;
   hasBeenReceived: BOOLEAN;
   isTarget: BOOLEAN;
   isExpanded: BOOLEAN;
   canMoveToTrash: BOOLEAN;
   canTag: BOOLEAN;
```

```
   {padByte2: Byte;}
   changeCount: LONGINT;
   targetComponent: SMPMailerComponent;
   canCut: BOOLEAN;
   canCopy: BOOLEAN;
   canPaste: BOOLEAN;
   canClear: BOOLEAN;
   canSelectAll: BOOLEAN;
   {padByte3: Byte;}
   undoState: SMPUndoState;
   undoWhat: Str63;
   END;

SMPSendOptions = RECORD
   signWhenSent: BOOLEAN;
   priority: IPMPriority;
   END;

SMPSendOptionsPtr = ^SMPSendOptions;
SMPSendOptionsHandle = ^SMPSendOptionsPtr;

{ SMPSendFormat: Structure describing the format of a letter.  If
kSMPNativeMask bit is set, the whichNativeFormat field indicates which of the
client-defined formats to use. }

SMPSendFormat = RECORD
   whichFormats: SMPSendFormatMask;
   whichNativeFormat: INTEGER;{ 0 based }
   END;

LetterSpec = RECORD
   spec: ARRAY[1..3] OF LONGINT;
   END;

SMPLetterInfo = RECORD
   letterCreator: OSType;
   letterType: OSType;
   subject: RString32;
   sender: RString32;
   END;
```

```
MailTime = RECORD
   time: UTCTime;    { current UTC (GMT) time }
   offset: UTCOffset;{ in seconds from GMT (positive is east) }
   END;

UTCTime = LONGINT;   { seconds since 1/1/1904 }
UTCOffset = LONGINT; { correct for local time }

{ pointers to functions for application-defined callback functions }

SMPDrawImageProcPtr = ProcPtr;
   { FUNCTION SMPDrawImageProcPtr(refcon: LONGINT; inColor: BOOLEAN): void;}

FrontWindowProcPtr = ProcPtr;
   { FUNCTION FrontWindowProcPtr(clientData: LONGINT): WindowPtr;}

PrepareMailerForDrawingProcPtr = ProcPtr;
   { FUNCTION PrepareMailerForDrawingProcPtr(window: WindowPtr;
                                        clientData: LONGINT): void;}

SendOptionsFilterProc = ProcPtr;
   { FUNCTION SendOptionsFilterProc(theDialog: DialogPtr;
                                VAR theEvent: EventRecord;
                                itemHit: INTEGER;
                                clientData: LONGINT): BOOLEAN;}
```

Standard Mail Package Functions

Send-Letter Functions

```
FUNCTION SMPSendLetter      (theLetter: SMPLetterPBPtr): OSErr;
FUNCTION SMPNewPage         (VAR newHeader: OpenCPicParams): OSErr;
FUNCTION SMPImageErr: OSErr;
FUNCTION SMPResolveToRecipient
                            (dsSpec: PackedDSSpecPtr;
                             VAR recipientList: SMPRecipientDescriptorPtr;
                             identity: AuthIdentity): OSErr;
```

Providing Mailers in Your Windows

```
FUNCTION SMPInitMailer      (mailerVersion: LONGINT): OSErr;
FUNCTION SMPNewMailer       (window: WindowPtr; upperLeft: Point;
                             canContract: BOOLEAN;
                             initiallyExpanded: BOOLEAN;
                             identity: AuthIdentity;
                             prepareMailerForDrawingCB:
                             PrepareMailerForDrawingProcPtr;
                             clientData: LONGINT): OSErr;
FUNCTION SMPGetDimensions   (VAR width: INTEGER; VAR contractedHeight:
                             INTEGER; VAR expandedHeight: INTEGER): OSErr;
FUNCTION SMPMailerForward   (window: WindowPtr; from: AuthIdentity): OSErr;
FUNCTION SMPMailerReply     (originalLetter: WindowPtr; newLetter:
                             WindowPtr; replyToAll: BOOLEAN; upperLeft:
                             Point; canContract: BOOLEAN;
                             initiallyExpanded: BOOLEAN;
                             identity: AuthIdentity;
                             prepareMailerForDrawingCB:
                             PrepareMailerForDrawingProcPtr;
                             clientData: LONGINT): OSErr;
FUNCTION SMPGetTabInfo      (VAR firstTab: SMPMailerComponent;
                             VAR lastTab: SMPMailerComponent): OSErr;
FUNCTION SMPBecomeTarget    (window: WindowPtr; becomeTarget: BOOLEAN;
                             whichField: SMPMailerComponent): OSErr;
FUNCTION SMPExpandOrContract
                            (window: WindowPtr; expand: BOOLEAN): OSErr;
FUNCTION SMPMoveMailer      (window: WindowPtr; dh: INTEGER; dv: INTEGER):
                             OSErr;
FUNCTION SMPTagDialog       (window: WindowPtr; theTag: RString32Ptr):
                             OSErr;
FUNCTION SMPPrepareToClose  (window: WindowPtr): OSErr;
FUNCTION SMPCloseOptionsDialog
                            (window: WindowPtr;
                             closeOptions: SMPCloseOptionsPtr): OSErr;
FUNCTION SMPDisposeMailer   (window: WindowPtr;
                             closeOptions: SMPCloseOptionsPtr): OSErr;
```

Handling Events in Mailers

```
FUNCTION SMPMailerEvent      (event: EventRecord;
                              VAR whatHappened: SMPMailerResult;
                              frontWindowCB: FrontWindowProcPtr;
                              clientData: LONGINT): OSErr;
FUNCTION SMPMailerEditCommand
                             (window: WindowPtr; command: SMPEditCommand;
                              VAR whatHappened: SMPMailerResult): OSErr;
FUNCTION SMPGetMailerState   (window: WindowPtr; VAR itsState:
                              SMPMailerState): OSErr;
FUNCTION SMPClearUndo        (window: WindowPtr): OSErr;
FUNCTION SMPDrawMailer       (window: WindowPtr): OSErr;
```

Sending and Saving Mail

```
FUNCTION SMPSendOptionsDialog
                             (window: WindowPtr; documentName: Str255;
                              VAR nativeFormatNames: StringPtr;
                              nameCount: INTEGER;
                              canSend: SMPSendFormatMask;
                              VAR currentFormat: SMPSendFormat;
                              filterProc: SendOptionsFilterProc;
                              clientData: LONGINT;
                              VAR shouldSend: SMPSendFormat;
                              sendOptions: SMPSendOptionsPtr): OSErr;
FUNCTION SMPContentChanged
                             (window: WindowPtr): OSErr;
FUNCTION SMPBeginSave        (window: WindowPtr; diskLetter: FSSpec;
                              creator: OSType; fileType: OSType;
                              saveType: SMPSaveType;
                              VAR mustAddContent: BOOLEAN): OSErr;
FUNCTION SMPEndSave          (window: WindowPtr; okToSave: BOOLEAN): OSErr;
FUNCTION SMPBeginSend        (window: WindowPtr; creator: OSType; fileType:
                              OSType; sendOptions: SMPSendOptionsPtr;
                              VAR mustAddContent: BOOLEAN): OSErr;
FUNCTION SMPPrepareToChange
                             (window: WindowPtr): OSErr;
FUNCTION SMPEndSend          (window: WindowPtr; okToSend: BOOLEAN): OSErr;
```

```
FUNCTION SMPAddContent      (window: WindowPtr; segmentType:
                             MailSegmentType; appendFlag: BOOLEAN; buffer:
                             UNIV Ptr; bufferSize: LONGINT; textScrap:
                             StScrpPtr; startNewScript: BOOLEAN; script:
                             ScriptCode): OSErr;
FUNCTION SMPImage           (window: WindowPtr; drawImageProc:
                             SMPDrawImageProcPtr; imageRefCon: LONGINT;
                             supportsColor: BOOLEAN): OSErr;
FUNCTION SMPAddMainEnclosure
                            (window: WindowPtr; enclosure: FSSpec): OSErr;
FUNCTION SMPAddBlock        (window: WindowPtr; blockType: OCECreatorType;
                             append: BOOLEAN; buffer: UNIV Ptr;
                             bufferSize: LONGINT; mode: MailBlockMode;
                             offset: LONGINT): OSErr;
```

Reading Mail

```
FUNCTION SMPGetLetterInfo   (VAR mailboxSpec: LetterSpec;
                             VAR info: SMPLetterInfo): OSErr;
FUNCTION SMPOpenLetter      (letter: LetterDescriptor; window: WindowPtr;
                             upperLeft: Point; canContract: BOOLEAN;
                             initiallyExpanded: BOOLEAN;
                             prepareMailerForDrawingCB:
                             PrepareMailerForDrawingProcPtr;
                             clientData: LONGINT): OSErr;
FUNCTION SMPGetNextLetter   (VAR typesList: OSType; numTypes: INTEGER;
                             VAR adjacentLetter: LetterDescriptor): OSErr;
FUNCTION SMPReadContent     (window: WindowPtr; segmentTypeMask:
                             MailSegmentMask; buffer: UNIV Ptr; bufferSize:
                             LONGINT;
                             VAR dataSize: LONGINT;
                             VAR textScrap: StScrpRec;
                             VAR script: ScriptCode;
                             VAR segmentType: MailSegmentType;
                             VAR endOfScript: BOOLEAN;
                             VAR endOfSegment: BOOLEAN;
                             VAR endOfContent: BOOLEAN;
                             VAR segmentLength: LONGINT;
                             VAR segmentID: LONGINT): OSErr;
FUNCTION SMPGetFontNameFromLetter
                            (window: WindowPtr; fontNum: INTEGER; fontName:
                             Str255; doneWithFontTable: BOOLEAN): OSErr;
```

```
FUNCTION SMPGetMainEnclosureFSSpec
                            (window: WindowPtr;
                             VAR enclosureDir: FSSpec): OSErr;
FUNCTION SMPEnumerateBlocks (window: WindowPtr; startIndex: INTEGER;
                             buffer: UNIV Ptr; bufferSize: LONGINT;
                             VAR dataSize: LONGINT; VAR nextIndex: INTEGER;
                             VAR more: BOOLEAN): OSErr;
FUNCTION SMPReadBlock       (window: WindowPtr; blockType: OCECreatorType;
                             blockIndex: INTEGER; buffer: UNIV Ptr;
                             bufferSize: LONGINT; dataOffset: LONGINT;
                             VAR dataSize: LONGINT; VAR endOfBlock:
                             BOOLEAN; VAR remaining: LONGINT): OSErr;
```

Printing Mailers

```
FUNCTION SMPPrepareCoverPages
                            (window: WindowPtr; VAR pageCount: INTEGER):
                             OSErr;
FUNCTION SMPDrawNthCoverPage
                            (window: WindowPtr; pageNumber: INTEGER;
                             doneDrawingCoverPages: BOOLEAN): OSErr;
```

Getting and Setting Information in the Mailer

```
FUNCTION SMPGetComponentSize
                            (window: WindowPtr; whichMailer: INTEGER;
                             whichField: SMPMailerComponent;
                             VAR size: INTEGER): OSErr;
FUNCTION SMPGetComponentInfo
                            (window: WindowPtr; whichMailer: INTEGER;
                             whichField: SMPMailerComponent;
                             buffer: UNIV Ptr): OSErr;
FUNCTION SMPGetListItemInfo (window: WindowPtr; whichMailer: INTEGER;
                             whichField: SMPMailerComponent;
                             buffer: UNIV Ptr; bufferLength: LONGINT;
                             startItem: INTEGER; VAR itemCount: INTEGER;
                             VAR nextItem: INTEGER; VAR more: BOOLEAN):
                             OSErr;
FUNCTION SMPSetSubject      (window: WindowPtr; text: RString): OSErr;
FUNCTION SMPSetFromIdentity
                            (window: WindowPtr; from: AuthIdentity): OSErr;
FUNCTION SMPAddAddress      (window: WindowPtr; addrType: SMPAddressType;
                             address: OCEPackedRecipientPtr): OSErr;
FUNCTION SMPAddAttachment   (window: WindowPtr; attachment: FSSpec): OSErr;
FUNCTION SMPAttachDialog    (window: WindowPtr): OSErr;
```

Application-Defined Functions

```
FUNCTION MyPrepareMailerForDrawing
                            (window: WindowPtr; clientData: LONGINT): void;
FUNCTION MyDrawImage        (refcon: LONGINT; inColor: BOOLEAN): void;
FUNCTION MyFrontWindowCB    (clientData: LONGINT): WindowPtr;
FUNCTION MySendOptionsFilterProc
                            (theDialog: DialogPtr;
                             VAR theEvent: EventRecord; itemHit: INTEGER;
                             clientData: LONGINT): BOOLEAN;
```

Assembly-Language Summary

Trap Macros

Trap Requiring Routine Selectors

$AA5D

Selector	Count	Routine
$01F4	$0002	SMPSendLetter
$044C	$0006	SMPResolveToRecipient
$0834	$0002	SMPNewPage
$0835	$0000	SMPImageErr
$125C	$0006	SMPGetDimensions
$125D	$000C	SMPNewMailer
$125E	$0004	SMPDisposeMailer
$125F	$0008	SMPMailerEvent
$1260	$0005	SMPMailerEditCommand
$1261	$0004	SMPMailerForward
$1262	$000F	SMPMailerReply
$1263	$0004	SMPGetMailerState
$1264	$0004	SMPPrepareCoverPages
$1265	$0004	SMPDrawNthCoverPage
$1266	$000B	SMPBeginSave
$1267	$000A	SMPBeginSend
$1268	$000C	SMPOpenLetter
$1269	$0002	SMPDrawMailer

continued

Selector	Count	Routine
$126A	$0004	SMPMoveMailer
$126B	$0004	SMPSetSubject
$126C	$0004	SMPSetFromIdentity
$126D	$0005	SMPAddAddress
$126E	$0004	SMPAddAttachment
$126F	$0002	SMPContentChanged
$1270	$0002	SMPEndSave
$1271	$0002	SMPEndSend
$1272	$0003	SMPExpandOrContract
$1273	$0005	SMPBecomeTarget
$1274	$0004	SMPGetTabInfo
$1275	$0002	SMPClearUndo
$1276	$0002	SMPAttachDialog
$1277	$0007	SMPGetComponentSize
$1278	$0007	SMPGetComponentInfo
$1279	$0010	SMPGetListItemInfo
$127A	$000D	SMPAddContent
$127B	$0019	SMPReadContent
$127C	$0006	SMPGetFontNameFromLetter
$127D	$0004	SMPAddMainEnclosure
$127E	$0004	SMPGetMainEnclosureFSSpec
$127F	$000C	SMPAddBlock
$1280	$000C	SMPReadBlock
$1281	$000D	SMPEnumerateBlocks
$1282	$0002	SMPImage
$1285	$0002	SMPInitMailer
$1286	$0008	SMPGetNextLetter
$1287	$0002	SMPPrepareToClose
$1288	$0004	SMPCloseOptionsDialog
$1289	$0002	SMPPrepareToChange
$128A	$0004	SMPGetLetterInfo
$128B	$0004	SMPTagDialog
$1388	$0013	SMPSendOptionsDialog

Result Codes

The allocated range of result codes for the Standard Mail Package is –1900 through –1949. Routines may also return standard Macintosh result codes such as `noErr` 0 (no error) and `fnfErr` –43 (file not found).

`kSMPNotEnoughMemoryForAllRecips`	–1900	Too many recipients in mailer
`kSMPCopyInProgress`	–1901	Enclosure being copied to mailer
`kSMPMailerNotInitialized`	–1902	Mailer has not been initialized
`kSMPShouldNotAddContent`	–1903	You cannot add content to letter
`kSMPMailboxNotFound`	–1904	Cannot find mailbox
`kSMPNoNextLetter`	–1905	There is no next letter in In Tray
`kSMPHasOpenAttachments`	–1906	One or more enclosures are open
`kSMPFinderNotRunning`	–1907	The Finder is not running
`kSMPCommandDisabled`	–1908	Requested command unavailable
`kSMPNoMailerInWindow`	–1909	No mailer is in specified window
`kSMPNoSuchAddress`	–1910	Requested address not found
`kSMPMailerAlreadyInWindow`	–1911	A mailer was previously allocated
`kSMPMailerUneditable`	–1912	Mailer cannot be edited
`kSMPNoMatchingBegin`	–1913	End function called without begin
`kSMPCannotSendReceivedLetter`	–1914	Letter is received; cannot be sent
`kSMPIllegalForDraftLetter`	–1915	Operation cannot be completed
`kSMPMailerCannotExpandOrContract`	–1916	Mailer created with `canContract` `false`
`kSMPMailerAlreadyExpandedOrContracted`	–1917	Mailer is already in requested state
`kSMPIllegalComponent`	–1918	Bad field name parameter
`kSMPMailerAlreadyNotTarget`	–1919	This mailer is not the target
`kSMPComponentIsAlreadyTarget`	–1920	The selected field is the target
`kSMPRecordDoesNotContainAddress`	–1921	Address is not in this record
`kSMPAddressAlreadyInList`	–1922	Specified address is in Recipients field
`kSMPIllegalSendFormats`	–1923	Format is not in `canSend` parameter
`kSMPInvalidAddressString`	–1924	Address string is invalid
`kSMPSubjectTooBig`	–1925	Subject string exceeds 127 characters
`kSMPParamCountErr`	–1926	Enclosure count should be 1
`kSMPTooManyPages`	–1927	Image is more than 127 pages
`kSMPTooManyEnclosures`	–1928	More than 50 total files and folders

CHAPTER 4

Standard Catalog Package

Contents

This chapter describes the AOCE Standard Catalog Package. The AOCE Standard Catalog Package provides a high-level interface that makes it easy for you to add catalog-browsing, authentication, and alias-resolution services to your applications.

This chapter assumes you are familiar with the nature and use of AOCE catalogs and authentication services.

If you want to design and implement your own interface to AOCE catalogs, see the chapter "Catalog Manager" in this book.

About the Standard Catalog Package

The AOCE Standard Catalog Package provides a high-level interface to the AOCE Catalog Manager. It makes it easy for you to add catalog-browsing and record-selection services to your application.

The Standard Catalog Package provides functions that

- Display a dialog box that allows users to enter their password for their authentication identity.
- Display and return information from a Catalog-Browsing panel that you can place in your window. The Catalog-Browsing panel lets users browse catalogs and select records.
- Display and return information from a Find panel that you can place in your window. The Find panel lets users search catalogs for a record if they know all or part of the record's name.
- Resolve aliases of records, catalogs, and other objects in the AOCE catalog system.
- Return icons for records, catalogs, and other AOCE components.
- Return lists of categories of catalog items available on a system (such as printers or users) and of types of items in these categories (such as LaserWriter and ImageWriter printers).

The Standard Catalog Package is a client of the AOCE Catalog and Authentication Managers. You do not have to call the underlying AOCE services directly to add catalog services to your application.

Finding and Selecting Records

The Standard Catalog package provides two standard interfaces for finding and selecting records in catalogs: a Catalog-Browsing panel and a Find panel. Catalog panels are described in "Creating, Displaying, and Disposing of a Catalog-Browsing Panel," beginning on page 4-29, and "Handling Events in a Catalog-Browsing Panel," beginning on page 4-51. Find panels are described in "Creating, Displaying, and Disposing of a Find Panel," beginning on page 4-61, and "Handling Events in a Find Panel," beginning on page 4-75. You can place a panel in any window you wish and provide menu items and dialog-box controls that make it easier for the user to browse or search for catalogs.

Figure 4-1 shows a Catalog-Browsing panel in use in an application's window, and Figure 4-2 shows a Find panel as it appears in a window of the AppleMail application. It is important to note that the use of such panels is not restricted to looking up addresses; you can use these interfaces to allow users to browse and search AOCE catalogs no matter what their content. For example, if you use an AOCE catalog to store information about routing forms (who should sign the form, the sequence in which to route it, and so forth), you can use the panels to help the user fill in and route a particular form.

Figure 4-1 A Catalog-Browsing panel in an application window

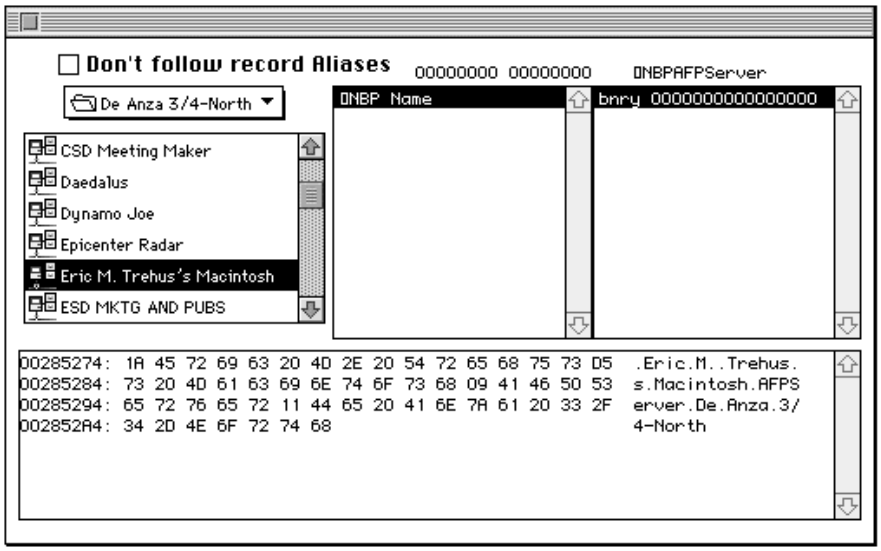
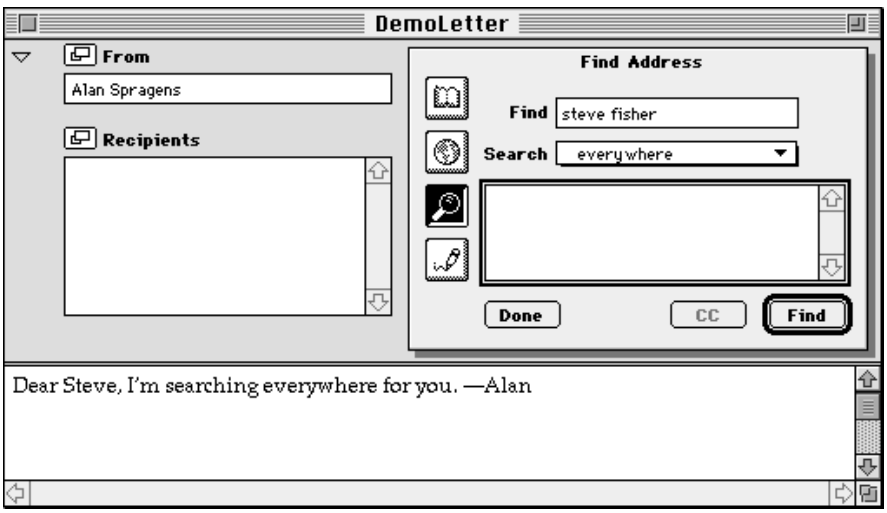


Figure 4-2 A Find panel in an application window



Using the Standard Catalog Package

This section describes how to use Standard Catalog Package routines to determine if AOCE is available, to authenticate a user, to create a Catalog-Browsing panel, and to handle Catalog-Browsing panel events. It also describes how to create and dispose of a Find panel.

Testing for the Presence of the Standard Catalog Package

Before using Standard Catalog Package functions in your application, you must use the Gestalt Manager to ensure that the system on which your application is running supports AOCE and the Standard Catalog Package.

To verify that the AOCE Collaboration toolbox is available, call 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" of *Inside Macintosh: Operating System Utilities*.

To determine the version of the Standard Catalog Package that is available, call the `Gestalt` function with the selector `gestaltSDPStandardDirectoryVersion`. The function returns the version number of the Standard Catalog Package in the low-order word of the `response` parameter. For example, a value of 0x0101 indicates version 1.0.1. If the Standard Catalog Package is not present and available, the `Gestalt` function returns 0 for the version number. Similarly, to determine the version of the Find panel, use the selector `gestaltSDPFindVersion`. To determine the version of the prompt-for-identity dialog box, use the selector `gestaltSDPPromptVersion`.

Listing 4-1 shows a function that checks for the availability of AOCE routines and returns `true` only if the Standard Catalog Package is installed and available.

Listing 4-1 Testing for the Standard Catalog Package

```
Boolean MyTestForStandardCatalog(void)
{
   OSErr    err;
   long     response;

   err = Gestalt(gestaltSDPStandardDirectoryVersion, &response);
   if ((err != noErr) || (response == 0))
      return false;
```

```
    return true;
}
```

Creating an Authentication Identity

The PowerTalk Key Chain lets the owner or principal user of a Macintosh computer enter a name, password, and other identifying information for PowerShare servers and whatever access modules are installed. The PowerTalk software protects that information with a single password, referred to as the **Key Chain Access Code**. The Authentication Manager takes the name and password set in the Key Chain and issues the computer a **local identity**. The AOCE toolbox can then gain access to the PowerShare server and access module services without requiring the user to log on again or enter another password.

Before the first time you send a message or let the user browse catalogs, you must provide identification to prove that the caller is an authorized user of the system. The `SDPPromptForID` function provides two versions of a dialog box that allows the user to identify himself or herself as one of the authorized users of the system (Figure 4-3). The simpler version of this dialog box allows the user to enter only the Key Chain Access Code. The function then returns the local identity to your application. The other version of this dialog box (also shown in Figure 4-3) allows the user to select a catalog and enter a name and password. The name must correspond to a record in the selected catalog. The function then verifies that the user has entered the correct access code and returns a **specific identity**. If the user logs on as a guest, the function returns 0 for the identity.

The “More Choices” version of the dialog box provides radio buttons corresponding to the three possibilities: Guest, PowerShare account (specific identity), or Key Chain Access Code (local identity). You can specify which of these options are enabled. You can also specify a catalog or a user or group record to be displayed initially in the dialog box, and you can restrict the user to this initially displayed selection.

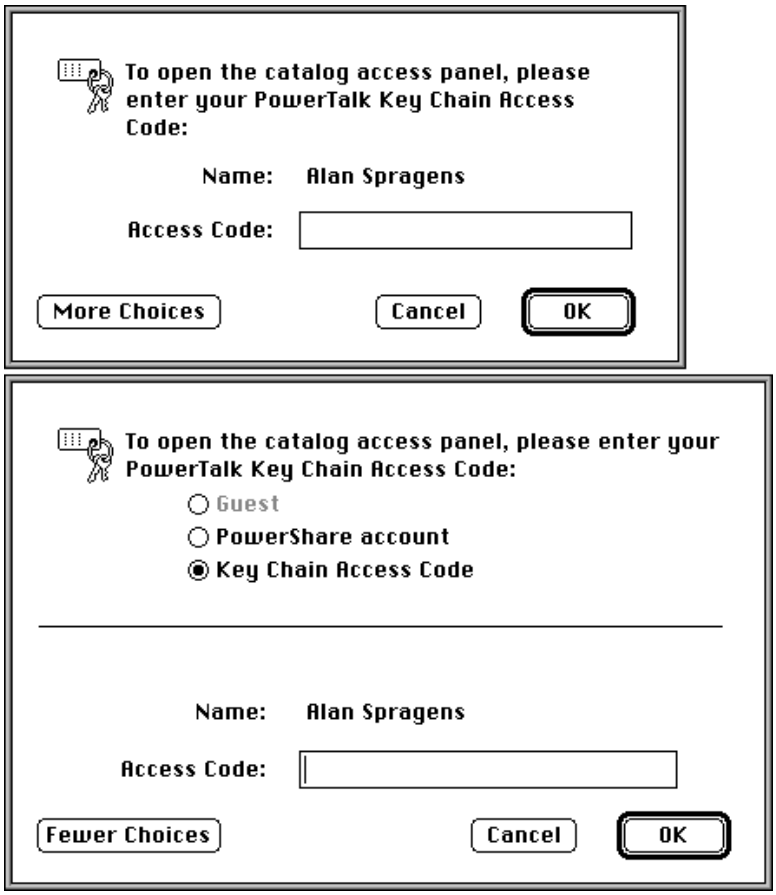
You must provide an identity when you call many of the Standard Catalog Package functions and most of the functions in the Catalog and Authentication Managers.

Note

Local AOCE resources such as personal messaging service access modules (MSAMs) are not available to a user who logs on with a specific identity or as a guest. u

If the user has never set up a local identity or selected any AOCE catalogs, the `SDPPromptForID` function prompts the user to make the appropriate choices.

Figure 4-3 Authentication dialog box



Before using Standard Catalog Package services, your application must determine the user's authentication identity. To discover if the user has previously been authenticated, without prompting the user, call the `AuthGetLocalIdentity` function, described in the "Authentication Manager" chapter of this book. If that function returns an error, then call the `SDPPromptForID` function to allow the user to unlock the local identity, set one up if none exists, or log on as a specific identity or guest.

The routine shown in Listing 4-2 first checks for an existing local identity. If none has been set up (if, for example, the system has just been started) or if the PowerTalk Key Chain is locked, the routine calls `SDPPromptForID`. The resulting authentication identity is stored by the application in the `AuthIdentity` pointer parameter `userIdentity`. If the local identity is already set up, it is returned in the authentication parameter block (type `AuthParamBlock`).

Listing 4-2 Getting an authentication identity

```
OSErr MyGetUserIdentity(AuthIdentity *userIdentity)
{
   SDPIdentityKind   selectedKind;
   AuthParamBlock    theAuthParamBlock;
   OSErr             status;

   status = AuthGetLocalIdentity(&theAuthParamBlock, FALSE);
   if (status == noErr) { /* Local identity is already set up. */
      *userIdentity =
         theAuthParamBlock.getLocalIdentityPB.theLocalIdentity;
   }
   else if (status == kOCELocalAuthenticationFail) {
      status = SDPPromptForID(
               userIdentity,  /* AuthIdentity *id */
               NULL,          /* default guest prompt */
               NULL,          /* default specific prompt */
               NULL,          /* default local prompt */
               NULL,          /* RString *recordType */
               (              /* SDPIdentityKind permittedKinds */
                  kSDPLocalIdentityMask | /* local identity or */
                  kSDPSpecificIdentityMask /* specific identity */
               ),
               &selectedKind, /* SDPIdentityKind *selectedKind */
               NULL,          /* RecordID *loginFilter */
               0              /* SDPLoginFilterKind filterKind */
            );
   }
   else {
      /* could check for (status == kOCESetupRequired) */
   }
   return status;
}
```

Creating a Catalog-Browsing Panel

This section illustrates how to create a Catalog-Browsing panel. The Catalog-Browsing panel provides a scrolling list and pop-up menu that you can place in any window. This list shows AOCE catalogs and their contained nodes and records. It also shows hierarchical file system (HFS) volumes, catalogs, and files so that the user can find AOCE information cards, aliases, and personal catalogs. The panel allows users to examine the contents of AOCE catalogs and to select a record in a catalog. Figure 4-4 shows a Catalog-Browsing panel, and Figure 4-1 on page 4-4 shows a panel in use in an

application window. Note that the font, size, and style of the characters in the Catalog-Browsing panel are those you set for the window before you call a routine to create a panel.

Figure 4-4 A Catalog-Browsing panel

In addition to the Catalog-Browsing panel, you can display a Personal-Catalog panel. This panel is identical to the Catalog-Browsing panel except that it displays only the contents of the user's default personal catalog. The Personal-Catalog panel looks just like the Catalog-Browsing panel in Figure 4-4 except that it does not include a pop-up menu.

Listing 4-3 illustrates the use of the `SDPNewPanel` function to create a new panel.

Listing 4-3 Using the `SDPNewPanel` function to create a new panel

```
OSErr MyCreateNewPanel(SDPPanelHandle *newPanel,
                       WindowPtr      theWindow,
                       Rect           *bounds,
                       AuthIdentity   identity)
{
   PackedRStringListHandle types = NULL;
   unsigned short          typeCount;
   RStringPtr              *typeList = NULL;
   DirEnumChoices          enumFlags;
   DirMatchWith            matchTypeHow;
   RStringPtr              *myCategory;
   Boolean                 typesIsResource = false;
   OSErr                   result;

   matchTypeHow = kExactMatch;
   enumFlags = kEnumDistinguishedNameMask + kEnumAliasMask + kEnumDNodeMask;

   myCategory = (RStringPtr *)GetResource('rstr',kMyCategory);
   result = !myCategory?resNotFound:ResError();

   if(result)
   {
```

```
   goto UseDefaultTypeList;
}
if(!result)
{
   result = SDPGetCategoryTypes(*myCategory ,&types);
   if(result)
   {
      /* Get default type list from a resource. */
      UseDefaultTypeList:

      types =(PackedRStringListHandle)GetResource('rtyp',kTypeListID);
      result = !types?resNotFound:ResError();
      DisposHandle((Handle)myCategory);
      typesIsResource = true;
   }
}

if(!result)
{
   typeCount = OCEDNodeNameCount(*types);
   typeCount++; /* Make space for DNode record type. */
   typeList = NewPtr(typeCount * sizeof(RStringPtr));
   OCEUnpackPathName(*types,typeList,typeCount);

   /* Allow aliases to DNodes to be shown by adding the following type.*/
   typeList[typeCount-1] = OCEGetIndRecordType(kDNodeRecTypeNum);
}

if(!result)
{
   result = SDPNewPanel(&newPanel,
             theWindow,
             &bounds,
             true,
             true,
             NULL,
             typeList,
             typeCount,
             identity,
             enumFlags,
             matchTypeHow,
             0);
}
```

```
    if(typesIsResource)
    {
       ReleaseResource((Handle),types);
    }
    else
    {
       DisposHandle((Handle),types);
    }
    DisposPtr(typeList);
    return(result);
}
```

Handling Catalog-Browsing Panel Events

When you receive a window event in your application that contains a Catalog-Browsing panel, you must first determine whether it took place in the panel. For mouse-down events, you can check the coordinates of the event to see if they are in the panel. You must keep track of where the user is working to know how to handle key-down events. For example, you can draw a heavy border (called a **focus rectangle** or focus box) around the panel or around the content portion of the window, according to the last location of a mouse-down event. The focus rectangle 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.

If the event took place in the Catalog-Browsing panel, you should call the `SDPPanelEvent` function, passing in the event record. If the return value of `SDPPanelEvent` tells you the user has double-clicked a record displayed in the panel or otherwise changed the selection, call `SDPGetPanelSelectionSize` and `SDPGetPanelSelection`.

If the return value of `SDPPanelEvent` tells you the event is an update event, call `SDPUpdatePanel`. If the return value says that the user has selected an item other than a record, and your application presents a way to open it, such as a menu command, you can open the item by calling `SDPOpenSelectedItem`. You can determine what the user selected in the panel by calling `SDPGetPanelSelectionState`. Finally, you should adjust your application menus appropriately to reflect their status in the wake of the event.

The routines shown in Listing 4-4 illustrate event handling in a Catalog-Browsing panel. The first application-defined function, `MyProcessEvent`, receives events from the application's main event loop and dispatches them, according to their type, to subroutines also shown in the listing.

Listing 4-4 omits some utility routines that perform functions such as `MyGetPanelForThisWindow`, which takes a window pointer and the address of a panel handle as parameters and returns a Boolean value. It returns `true` if it successfully sets the panel handle to the Catalog-Browsing panel associated with the window. This

application associates a panel with a window by storing the panel handle in the window record's reference constant. The application-defined function `MyDisplayDataForSelection` shows code that retrieves a catalog record selected by the user and unpacks the record information. The listing omits code that extracts other information and that displays the unpacked information.

Listing 4-4 Handling events in a Catalog-Browsing panel

```
void MyProcessEvent(EventRecord *ev)
{
   SDPPanelHandle panel;

   switch (ev->what) {
         case mouseDown:
         case mouseUp:
            MyHandleMouseUp(ev);
            break;
         case keyDown:
         case autoKey:
            if (MyGetPanelForThisWindow(FrontWindow(),&panel) == true)
               MyHandleKeyDownsForPanel(FrontWindow(),ev,panel);
            break;
         case updateEvt:
            MyHandleUpdates((WindowPtr)ev->message);
            break;
         case activateEvt:
            MyHandleActivates(ev);
            break;
         case osEvt:
            MyHandleSREvt(ev->message);
            break;
         case nullEvent:
            MyHandleIdle(ev,FrontWindow());
            break;
   }
}

void MyHandleMouseUp(EventRecord *ev)
{
   WindowPtr window;
   Rect limit;
   SDPPanelState whatHappened;
   SDPPanelHandle panel;
```

```
   OSErr err;

   SetRect(&limit,-32000,-32000,32000,32000);
   switch (FindWindow(ev->where,&window))
   {
      case inDrag:
         DragWindow(window,ev->where,&limit);
         break;
      case inGoAway:
         if (TrackGoAway(window,ev->where))
            gDone = true;
         break;
      case inMenuBar:
         /* MyDoMenuCommand executes the menu command the user chose. */
         MyDoMenuCommand(FrontWindow(),MenuSelect(ev->where));
         break;
      case inSysWindow:
         SystemClick(ev,window);
         break;
      case inContent:
         if (MyGetPanelForThisWindow(window,&panel) == true)
         {
            /* MyClickedInOurPanel returns true if click was in panel. */
            if (MyClickedInOurPanel(ev, (*panel)->bounds) == true)
            {
                  /* Pass the event to the Standard Catalog Package. */
                  err = SDPPanelEvent(panel, ev, &whatHappened);
                  if (err == noErr)
                     MyHandleUserSelection(window,whatHappened,panel);
            }
         }
         break;
   }
}

void MyHandleKeyDownsForPanel(WindowPtr window, EventRecord *ev,
                              SDPPanelHandle panel)
{
   short theChar;
   SDPPanelState whatHappened;
   OSErr err;
   /* dummy string that holds keys pressed by the user */
   RString sKeyDowns = {smRoman,1,{'?'}};
```

```
   theChar = ev->message & charCodeMask;
   if ((ev->modifiers & cmdKey) != 0)
      MyDoMenuCommand(window, MenuKey(theChar));
   else
   {
      /* Did they press the Return key? If so,
         open the currently selected item. */
      if (theChar == 0x0D)
      {
         err = SDPOpenSelectedItem(panel,&whatHappened);
         MyHandleUserSelection(window,whatHappened,panel);
      }
      else
      {
         /* First check if last keypress was less than kMaxTickDelay
            ticks ago - if so, append the current character onto
            our current search string and continue the search. */
         if (((ev->when - gLastTime) < kMaxTickDelay))
         {
            /* Append character only if it doesn't overflow our buffer. */
            if (sKeyDowns.dataLength < kRStringMaxBytes)
            {
               /* Append key onto our previously saved string. */
               sKeyDowns.body[sKeyDowns.dataLength] = theChar;
               ++sKeyDowns.dataLength;
            }
         }
         else
         {
            /* Last keypress was more than kMaxTickDelay ticks ago,
               so treat it as a "new" key to search on. */
            sKeyDowns.body[0] = theChar;
            sKeyDowns.dataLength = 1;
         }

         /* Save time of last keypress for the comparison above. */
         gLastTime = ev->when;
         /* Go select the appropriate item based on our search string. */
         SDPSelectString(panel, &sKeyDowns);
         MyHandleUserSelection(window, kSDPChangedSelection, panel);
      }
   }
}
```

```
void MyHandleUserSelection(WindowPtr window, SDPPanelState whatHappened,
                           SDPPanelHandle panel)
{
   OSErr err;
   SDPSelectionState itsState;

   err = SDPGetPanelSelectionState(panel, &itsState);
   if (err == noErr)
   {
      switch (itsState)
      {
         case kSDPContainerSelected:
            if (whatHappened == kSDPChangedSelection)
            {
               MyDisplayDataForSelection(panel, window);
            }
            break;
         case kSDPRecordSelected :
            if (whatHappened == kSDPChangedSelection)
            {
               MyDisplayDataForSelection(panel, window);
            }
            break;
         case kSDPRecordAliasSelected:
            break;
         case kSDPLockedContainerSelected:
            break;
         case kSDPContainerAliasSelected:
            break;
         case kSDPNothingSelected:
            break;

      }
   }
}

void MyDisplayDataForSelection(SDPPanelHandle panel, WindowPtr window)
{
   OSErr err;
   PackedDSSpec selection;
   DSSpec dss;
   RLI theRLI;
   RecordID rid;
   RString sTempRStr;
```

```
   err = SDPGetPanelSelection(panel, &selection);
   if (err == noErr)
   {
      OCEUnpackDSSpec(&selection, &dss, &rid);
      OCEUnpackRLI(rid.rli,&theRLI);

      /* record name */
      OCECopyRString(rid.local.recordName, &sTempRStr, kRStringMaxBytes);
      /* display the record name now in sTempRStr */
      ...

      /* catalog name */
      OCECopyRString((RStringPtr)theRLI.directoryName, &sTempRStr,
                        kRStringMaxBytes);
      /* display the catalog name now in sTempRStr */
      ...

      /* record type */
      OCECopyRString(rid.local.recordType, &sTempRStr, kRStringMaxBytes);
      /* display the record type now in sTempRStr */
      ...

   }
}

void MyHandleUpdates(WindowPtr window)
{
   GrafPtr savePort;
   SDPPanelHandle panel;

   GetPort(&savePort);
   SetPort(window);
   BeginUpdate(window);
   EraseRect(&window->portRect);

   if (MyGetPanelForThisWindow(window, &panel) == true)
      SDPUpdatePanel(panel, savePort->visRgn);

   EndUpdate(window);
   SetPort(savePort);
}
```

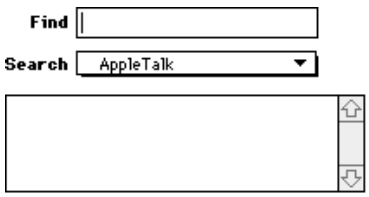
```
void MyHandleActivates(EventRecord *ev)
{
   if ((ev->modifiers & activeFlag) != 0) {
      MyDoActivate((WindowPtr)ev->message);
   }
   else {
      MyDoDeactivate((WindowPtr)ev->message);
   }
}

void MyDoActivate(WindowPtr window)
{
   SDPPanelHandle panel;

   if (MyGetPanelForThisWindow(window, &panel) == true)
   {
      SDPEnablePanel(panel, true);
      SDPUpdatePanel(panel, nil);
   }
}

void MyDoDeactivate(WindowPtr window)
{
   SDPPanelHandle panel;

   if (MyGetPanelForThisWindow(window, &panel) == true)
      SDPEnablePanel(panel, false);
}

void MyHandleSREvt(long message)
{
   extern Boolean gInBackground;
   unsigned long whatMessage;

   whatMessage = message >> 24;

   if (whatMessage == suspendResumeMessage) {
      if ((message & 1) != 0) {
         gInBackground = false;
         SetCursor(&qd.arrow);
         if (FrontWindow()) {
            HiliteWindow(FrontWindow(),true);
            MyDoActivate(FrontWindow());
         }
```

```
      }
      else if (FrontWindow()) {
         gInBackground = true;
         HiliteWindow(FrontWindow(),false);
         MyDoDeActivate(FrontWindow());
      }
   }
}

void MyHandleIdle(EventRecord *ev, WindowPtr window)
{
   OSErr err;
   SDPPanelState whatHappened;
   SDPPanelHandle panel;

   if (MyGetPanelForThisWindow(window, &panel) == true)
      err = SDPPanelEvent(panel, ev, &whatHappened);
}
```

Creating and Disposing of a Find Panel

The Find panel provides an editable text box, a scrolling list, and a pop-up menu that you can place in any window. The text box lets the user enter a search string. The menu specifies the catalogs and volumes to be searched. The list shows AOCE records that have the record types you specify and that have names that start with the text string the user types in the text box. The Find panel allows users to search AOCE catalogs, personal catalogs, and HFS volumes and to select a record. Figure 4-5 shows a Find panel.

Figure 4-5 The Find panel



The preferred user interface to display a Find panel is in a dialog box that also displays a Catalog-Browsing panel. You should provide clearly labeled buttons enabling the user to switch between the find and Catalog-Browsing panels within the dialog box. The AppleMail application included with the AOCE software provides an example of this user interface, which is invoked when the user clicks the Recipients button on a mailer.

The function shown in Listing 4-5 illustrates how to create a Find panel. If the call to the `SDPNewFindPanel` function succeeds, and if there is a `'STR#'` resource available containing help balloon strings for the Find panel, the function installs Balloon Help online assistance. Finally, the function installs an application-defined callback routine for the Find panel to call whenever it is busy, such as while it is searching.

Listing 4-5 Creating a Find panel

```
OSErr MyCreateFindPanel(WindowPtr            window,
                        const RStringPtr  *typesList,
                        unsigned long     typeCount,
                        AuthIdentity      identity,
                        short             helpResourceID)
{
   OSErr status;
   Point panelLoc;
   SDPFindPanelHandle thePanel;

   SetPt(&panelLoc, 0, 0);
   status = SDPNewFindPanel(
         &thePanel,                 /* address of panel handle*/
         window,                    /* in this window */
         panelLoc,                  /* at upper left in window */
         kStandardFindLayout,       /* normal layout */
         false, false,              /* initially invisible */
         typesList,                 /* types to display */
         typeCount,                 /* number of types in list */
         kExactMatch,               /* display only list types */
         identity,                  /* identity of the caller */
         10,                        /* simultaneous searches */
         kSDPFindPanelTextHasFocus, /* text input field focus */
         (long) window              /* reference constant */
         );

   if (status == noErr && helpResourceID != 0)
      status = SDPSetFindPanelBalloonHelp(
                                 thePanel, helpResourceID);

   if (status == noErr)
      status = SDPInstallFindPanelBusyProc(
                                 thePanel, MyArrowSpinnerFunc);
```

```
   return status;
}
```

The code in Listing 4-6 illustrates an application-defined function that disposes of a Find panel. The only feature of this function in addition to calling `SDPDisposeFindPanel` is a check to see if the panel has already been deallocated.

Listing 4-6 Disposing of a Find panel

```
OSErr MyFindPanelDispose(SDPFindPanelHandle thePanel)
{
   OSErr status;

   if (thePanel == NULL)
      status = noErr;
   else {
      status = SDPDisposeFindPanel(thePanel);
      thePanel = NULL;
   }
   return status;
}
```

Standard Catalog Package Reference

This section describes the data types and routines provided by the Standard Catalog Package.

Data Types

The Standard Catalog Package routines use the data types described in this section.

Catalog-Browsing Panel Structure

The Catalog-Browsing panel provides a way for users to locate catalogs, look through them, and select individual records.

The `SDPPanelRecord` structure is the main data type representing the panel. You can examine the contents of any of the fields, but you should not write to any field except the `refCon` field, which you can use for whatever purpose you wish. You might want to look at the contents of the `bounds` field, which stores the rectangle around the panel in the local coordinates of the window that contains the panel. (You provide the `GrafPort`

structure for this window when you call the SDPNewPanel function.) The SDPNewPanel and SDPGetNewPanel functions return handles to SDPPanelRecord structures.

IMPORTANT

The public fields of the SDPPanelRecord structure are followed by private fields. You must not resize the handle to this structure or allocate one yourself. Always use the SDPGetNewPanel function (page 4-34) or SDPNewPanel function (page 4-30) to allocate an SDPPanelRecord structure. s

```
struct SDPPanelRecord {
   Rect           bounds;     /* rectangle around panel */
   Boolean        visible;    /* Is panel visible? */
   Boolean        enabled;    /* Is panel enabled? */
   Boolean        focused;    /* Is focus box around panel? */
   Byte           padByte;    /* reserved */
   AuthIdentity   identity;   /* auth ID of caller of panel */
   long           refCon;     /* for your use */
   Rect           listRect;   /* rectangle on scrolling list */
   Rect           popupRect;  /* rectangle around pop-up menu */
   short          numberOfRows;/* rows in scrolling list */
   short          rowHeight;  /* height of list rows (points) */
   Boolean        pabMode;    /* Is panel in personal catalog? */
};
```

Field descriptions

bounds	The rectangle around the panel in the local coordinates of the window that contains the panel. The bounds rectangle includes both the scrolling field and the pop-up menu above it.
visible	A Boolean value indicating controlling whether the panel is currently visible. You can use the SDPShowPanel and SDPHidePanel functions to set this value.
enabled	A Boolean value that specifies whether the panel is currently enabled. You can use the SDPEnablePanel function to set this value.
focused	A Boolean value that specifies whether the panel has a focus rectangle drawn around it to indicate that it is the target of keystroke events.
identity	The authentication identity of the caller of the panel, corresponding to the name and password of the user.
refCon	A reference constant. You can use this field for whatever you wish.
listRect	The rectangle around only the scrolling list.
popupRect	The rectangle around only the pop-up menu.
numberOfRows	The number of lines in the scrolling list.
rowHeight	The height, in printer's points, of each line in the scrolling list.

`pabMode` A Boolean value indicating whether the panel is browsing the user's default personal catalog, rather than a standard catalog.

Find Panel Structure

The Find panel provides a way for users to locate catalogs, look through them, and select individual records.

The `SDPFindPanelRecord` structure is the main data type representing the Find panel. Its fields are reserved for internal use only, except for the `refCon` field, which you can use for whatever purpose you wish. The `SDPNewFindPanel` function returns a handle to an `SDPFindPanelRecord` structure.

IMPORTANT

The public fields of the `SDPFindPanelRecord` structure are followed by private fields. You must not resize the handle to this structure or allocate one yourself. Always use the `SDPNewFindPanel` function (page 4-61) to allocate an `SDPFindPanelRecord` structure. s

```
struct SDPFindPanelRecord {
   Point             upperLeft;                /* reserved */
   Boolean           visible;                  /* reserved */
   Boolean           enabled;                  /* reserved */
   Boolean           nowFinding;               /* reserved */
   Byte              padByte;                  /* reserved */
   SDPFindPanelFocus currentFocus;             /* reserved */
   AuthIdentity      identity;                 /* reserved */
   short             simultaneousSearchCount;  /* reserved */
   long              refCon;                   /* for your use */
};
```

Field descriptions

`upperLeft` The upper-left corner of the Find panel in the window's local coordinates.

`visible` A Boolean value indicating whether the Find panel is currently visible. You can use the `SDPShowFindPanel` and `SDPHideFindPanel` functions to set this value.

`enabled` A Boolean value that specifies whether the Find panel is currently enabled. You can use the `SDPEnableFindPanel` function to set this value.

`nowFinding` A Boolean value indicating whether the Find panel is currently busy.

`currentFocus` A constant that specifies whether there is a focus rectangle in the Find panel, and if so, whether it is around the scrolling list in the Find panel or around the text-input field (the Find field; see Figure 4-5 on page 4-18). You can use the `SDSetFindPanelFocus` function to change the location of the focus rectangle.

`identity`	The authentication identity of the caller of the Find panel, corresponding to the name and password of the user.
`simultaneousSearchCount`	The number of catalog searches that can be done simultaneously. See the description of this parameter on page 4-62 for more information.
`refCon`	A reference constant. You can use this field for whatever you wish.

RString List

The `SDPGetCategories` function (page 4-91) and `SDPGetCategoryTypes` function (page 4-92) return handles to lists of items in the form of a packed `RString` list.

```
typedef PackedPathNamePtr *PackedRStringListHandle;
```

A `PackedPathName` structure contains a length plus an array of bytes:

```
#define PackedPathNameHeader\
   unsigned short dataLength;/* excludes the dataLength field */

struct PackedPathName {
   PackedPathNameHeader
   Byte  data[kPathNameMaxBytes - sizeof(unsigned short)];
};
```

The packed `RString` list consists of a `PackedPathName` structure containing packed `RString` structures. Each `RString` structure contains a script code, a length, and a string: Use the AOCE string utility routines described in the chapter “AOCE Utilities” in this book to unpack and manipulate this structure. Use the `OCEDNodeNameCount` function described in that chapter to determine the number of items in a `PackedPathName` structure.

```
#define RStringHeader      \
   CharacterSet charSet;   \
   unsigned short dataLength;

struct RString {
   RStringHeader
   Byte  body[kRStringMaxBytes];    /* place of characters */
};
```

Standard Catalog Package Functions

This section and the following sections describe the routines provided by the AOCE Standard Catalog Package. Many Standard Catalog Package routines require you to

provide an authentication identity as input. The subsection “Authenticating a User,” beginning on page 4-25, describes routines that prompt the user for a name and password, authenticate the user, and return the authentication identity number to your application.

The second dialog box allows the user to search for a record by specifying the name or the first part of the name of the record.

The section “Creating, Displaying, and Disposing of a Catalog-Browsing Panel,” beginning on page 4-29, provides functions that you can use to display a Catalog-Browsing panel. This panel lets the user browse through HFS catalogs and AOCE catalogs and select a record from a catalog. You can place a Catalog-Browsing panel in any of your application’s windows.

The routines in“Handling Events in a Catalog-Browsing Panel,” beginning on page 4-51, allow you to process events related to the panel, such as a user selecting an item displayed in the panel.

The section “Creating, Displaying, and Disposing of a Find Panel” on page 4-61 provides functions that you can use to display a Find panel. The Find panel allows the user to search for a record by specifying the name or the first part of the name of the record. The Find panel is similar in implementation to the Catalog-Browsing panel; you can place one in any of your application’s windows.

The routines in “Handling Events in a Find Panel,” beginning on page 4-75, allow you process events related to the Find panel.

The section “Resolving Aliases,” beginning on page 4-85, provides functions that resolve HFS and AOCE aliases of objects in the AOCE catalog system.

The section “Obtaining Icons and Lists of Catalog-Item Categories and Types,” beginning on page 4-88, provides functions that return icons for AOCE components such as attributes, records, and catalogs, and functions that return lists of the record types and record categories currently available.

Assembly-Language Interface

To call a Standard Catalog Package routine from assembly language, you must do the following:

1. Push space for the function result and all routine parameters (in Pascal calling-convention order) on the stack.
2. Put in the D0 register a long word consisting of the parameter word count for the routine followed by the routine selector. The parameter word count indicates how many words of parameters you are placing on the stack; for example, if the function has two parameters and each is a pointer, the parameter word count for the function is $0004.
3. Call the Standard Catalog Package trap, $AA5D.

Each routine description in the following sections lists the parameter word count and routine selector for that routine.

Note
The routines described in the section "Obtaining Icons and Lists of Catalog-Item Categories and Types," beginning on page 4-88 use the $AA5C trap and do not require a parameter count. u

Authenticating a User

The `SDPPromptForID` function in this section authenticates a user. The function displays dialog boxes that enable the user to enter a name and password, and it returns either a local or specific authentication identity. Authentication identities are described in "Creating an Authentication Identity," beginning on page 4-6. You must provide a valid authentication identity as a parameter to many AOCE functions.

SDPPromptForID

The `SDPPromptForID` function displays a dialog box that allows the user to enter a name and password. The function returns the user's authentication identity.

```
pascal OSErr SDPPromptForID (AuthIdentity *id,
                             ConstStr255Param guestPrompt,
                             ConstStr255Param specificIDPrompt,
                             ConstStr255Param localIDPrompt,
                             const RString *recordType,
                             SDPIdentityKind permittedKinds,
                             SDPIdentityKind *selectedKind,
                             const RecordID *loginFilter,
                             SDPLoginFilterKind filterKind);
```

`id` The authentication identity returned by the function. This identity corresponds to the name and password entered by the user and can be a local identity or a specific identity, depending on the value you specify for the `permittedKinds` parameter and which radio button the user selects. If the user cancels the authentication, the function returns the `userCanceledErr` result code and the ID is undefined. If the user logs on with guest access, the function returns 0 in this parameter.

`guestPrompt`
The prompt string displayed when the user selects the Guest radio button.

`specificIDPrompt`
The prompt string displayed when the user selects the PowerShare account radio button.

`localIDPrompt`
: The prompt string displayed when the user selects the "Fewer Choices" version of the dialog box or when the user selects the Key Chain Access Code radio button.

`recordType`
: The record type of the records that can be authenticated. The function uses this type to determine for which records to accept a name and password in the dialog box. If you specify `nil`, the function uses the user record type. To obtain the value to use for a standard record type, you pass an index value to the utility function `OCEGetIndRecordType`. For example, you can use the index value `kUserRecTypeNum` to get a pointer to the user record type. Other index values are listed in the chapter "AOCE Utilities" in this book.

`permittedKinds`
: A value that specifies whether the user should be allowed to log on with guest access (an authentication ID of 0), as an alternate user (that is, with the password for a specific identity), or with the password for the local identity. Use any combination of the bit masks `kSDPGuestMask`, `kSDPSpecificIdentityMask`, and `kSDPLocalIdentityMask` for this parameter.

`selectedKind`
: A value that indicates which type of identity the function returned: an authentication ID of 0 (guest access), a specific identity (alternate-user access), or a local identity. This parameter can have any one of the following values: `kSDPGuestMask`, `kSDPSpecificIdentityMask`, or `kSDPLocalIdentityMask`.

`loginFilter`
: A pointer to a record ID that specifies a catalog or record name that is initially displayed in the dialog box. You use the `filterKind` parameter to specify what is in this parameter and whether to restrict the user to this catalog or record. Specify `nil` for this parameter to allow any user to log on to any catalog. The function uses this parameter only when the user selects the PowerShare account radio button.

`filterKind`
: A value that indicates which type of information you have provided in the `loginFilter` parameter. If you specify `kSDPRestrictToDirectory` for the `filterKind` parameter, you must specify a catalog in the `loginFilter` parameter. The dialog box displays this catalog and allows only users with records in that catalog to log on. If the record ID you provide in the `loginFilter` parameter includes a record name, the function places that name in the Name text field of the dialog box but doesn't restrict the user to that record.

  If you specify `kSDPRestrictToRecord` for the `filterKind` parameter, you must provide the record ID of a user record for the `loginFilter` parameter, and only the user specified by that record ID can log on.

If you specify `kSDPSuggestionOnly` for the `filterKind` parameter, the dialog box initially displays the catalog and record name you specify in the `loginFilter` parameter but does not restrict the user to that selection.

If you specify `nil` for the `loginFilter` parameter, the function ignores the `filterKind` parameter.

The function uses the `filterKind` parameter only when the user selects the PowerShare account radio button.

DESCRIPTION

The dialog box displayed by the `SDPPromptForID` function lets the user verify a local identity by entering a password, log on with a specific identity by selecting a catalog and then entering a name and password, or log on to a catalog as a guest. The function returns the authentication identity for the user in the `id` parameter. If the name or password entered by the user is not valid, the dialog box prompts the user to reenter the information. If the user fails three times to enter the correct password, the function returns the `kSDPTooManyLoginAttempts` result code. If the user cancels the dialog box, the function returns the `userCanceledErr` result code.

The owner or principal user of a Macintosh computer enters a name and a password during the first attempt to use an AOCE catalog or mailbox. If the user elects access as the principal user and has never set up a local identity, the function guides the user through the process of setting up a local identity. If the user selects the Key Chain Access Code radio button and has previously entered a password to verify the local identity during the current computer session, the function displays a dialog box with the user's name but without a text-entry field for the password. When the user clicks the OK button, the function returns the local identity.

To obtain a local identity without displaying a dialog box, call the `AuthGetLocalIdentity` function. If that function returns an error, then call the `SDPPromptForID` function to allow the user to unlock the local identity or set one up if none exists.

You can use the `loginFilter` and `filterKind` parameters to restrict the dialog box to a single catalog or record, or to set a specific record as the one displayed initially in the dialog box.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0010	$0388

RESULT CODES

`noErr`	0	No error
`userCanceledErr`	–128	User clicked Cancel button
`kSDPTooManyLoginAttempts`	–1951	User failed three times to enter correct password

SEE ALSO

You can use the `DirEnumerateDirectoriesGet` and `DirEnumerateDirectoriesParse` functions to obtain discriminators for catalogs that are listed in the PowerTalk Setup catalog. You can use the `DirNetSearchADAPDirectoriesGet` and `DirNetSearchADAPDirectoriesParse` functions to obtain discriminators for all catalogs on the network. These functions are all described in the chapter "Catalog Manager" in this book.

You can use the `AuthGetLocalIdentity` function to obtain a local identity without displaying a dialog box. See the chapter "Authentication Manager" in this book for a description of that function.

Sorting a Personal Catalog

The routine in this section operates on personal catalogs.

SDPRepairPersonalDirectory

The `SDPRepairPersonalDirectory` function sorts the contents of a personal catalog according to the current script system.

```
pascal OSErr SDPRepairPersonalDirectory (FSSpec *pd,
                                         Boolean showProgress);
```

`pd` The file system specification structure for the personal catalog you wish to sort.

`showProgress`
A Boolean value indicating whether the Standard Catalog Package should display a progress bar to show the user the progress of the sorting operation.

DESCRIPTION

If the user moves a personal catalog to a computer whose operating system uses a different script system from the one last used to sort the catalog, the personal catalog must be resorted before the Catalog Manager can open it. If the `DirOpenPersonalDirectory` function returns the error `kOCEVersionErr`, you must

call the SDPRepairPersonalDirectory function to resort the personal catalog and then call the DirOpenPersonalDirectory function again to open the catalog.

If you specify true for the showProgress parameter, the Standard Catalog Package first displays a dialog box that tells the user there is a problem with the personal catalog and asks if it should correct the problem. If the user elects to correct the problem, the Standard Catalog Package sorts the catalog, displaying a progress bar as it does so. If you specify false for the showProgress parameter, the Standard Catalog Package sorts the catalog without any feedback to the user.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0003	$1A2C

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

The DirOpenPersonalDirectory function is described in the chapter "Catalog Manager" in this book.

Creating, Displaying, and Disposing of a Catalog-Browsing Panel

You can call either the SDPNewPanel function (page 4-30) or the SDPGetNewPanel function (page 4-34) to create a new panel. The two functions are identical, except that you pass all the parameters to the SDPNewPanel function when you call the function, whereas you specify several of the parameters to the SDPGetNewPanel function in a resource.

The SDPNewPanel and SDPGetNewPanel functions allow you to specify a particular container (volume, folder, AOCE catalog, personal catalog, or node) for the panel to display when it opens. After the panel is open, you can use the SDPSetPath function (page 4-38) to change the container the panel is displaying.

The other functions in this section, as their names suggest, allow you to hide, show, enable, and disable a panel, redraw a panel, move a panel within your window, and change the size of a panel. When you are finished using a panel or close the window, call the SDPDisposePanel function (page 4-50).

Once you have placed a panel in your window, you use the functions in "Handling Events in a Catalog-Browsing Panel," beginning on page 4-51, to handle events that take place within the panel.

SDPNewPanel

The SDPNewPanel function creates a new Catalog-Browsing panel with the size and location you specify in the window you specify.

```
pascal OSErr SDPNewPanel (SDPPanelHandle *newPanel,
                          WindowPtr window,
                          const Rect *bounds,
                          Boolean visible,
                          Boolean enabled,
                          const PackedRLI *initialRLI,
                          const RStringPtr *typesList,
                          unsigned long typeCount,
                          AuthIdentity identity,
                          DirEnumChoices enumFlags,
                          DirMatchWith matchTypeHow,
                          long refCon);
```

`newPanel` The handle to the new panel created by the function. The SDPNewPanel function allocates this handle.

`window` The window into which the panel is to be inserted.

`bounds` The bounds for the panel, including both the pop-up menu above the scrollable list and the list itself (see Figure 4-4), in the local coordinates of the window you specify in the `window` parameter. You can calculate the minimum width of a panel as

```
5 * FontInfo.widMax + 42
```

and the minimum height as

```
Max(64, 3 * FontHeight) + fontHeight + 18
```

where

```
fontHeight = FontInfo.ascent + FontInfo.descent +
             FontInfo.Leading.
```

If the bounds you specify do not enclose an integral number of items in the scrollable list, the SDPNewPanel function shortens the panel slightly to prevent the last item in the list from being cropped.

`visible` A Boolean value that specifies whether the panel is initially visible. Set this parameter to `true` if you want the panel to be visible. You can use the SDPShowPanel and SDPHidePanel functions to show and hide a panel.

`enabled` A Boolean value that specifies whether the panel is initially enabled. Set this parameter to `true` to enable the panel. You can use the `SDPEnablePanel` function to enable and disable the panel.

`initialRLI`
A pointer to the volume, folder, AOCE catalog, personal catalog, or node that the panel should display initially. If you specify `nil` for this parameter, the panel displays a list of volumes and AOCE catalogs. Specify –1 for this parameter to open a Personal-Catalog panel rather than the standard Catalog-Browsing panel. You can use the `OCEGetDirectoryRootPackedRLI` function to get the location of the root of all catalogs (represented on the desktop by the Catalogs icon).

`typesList` A pointer to an array, each item of which is of type `RStringPtr`, providing a list of the types of records you want the panel to display. The panel displays only records of the types you specify in this list. The `matchTypeHow` parameter specifies how the `SDPNewPanel` function interprets the types list. To display aliases of dNodes, you must include a record type of "DNode" (the value `kDNodeRecTypeNum`) and include the value `kEnumAliasMask` in the `enumFlags` parameter.

`typeCount` The number of record types in your types list.

`identity` The authentication identity of the caller. Specify 0 for guest access.

`enumFlags` A mask value that specifies what sort of information you want the Standard Catalog Package to display in the panel. You can set this field to any combination of the following constants. The constant `kEnumAllMask` combines all of the other values.

`kEnumDistinguishedNameMask`
records

`kEnumAliasMask`
aliases

`kEnumPseudonymMask`
pseudonyms

`kEnumDNodeMaskd`
Nodes

`kEnumInvisibleMask`
invisible entities

`kEnumAllMaskall`
entities

`matchTypeHow`
A constant that specifies how the function interprets the types list. The possible values for this parameter are described in the following function description.

`refCon` A reference constant. The function places this value in the `refCon` field of the `SDPPanelRecord` structure pointed to by the `newPanel` parameter. You can use the `refCon` parameter for whatever you wish.

DESCRIPTION

You use the SDPNewPanel function to create a new Catalog-Browsing panel in your window. You specify the location and size of the panel, and can specify that the panel is to be initially visible or invisible and initially enabled or disabled. You can use the enumFlags parameter to specify which types of entities (records, dNodes, aliases, or pseudonyms) the panel displays.

The typesList and matchTypeHow parameters let you specify what types of records the panel should display. The possible values for the matchTypeHow parameter are as follows:

```
enum {
   kMatchAll,
   kExactMatch,
   kBeginsWith,
   kEndingWith,
   kContaining
};
typedef unsigned char DirMatchWith;
```

Constant descriptions

kMatchAll	Display all record types. The function ignores the typesList and typeCount parameters.
kExactMatch	Display only the record types in the types list.
kBeginsWith	Display only records whose types begin with the string pointed to by the typesList parameter. The typesList parameter can point to only one string; if the typeCount parameter is not equal to 1, the function returns an error.
kEndingWith	Display only records whose types end with the string pointed to by the typesList parameter. The typesList parameter can point to only one string; if the typeCount parameter is not equal to 1, the function returns an error.
kContaining	Display only records whose types contain the string pointed to by the typesList parameter. The typesList parameter can point to only one string; if the typeCount parameter is not equal to 1, the function returns an error.

You can use the SDPGetCategories and SDPGetCategoryTypes functions to determine what record types are available and use that information to let the user select which types of records the Catalog-Browsing panel should display. Then you can use the typesList parameter to restrict the panel to those record types.

Subsequently, you can use the other routines in this and the following section to hide and show the panel, enable and disable it, and handle events that occur within the panel. If you wish to specify some parameters in a resource rather than when you call the function, use the SDPGetNewPanel function instead.

Listing 4-3 on page 4-9 illustrates the use of the SDPNewPanel function to create a new panel.

SPECIAL CONSIDERATIONS

The window's font, text size, and text style affect how the panel appears when it is created.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0015	$0064

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

The `SDPGetNewPanel` function, described next, does the same thing as the `SDPNewPanel` function but allows you to specify several of the parameters in a resource.

You can use the `SDPHidePanel` function (page 4-43) and `SDPShowPanel` function (page 4-44) to hide and show a Find panel.

You can use the `SDPEnablePanel` function (page 4-45) to enable and disable the Find panel.

Use the other routines in this section to dispose of, draw, move, resize, and otherwise manipulate panels. Use the routines in "Handling Events in a Catalog-Browsing Panel," beginning on page 4-51, to handle events that occur in the panel and to determine what record the user selected.

The `PackedRecordLocationInfo` structure is described in the chapter "Catalog Manager" in this book.

You can use the `OCEGetDirectoryRootPackedRLI` function, described in the chapter "AOCE Utilities" in this book, to get the location of the root of all catalogs.

You can use the `SDPPromptForID` function (page 4-25) to get a local or specific identity.

You can use the `SDPGetCategories` function (page 4-91) to obtain a list of all the record-type categories currently available and the `SDPGetCategoryTypes` function (page 4-92) to list all the types of records included in a specific category.

SDPGetNewPanel

The SDPGetNewPanel function creates a new panel in the window you specify with the size and location specified in a resource of type 'panl'.

```
pascal OSErr SDPGetNewPanel (SDPPanelHandle *newPanel,
                             short resourceID,
                             WindowPtr window,
                             const PackedRLI *initialRLI,
                             AuthIdentity identity);
```

newPanel A handle to the new panel created by the function.

resourceID
The resource ID of the panel resource, which contains several parameters for use by the function.

window A pointer to the window into which the panel is to be inserted.

initialRLI
A pointer to the volume, folder, AOCE catalog, node, or personal catalog that the dialog box should display initially. If you specify nil for this parameter, the dialog box displays a list of volumes and AOCE catalogs. Specify –1 for this parameter to open a Personal-Catalog panel rather than the standard Catalog-Browsing panel.

identity The authentication identity of the caller. Specify 0 for guest access.

DESCRIPTION

The SDPGetNewPanel function creates a new panel and returns its handle, given a resource with the following Rez type:

```
type 'panl' {
   rect;                              /* bounds parameter */
   byte     invisible, visible;       /* visible parameter */
   byte     disabled, enabled;        /* enabled parameter */
   longint;                           /* enumFlags parameter */
   integer;                           /* matchTypeHow parameter */
   longint;                           /* refCon parameter */
   integer = $$CountOf(TypeIdArray);/* typeCount parameter */
   array TypeIdArray {
    integer;                          /* typesList parameter
                                         ('rtyp' resource ID
                                           for each type) */
   };
};

#define kSDPPanelResourceType    'panl'
```

See the SDPNewPanel function on page 4-30 for descriptions of these parameters.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0009	$0065

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

The SDPNewPanel function (page 4-30) does the same thing as the SDPGetNewPanel function, but you specify all of the parameters when you call the SDPNewPanel function rather than using a panel resource.

Use the other routines in this section to dispose of, hide, show, enable, disable, draw, move, and resize panels. Use the routines in "Handling Events in a Catalog-Browsing Panel," beginning on page 4-51, to handle events that occur in the panel and to determine what record the user selected.

The PackedRecordLocationInfo structure is described in the chapter "Catalog Manager" in this book.

You can use the OCEGetDirectoryRootPackedRLI function, described in the chapter "AOCE Utilities" in this book, to get the location of the root of all catalogs.

You can use the SDPPromptForID function (page 4-25) to get a specific or local identity.

SDPInstallPanelBusyProc

The SDPInstallPanelBusyProc function installs an application-defined routine that the panel calls when it is busy.

```
pascal OSErr SDPInstallPanelBusyProc (SDPPanelHandle panel,
                                      PanelBusyProc busyProc);
```

panel A handle for the panel for which you want to install a panel-busy callback routine.

`busyProc` A pointer to your callback routine. To remove a callback routine that you installed previously, specify `nil` for this parameter.

DESCRIPTION

If you use the `SDPInstallPanelBusyProc` function to install a panel-busy callback routine, the Catalog-Browsing panel calls your routine whenever the panel is busy. For example, if the user double-clicks a dNode in the panel, the panel calls your callback routine while it searches the dNode.

The panel passes to your callback routine the handle to the panel and a Boolean value that indicates whether the panel is busy. You can use your callback routine to provide feedback to the user that indicates that the panel is busy. One good way to do this is to display the Standard Catalog Package's spinning arrow icon.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0079

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

The panel-busy callback routine is described on page 4-94.

SDPSetPanelBalloonHelp

The `SDPSetPanelBalloonHelp` function enables Balloon Help for the panel.

```
pascal OSErr SDPSetPanelBalloonHelp (SDPPanelHandle panel,
                                     short balloonHelpID);
```

`panel` A handle for the panel for which you want to enable Balloon Help.

`balloonHelpID`
The resource ID of a `'STR#'` resource that contains Balloon Help strings for the panel.

DESCRIPTION

You must use the `SDPSetPanelBalloonHelp` function to provide text strings for Balloon Help and to activate Balloon Help for the panel. You must provide two text strings, one for each element of the panel, in the following order:

1. the scrolling list
2. the pop-up menu

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0003	$0078

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Balloon Help is described in the "Help Manager" chapter of *Inside Macintosh: More Macintosh Toolbox*.

SDPSetIdentity

The `SDPSetIdentity` function changes the authentication identity used by the panel.

```
pascal OSErr SDPSetIdentity (SDPPanelHandle panel,
                             AuthIdentity identity);
```

`panel` A handle for the panel for which you want to change the caller's authentication identity.

`identity` The new authentication identity. Specify 0 for guest access.

DESCRIPTION

You can use the `SDPSetIdentity` function to change the authentication identity of the user while the user is browsing catalogs. If the new identity does not have read privileges for the dNode being browsed, the panel moves up levels in the catalog until it

reaches a level at which the user has read privileges and the function returns the `kOCEReadAccessDenied` result code.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0073

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter
`kOCEReadAccessDenied`	–1540	User does not have read privileges.

SEE ALSO

The `SDPPromptForID` function (page 4-25) prompts the user for a password and returns an authentication identity.

SDPSetPath

The `SDPSetPath` function causes the panel to display the contents of the container specified by the packed record location information structure (`packedRLI` data type) you specify.

```
pascal OSErr SDPSetPath (SDPPanelHandle panel,
                         const PackedRLI *prli);
```

`panel` The panel on which you want this function to act.

`prli` A pointer to the record location information for the container whose contents you want the panel to display. Specify `nil` for this parameter to display a list of volumes and AOCE catalogs. You can use the `OCEGetDirectoryRootPackedRLI` function to get the location of the root of all catalogs (represented on the desktop by the Catalogs icon).

DESCRIPTION

When you call the `SDPSetPath` function, the panel's pop-up menu displays the container (volume, folder, AOCE catalog, personal catalog, or node) you specify in the `prli` parameter, and the scrollable list displays the contents of that container. Whereas

the SDPNewPanel and SDPGetNewPanel functions let you specify which container the panel displays when it opens, the SDPSetPath function lets you change the container displayed by a panel that is already open. If the container you specify in the prli parameter is the one currently being browsed in the panel, then this function does nothing.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0070

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

The PackedRecordLocationInfo structure is described in the chapter "Catalog Manager" in this book.

You can use the OCEGetDirectoryRootPackedRLI function, described in the chapter "AOCE Utilities" in this book, to get the location of the root of all catalogs.

To select an item in the scrollable list of the panel, use the SDPSelectString function (page 4-42).

Use the SDPNewPanel (page 4-30) or SDPGetNewPanel (page 4-34) functions to specify which container a panel displays initially.

SDPGetPathLength

The SDPGetPathLength function returns the size of the buffer required to hold the pathname of the current item in the panel's pop-up menu.

```
pascal OSErr SDPGetPathLength (SDPPanelHandle panel,
                               unsigned short *pathNameLength);
```

panel The panel for which you want information.

`pathNameLength`
A pointer to the length, in bytes, of the `PackedRLI` structure representing the current pathname. You must allocate this pointer.

DESCRIPTION

The `SDPGetPathLength` function returns the number of bytes required to hold the `PackedRLI` structure that represents the current pathname of the container (volume, folder, AOCE catalog, dNode, or personal catalog) in the pop-up menu of the panel. You can allocate a buffer of this size and pass a pointer to it in the `prli` parameter of the `SDPGetPath` function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0075

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

You use the value returned by this function to determine how big a buffer to allocate for the `SDPGetPath` function, described next.

SDPGetPath

The `SDPGetPath` function returns the pathname of the current item in the panel's pop-up menu.

```
pascal OSErr SDPGetPath (SDPPanelHandle panel,
                         PackedRLI *prli,
                         short *dsRefNum);
```

`panel` The panel for which you want information.

`prli` A pointer to a buffer you provide to hold the current pathname. Specify `nil` for this parameter if you want the function to return only the `dsRefNum` parameter.

`dsRefNum` A pointer to a location to hold a reference number if the item is a personal catalog. You must allocate this pointer. Specify `nil` for this parameter if you want the function to return only the `prli` parameter.

DESCRIPTION

The `SDPGetPath` function returns the current pathname of the container (volume, folder, AOCE catalog, dNode, or personal catalog) in the pop-up menu of the panel and—if the container is a personal catalog—a reference number. You can call the `SDPGetPathLength` function to determine the size of buffer required for the pathname. The `SDPGetPath` function returns the pathname in packed RLI format.

If you want the function to return either the pathname or the reference number, but not both, specify `nil` for the parameter you do not want returned.

You can use the record location information (`prli`) returned by this function to call Catalog Manager functions, such as `DirGetDirectoryInfo` or `DirFindValue`, that return information about the container. You can use the value returned in the `dsRefNum` parameter if you immediately want to call a Catalog Manager function that takes a personal-catalog reference number as input. Once you have called the `SDPPanelEvent` function or the `SDPDisposePanel` function, you can no longer use this reference number. In that case, you must use the `DirOpenPersonalDirectory` function to open the personal catalog and obtain a new reference number.

SPECIAL CONSIDERATIONS

Because the personal-catalog reference number returned in the `dsRefNum` parameter is valid only while the personal catalog is open, this value may not be valid after you call the `SDPPanelEvent` function and is never valid after you call the `SDPDisposePanel` function.

The pathname of a container returned by the `SDPGetPath` function is not necessarily the same as the pathname you get by using the `SDPGetPanelSelection` function to obtain location information for an item the user has selected in that container. This discrepancy results from the fact that the item might be an alias or information card. The `SDPGetPanelSelection` function returns the location of the original item, not that of the alias or information card itself.

The `SDPGetPath` function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0006	$0076

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

You use the value returned by the SDPGetPathLength function (page 4-39) to determine the size of the buffer to allocate for the pathname.

The packedRLI structure and Catalog Manager routines are described in the chapter "Catalog Manager" in this book.

You call the SDPPanelEvent function (page 4-52) to handle events that take place in the panel. You call the SDPDisposePanel function (page 4-50) to dispose of a panel that you no longer need.

You can use the SDPGetPanelSelection function (page 4-58) to obtain location information for a panel item that a user has selected. You can use the SDPGetPanelSelectionState function (page 4-55) to determine the nature of the selected item.

SDPSelectString

The SDPSelectString function scrolls to and highlights the item in the panel that best matches the string you specify.

```
pascal OSErr SDPSelectString (SDPPanelHandle panel,
                              const RString *name);
```

panel	The panel on which you want this function to act.
name	A pointer to the string you want the function to match.

DESCRIPTION

When you call the SDPSelectString function, the panel's scrollable list displays and highlights the item that best matches the string you specify in the name parameter. The matching algorithm is the same one used by the Standard File dialog box when the user types a string: the function matches characters starting at the beginning of the string. If the function cannot match all of the characters in the string, it selects the item in the list that follows the string alphabetically. For example, if the panel contains the items Andy, Atticus, Bernice, Bruce, and Freda, and you specify the string "Bru", the function highlights "Bruce". If you specify the string "Charlie", the function highlights "Freda". The SDPSelectString function acts on the list at the current level; that is, it does not change the path represented by the item selected in the pop-up menu.

You could use this function, for example, to display the item the user had selected the last time the panel was opened. If the string you specify in the `name` parameter matches the item currently being selected, then this function does nothing.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0074

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

To set a specific path in the pop-up menu of the panel, use the `SDPSetPath` function (page 4-38).

SDPHidePanel

The `SDPHidePanel` function hides a panel.

```
pascal OSErr SDPHidePanel (SDPPanelHandle panel);
```

`panel` The panel you wish to hide.

DESCRIPTION

If a panel is visible, this function makes it invisible by hiding the menu, erasing and invalidating the list's rectangle, and causing the list to not be drawn. If the panel is already hidden, this function does nothing.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$0067

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Use the `SDPShowPanel` function (described next) to display a panel that you have hidden.

SDPShowPanel

The `SDPShowPanel` function displays a panel.

```
pascal OSErr SDPShowPanel (SDPPanelHandle panel);
```

`panel` The panel you wish to display.

DESCRIPTION

You can hide a panel by setting the `visible` parameter in the `SDPNewPanel` or in the `'panl'` resource of the `SDPGetNewPanel` function to `false` or by calling the `SDPHidePanel` function. If the panel is hidden, the `SDPShowPanel` function makes it visible. If the panel is already visible, this function does nothing. Use the `SDPUpdatePanel` function to handle an update event.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$0068

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

You can specify that a panel is to be initially hidden when you create it with the SDPNewPanel function (page 4-30) or SDPGetNewPanel function (page 4-34).

Use the SDPHidePanel function (page 4-43) to hide a panel.

Use the SDPUpdatePanel function (page 4-47) to handle an update event.

SDPEnablePanel

The SDPEnablePanel function enables or disables a panel.

```
pascal OSErr SDPEnablePanel (SDPPanelHandle panel,
                             Boolean enable);
```

panel	The panel you wish to enable or disable.
enable	A Boolean value that specifies whether you wish to enable or disable the panel. Specify true to enable the panel.

DESCRIPTION

This function disables the list and menu so that they do not accept any commands, or enables a disabled panel.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0003	$0069

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

When you call the SDPNewPanel function (page 4-30) to create a panel, you specify whether it should be enabled or disabled initially.

SDPSetFocus

The SDPSetFocus function draws or removes a focus rectangle around a panel.

```
pascal OSErr SDPSetFocus (SDPPanelHandle panel,
                          Boolean focus);
```

panel	The panel for which you wish to draw or remove a focus rectangle.
focus	A Boolean value that specifies whether you wish to draw or remove the rectangle. Specify true to draw the rectangle and false to remove it.

DESCRIPTION

When the user clicks in a panel or uses the Tab key to select it (assuming you support this feature), you can use the SDPSetFocus function to draw a heavy border around the panel. This border, called a focus rectangle (or focus box), indicates to the user that the panel is active and that any keypress commands will be applied to the panel.

If you never call the SDPSetFocus function, the Standard Catalog Package highlights the user's current selection regardless of whether the user is working in the panel or in your window. However, once you call the SDPSetFocus function, the Standard Catalog Package removes highlighting in the panel whenever you remove the focus rectangle from the panel.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0003	$0077

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SDPUpdatePanel

The SDPUpdatePanel function draws all or part of a panel.

```
pascal OSErr SDPUpdatePanel (SDPPanelHandle panel,
                             RgnHandle theRgn);
```

panel	The panel you wish to draw.
theRgn	The region of the window you wish to draw. Any component of the panel that intersects this region is redrawn. For example, if any portion of the list of the panel intersects this region, the Standard Catalog Package redraws the whole list. Pass nil in this parameter to draw the entire panel.

DESCRIPTION

You must call the SDPUpdatePanel function to draw a panel in response to an update event. (Do not pass an update event to the SDPPanelEvent function.) You can also use the SDPUpdatePanel function to draw a panel when your application needs to redraw its window because of some activity other than an update event.

In response to an update event, you should first call the BeginUpdate routine, which takes a window pointer as a parameter. Use the value in the visRgn field of the window's grafPort structure for the theRgn parameter to the SDPUpdatePanel function.

Note that the SDPUpdatePanel function does not use the region specified by theRgn as a clipping region. To clip the drawing region, you must first call the SetClip routine. (Remember to save the old clipping region so that you can restore it when you have finished clipping.)

If you have hidden the panel (either by setting the visible parameter in the SDPNewPanel function or SDPGetNewPanel function to false or by calling the SDPHidePanel function), then the SDPUpdatePanel function does not display the panel. To display a hidden panel, you must call the SDPShowPanel function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$006A

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

Use the SDPShowPanel function (page 4-44) to display a panel that you have hidden with the SDPNewPanel function (page 4-30), SDPGetNewPanel function (page 4-34), or SDPHidePanel function (page 4-43).

Use the BeginUpdate routine to begin the process of updating your window. The chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* describes how to respond to update events. The BeginUpdate routine is described in the chapter "Window Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*.

Use the SetClip routine to set a clipping region. The SetClip routine is described in *Inside Macintosh: Imaging With QuickDraw*.

SDPMovePanel

The SDPMovePanel function moves the panel to a location you specify.

```
pascal OSErr SDPMovePanel (SDPPanelHandle panel,
                           short h,
                           short v);
```

panel	The panel you wish to move.
h	The horizontal coordinate to which you want to move the upper-left corner of the panel, in the local coordinates of the panel's window.
v	The vertical coordinate to which you want to move the upper-left corner of the panel, in the local coordinates of the panel's window.

DESCRIPTION

Use the SDPMovePanel function to move the upper-left corner of the panel to (h, v), given in local coordinates of the panel's window. You set the original location of the panel with the bounds parameter in the SDPNewPanel function or SDPGetNewPanel function.

If you have hidden the panel (either by setting the `visible` parameter in the `SDPNewPanel` function or `SDPGetNewPanel` function to `false` or by calling the `SDPHidePanel` function), then the `SDPMovePanel` function does not display the panel. To display a hidden panel, you must call the `SDPShowPanel` function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$006B

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

You use the `SDPNewPanel` function (page 4-30) or `SDPGetNewPanel` function (page 4-34) to set the original location of the panel.

Use the `SDPShowPanel` function (page 4-44) to display a panel that you have hidden with the `SDPNewPanel` function, `SDPGetNewPanel` function, or `SDPHidePanel` function (page 4-43).

SDPSizePanel

The `SDPSizePanel` function resizes the panel to a size you specify.

```
pascal OSErr SDPSizePanel (SDPPanelHandle panel,
                           short width,
                           short height);
```

`panel` The panel you wish to resize.

`width` The width, in QuickDraw coordinates, that you want to use for the panel. You must specify a value for this parameter—it has no default value. You can calculate the minimum width of a panel as

```
5 * FontInfo.widMax + 42
```

`height` The height, in points, that you want to use for the panel. You must specify a value for this parameter—it has no default value. You can calculate the minimum height of a panel as

```
Max(64, 3 * FontHeight) + fontHeight + 18
```

where

```
fontHeight = FontInfo.ascent + FontInfo.descent +
             FontInfo.Leading.
```

DESCRIPTION

The `SDPSizePanel` function resizes the panel to have the specified width and height (keeping the upper-left corner in a fixed position). You set the original size of the panel with the `bounds` parameter in the `SDPNewPanel` function or `SDPGetNewPanel` function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$006C

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

You use the `SDPNewPanel` function (page 4-30) or `SDPGetNewPanel` function (page 4-34) to set the original size of the panel.

You can use the `GetFontInfo` routine described in *Inside Macintosh: Imaging With QuickDraw*, to determine the font size.

SDPDisposePanel

The `SDPDisposePanel` function deallocates all of the data structures associated with a panel.

```
pascal OSErr SDPDisposePanel (SDPPanelHandle panel);
```

`panel` The handle of the panel you wish to deallocate.

DESCRIPTION

Call this function when you are finished using a panel.

SPECIAL CONSIDERATIONS

A personal-catalog reference number returned by the `SDPGetPath` function is no longer valid after you call the `SDPDisposePanel` function.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$0066

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

The `SDPGetPath` function (page 4-40) returns a personal-catalog reference number that is no longer valid after you dispose of the panel.

Call the `SDPNewPanel` function (page 4-30) or the `SDPGetNewPanel` function (page 4-34) to open a new panel.

Handling Events in a Catalog-Browsing Panel

The routines in this section let you find out what action the user has taken in the Catalog-Browsing panel (or Personal-Catalog panel) and handle the resulting event.

If an event takes place in the Catalog-Browsing panel, you call the first function in this section, `SDPPanelEvent`, to handle the event. You must also pass regular null events to the `SDPPanelEvent` function. This function returns a value that tells you if the user has double-clicked a record or has otherwise changed the selection in the panel. You can use the `SDPGetPanelSelectionSize` (page 4-57) and `SDPGetPanelSelection` (page 4-58) functions to determine what item the user has selected. If the user has double-clicked a record, you can use these functions to determine what record the user has double-clicked.

You must call the `SDPUpdatePanel` function (page 4-47) to handle update events.

If the user double-clicks any item other than a record, the panel opens the item and displays its contents. If you want to provide a method other than double-clicking for a user to open an item, such as an Open item in a menu or a button in your window, you can use the `SDPOpenSelectedItem` function (page 4-59) to do so.

If you are implementing menu items or controls in addition to those in the panel, you can use the `SDPGetPanelSelectionState` function (page 4-55) to determine what the user is doing in order to decide whether to enable your menu items or buttons and what their labels should be. For example, if the user has clicked a record once to highlight it, you can enable a button labeled Choose. If the user has highlighted a container, you can change the button to read Open.

SDPPanelEvent

The `SDPPanelEvent` function handles events intended for the panel.

```
pascal OSErr SDPPanelEvent (SDPPanelHandle panel,
                            const EventRecord *theEvent,
                            SDPPanelState *whatHappened);
```

`panel` The panel in which the event occurred.

`theEvent` The event record for the event.

`whatHappened`
A return value that tells you how the `SDPPanelEvent` function handled the event.

DESCRIPTION

Because the panel is a modeless dialog box, the Event Manager passes all events that occur in the panel to you. If you determine that the event occurred in the panel, you can use the `SDPPanelEvent` function to handle the event.

You should use this function to handle mouse-up and mouse-down events in the panel, any key-down and auto-key events you want the panel to process, disk-insert events for which the panel is the intended target, and activate events for the panel's window. It treats suspend and resume events like activate events and all other events as null events. You must call the `SDPUpdatePanel` function to handle update events.

Before you call the `SDPPanelEvent` function for a key-down event, you should provide the appropriate feedback to the user. For example, if you provide a button labeled Open for opening an item in the panel and the user presses the Return key, you should highlight the Open button before calling the `SDPPanelEvent` function.

You must regularly provide the panel with null events so that it can update the scrollable list when the user moves up or down in the catalog hierarchy and so it can enumerate catalogs when appropriate to do so.

The whatHappened parameter can have any of the following values:

```
enum {
   kSDPProcessed,
   kSDPSelectedAnItem,
   kSDPChangedSelection
};

typedef unsigned short SDPPanelState;
```

Constant descriptions

kSDPProcessed The event resulted in no state change.

kSDPSelectedAnItem
The user wants to select the currently highlighted record. The function returns this value, for example, when a user double-clicks a record. When you are displaying the Personal-Catalog panel, the SDPPanelEvent function also returns this value when the user double-clicks an alias of a container.

kSDPChangedSelection
This is a good time to check the state of the panel. The function returns this value when item selected in the panel has changed because the user clicked a new item (which may mean that no item is selected), clicked in a menu, pressed a Command-key combination, or inserted a disk. The function also returns this value from the first null event you send to a panel after creating the panel.

If you are displaying the Personal-Catalog panel and the user attempts to open an alias of a container, the panel does not open the container, because the contents of that container are not within the user's default personal catalog. In that case, the SDPPanelEvent function returns the value kSDPSelectedAnItem in the whatHappened parameter. You can distinguish this case from that of the user selecting a record by calling the SDPGetPanelSelectionState function, which returns the value kSDPRecordSelected or kSDPRecordAliasSelected if the user selected a record or the alias of a record, but returns the value kSDPContainerAliasSelected if the user selected the alias of a container.

To display the contents of a container when the user tries to open the alias of a container in the Personal-Catalog panel, you can call the SDPGetPanelSelectionSize and SDPGetPanelSelection functions to get the location of the container, then close the Personal-Catalog panel and open the Catalog-Browsing panel. Use the location of the container as the value of the initialRLI parameter when you call the SDPNewPanel function or the SDPGetNewPanel function to open the new panel. Alternatively, you can get faster performance at the expense of more memory use by keeping both a Personal-Catalog panel and a Catalog-Browsing panel open and hiding the one not in use. Then when the user attempts to open the alias of a container in the Personal-Catalog panel, you can hide the Personal-Catalog panel, display the Catalog-Browsing panel, and use the SDPSetPath function to display the contents of the appropriate container.

SPECIAL CONSIDERATIONS

You cannot assume that a personal-catalog reference number returned by the `SDPGetPath` function is valid after you call the `SDPPanelEvent` function.

You must call the `SDPUpdatePanel` function to handle update events.

Be sure to pass mouse-down and mouse-up events to the `SDPPanelEvent` function. If you don't do so, the panel cannot respond to double clicks.

The `SDPPanelEvent` function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0006	$0071

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Call the `SDPUpdatePanel` function (page 4-47) to handle update events.

Use the `SDPGetPanelSelectionState` function (described next) to find out what the user has selected in the panel.

Call the `SDPGetPanelSelectionSize` function (page 4-57) to determine the size of the packed `DSSpec` structure of a selected item in the panel and the `SDPGetPanelSelection` function (page 4-58) to get a pointer to the packed `DSSpec` structure.

Call the `SDPNewPanel` function (page 4-30) or the `SDPGetNewPanel` function (page 4-34) to open a new panel. Call the `SDPHidePanel` (page 4-43) and `SDPShowPanel` (page 4-44) functions to hide and display open panels.

Use the `SDPSetPath` function (page 4-38) to display the contents of a specific container in an open panel.

The `SDPGetPath` function (page 4-40) returns a personal-catalog reference number that you can no longer assume to be valid after you call the `SDPPanelEvent` function.

SDPGetPanelSelectionState

The `SDPGetPanelSelectionState` function tells you what the user has selected in the panel.

```
pascal OSErr SDPGetPanelSelectionState (SDPPanelHandle panel,
                                        SDPSelectionState *itsState);
```

`panel` The panel on which you want this function to act.

`itsState` The state of the panel.

DESCRIPTION

The parameter `itsState` can return any of the following values:

```
enum {
   kSDPNothingSelected,
   kSDPLockedContainerSelected,
   kSDPContainerSelected,
   kSDPRecordSelected,
   kSDPRecordAliasSelected,
   kSDPContainerAliasSelected
};

typedef unsigned short SDPSelectionState;
```

Constant descriptions

`kSDPNothingSelected`
Nothing is currently selected.

`kSDPLockedContainerSelected`
The user has selected a container (a volume, folder, AOCE catalog, dNode, or personal catalog), but doesn't have access privileges.

`kSDPContainerSelected`
The user has selected a volume, folder, AOCE catalog, dNode, or personal catalog.

`kSDPRecordSelected`
The user has selected a record.

`kSDPRecordAliasSelected`
The user has selected an alias of a record.

`kSDPContainerAliasSelected`
The user has selected an alias of a container.

If the user has selected a record or the alias of a record, you can use the SDPGetPanelSelectionSize function to determine the size of the PackedDSSpec structure and the SDPGetPanelSelection function to get the PackedDSSpec structure for that record. If the SDPGetPanelSelectionState function returns the value kSDPContainerSelected or kSDPLockedContainerSelected, you can use these functions to determine which container the user has selected.

If you are displaying the Catalog-Browsing panel and the user selects an alias of a container, the SDPGetPanelSelectionState function returns the value kSDPContainerAliasSelected, but otherwise the panel behaves as if the user had selected the container itself. However, if you are displaying the Personal-Catalog panel and the user attempts to open an alias of a container, the panel does not open the container. In this case, you can call the SDPGetPanelSelectionSize and SDPGetPanelSelection functions to get the location of the container, then close (or hide) the Personal-Catalog panel and open (or show, if it's already open) the Catalog-Browsing panel to display the contents of that container.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$006E

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

See the description of the SDPPanelEvent function (page 4-52) for more information on handling user selections in panels.

Call the SDPGetPanelSelectionSize function (page 4-57) to determine the size of the packed DSSpec structure of a selected record and the SDPGetPanelSelection function (page 4-58) to get a pointer to the packed DSSpec structure.

Use the SDPNewPanel function (page 4-30) or SDPGetNewPanel function (page 4-34) to open a new panel.

SDPGetPanelSelectionSize

The `SDPGetPanelSelectionSize` function returns the size of the `PackedDSSpec` structure containing the packed record ID of the currently selected record or container.

```
pascal OSErr SDPGetPanelSelectionSize (SDPPanelHandle panel,
                                       unsigned short *dsSpecSize);
```

`panel` The panel on which you want this function to act.

`dsSpecSize`
A pointer to the size of the `PackedDSSpec` structure containing the record ID and location information for the record or container (dNode, AOCE catalog, personal catalog, volume, or folder) that the user has selected.

DESCRIPTION

If the `SDPGetPanelSelectionState` function returns the value `kRecordSelected` or `kSDPRecordAliasSelected`, you can use the `SDPGetPanelSelectionSize` and `SDPGetPanelSelection` functions to determine which record the user selected. If the `SDPGetPanelSelectionState` function returns the value `kContainerSelected`, `kLockedContainerSelected`, or `kContainerAliasSelected`, these functions tell you which container the user has selected.

You can use the length returned in the `dsSpecSize` parameter to allocate a buffer for a `PackedDSSpec` structure. You can then use a pointer to this buffer for the `selection` parameter when you call the `SDPGetPanelSelection` function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0072

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Use the SDPGetPanelSelectionState function (page 4-55) to determine what (a record, container, locked container, or nothing) the user has selected in the panel.

Use the SDPGetPanelSelection function, described next, to get the DSSpec structure identifying the record or container the user has selected.

SDPGetPanelSelection

The SDPGetPanelSelection function returns a DSSpec structure for the record or container the user has selected in the panel.

```
pascal OSErr SDPGetPanelSelection (SDPPanelHandle panel,
                                   PackedDSSpec *selection);
```

panel	The panel on which you want this function to act.
selection	A pointer to a PackedDSSpec structure, which contains location information for a volume, folder, dNode, personal catalog, or record. You must allocate memory and provide this pointer before you call the function.

DESCRIPTION

If the SDPGetPanelSelectionState function returns the value kRecordSelected, you can use the SDPGetPanelSelection function to determine which record the user selected. If the SDPGetPanelSelectionState function returns the value kContainerSelected or kLockedContainerSelected, the user has selected a volume, folder, AOCE catalog, dNode, or personal catalog, and the SDPGetPanelSelection function tells you the location of that container.

If the SDPGetPanelSelectionState function returns the value kRecordAliasSelected, the SDPGetPanelSelection function returns the DSSpec structure for the record referred to by the alias, not for the alias itself. Similarly, if the SDPGetPanelSelectionState function returns the value kContainerAliasSelected, the SDPGetPanelSelection function tells you the location of the container referred to by the alias.

Before you call the SDPGetPanelSelection function, you can call the SDPGetPanelSelectionSize function to get the size of the PackedDSSpec structure and use it to allocate the buffer for the selection parameter.

SPECIAL CONSIDERATIONS

The pathname of a container returned by the SDPGetPath function is not necessarily the same as the pathname you get by using the SDPGetPanelSelection function to obtain location information for an item the user has selected in that container. This discrepancy results from the fact that the item might be an alias or information card. The

SDPGetPanelSelection function returns the location of the original item, not that of the alias or information card itself.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$006F

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

Use the SDPGetPanelSelectionState function (page 4-55) to determine whether the user has selected a record, a container, a locked container, or an alias.

Use the SDPGetPanelSelectionSize function (page 4-57) to determine the size of the packed DSSpec structure.

The SDPGetPath function (page 4-40) returns the current pathname of the container in the pop-up menu of the panel.

SDPOpenSelectedItem

The SDPOpenSelectedItem function simulates a double-click on an item that the user has selected in the panel.

```
pascal OSErr SDPOpenSelectedItem (SDPPanelHandle panel,
                                  SDPPanelState *whatHappened);
```

panel The panel on which you want this function to act.

whatHappened
The result of the double click. This parameter can return the value kSDPProcessed, kSDPSelectedAnItem, or kSDPChangedSelection.

DESCRIPTION

If the user has selected a container (that is, a volume, folder, AOCE catalog, personal catalog, or node), then the SDPOpenSelectedItem function opens that container and returns the value kSDPChangedSelection in the whatHappened parameter. If the user has selected a record or an alias of a record, then the function returns the value

kSDPSelectedAnItem in this parameter. If the user has selected an alias of a container in the Catalog-Browsing panel, the SDPOpenSelectedItem function returns the value kSDPChangedSelection and opens that container just as if the user had selected the container itself. However, if the user has selected an alias of a container in the Personal-Catalog panel, the function returns the value kSDPSelectedAnItem and does not open the container. If the user has selected a locked container or there is no item selected, the function returns kSDPProcessed.

The Catalog-Browsing panel allows the user to double-click an item but provides no other user interface for opening a container or choosing a record. You can use the SDPOpenSelectedItem function to provide a way for the user to open a container or choose a record in the panel without double-clicking. For example, you can provide a button adjacent to the panel and call the SDPOpenSelectedItem function when the user clicks the button. To implement such a feature, you should call the SDPGetPanelSelectionState function each time either the SDPOpenSelectedItem function or the SDPPanelEvent function returns the value kSDPChangedSelection. If the user has selected a container or an alias of a container, you can enable the Open button. If there is no item selected or the user has selected a locked container, you can disable the button. If the user has selected a record, you can change the button to read “Choose” or whatever is appropriate for your application.

If the SDPOpenSelectedItem function returns the value kSDPSelectedAnItem for the Catalog-Browsing panel, you can assume the item is a record and you can use the SDPGetPanelSelectionSize function to determine the size of the packed DSSpec structure and the SDPGetPanelSelection function to get the DSSpec for that record. For the Personal-Catalog panel, you must first determine whether the item is a record or an alias of a container. If it’s a record, you can get the DSSpec structure just as in the case of the Catalog-Browsing panel. If the item is an alias of a container, you can switch to a Catalog-Browsing panel to display the contents of the container.

If the SDPOpenSelectedItem function returns the value kSDPChangedSelection, you should call the SDPGetPanelSelectionState function to determine how to label your buttons and whether to enable them. If the function returns kSDPProcessed, you need take no further action.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$006D

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

Call the SDPPanelEvent function (page 4-52) for mouse-up and mouse-down events in the panel to find out if the user changed the selection.

Call the SDPGetPanelSelectionSize function (page 4-57) to determine the size of the packed DSSpec structure of a selected item and the SDPGetPanelSelection function (page 4-58) to get a handle to the packed DSSpec structure.

Creating, Displaying, and Disposing of a Find Panel

The routines in this section create and display a Find panel. You can call the SDPNewFindPanel function (described next) to create a new Find panel.

The other functions in this section, as their names suggest, allow you to hide, show, enable, and disable a Find panel, redraw a Find panel, and move a Find panel within your window. When you are finished using a Find panel or close the window, call the SDPDisposeFindPanel function (page 4-75).

Once you have placed a Find panel in your window, you use the functions in "Handling Events in a Find Panel," beginning on page 4-75, to handle events that take place within the Find panel.

SDPNewFindPanel

The SDPNewFindPanel function creates a new Find panel in the location you specify in the window you specify.

```
pascal OSErr SDPNewFindPanel (SDPFindPanelHandle *newPanel,
                  WindowPtr window,
                  Point upperLeft,
                  short layoutResourceID,
                  Boolean visible,
                  Boolean enabled,
                  const RStringPtr *typesList,
                  long typeCount,
                  DirMatchWith matchTypeHow,
                  AuthIdentity identity,
                  short simultaneousSearchCount,
                  SDPFindPanelFocus initialFocus,
                  long refCon);
```

`newPanel` The address of the handle to the new Find panel created by the function. The `SDPNewFindPanel` function allocates this handle.

`window` A pointer to the window into which the Find panel is to be inserted.

`upperLeft` The upper-left corner, in the window's local coordinates, of the Find panel.

`layoutResourceID`
The resource ID of a Find-panel layout resource (type `kSDPFindPanelResourceType`) that specifies the layout of the Find panel. You can specify `kStandardFindLayout` as the resource ID to use a standard layout for the Find panel.

`visible` A Boolean value that specifies whether the Find panel is to be initially visible. Set this parameter to `true` if you want the Find panel to be visible. You can use the `SDPShowFindPanel` and `SDPHideFindPanel` functions to hide and show a Find panel.

`enabled` A Boolean value that specifies whether the Find panel is to be initially enabled. Set this parameter to `true` to enable the Find panel. You can use the `SDPEnableFindPanel` function to enable and disable the Find panel.

`typesList` A pointer to an array, each item of which is of type `RStringPtr`, providing a list of the types of records you want the Find panel to display. The Find panel displays only records of the types you specify in this list. The `matchTypeHow` parameter specifies how the `SDPNewFindPanel` function interprets the types list.

`typeCount` The number of record types in your types list.

`matchTypeHow`
A constant that specifies how the function interprets the types list. The possible values for this parameter are described in the following function description.

`identity` The authentication identity of the caller. Specify 0 for guest access.

`simultaneousSearchCount`
The number of catalog searches that may be done simultaneously. If you specify a larger number for this parameter, the searches are generally faster but require more memory. Each search requires 4600 bytes. The function does not use more memory than is necessary; for example, if you specify 10 for this parameter but there are only two catalogs to search, the function uses only the memory necessary to search those two catalogs. Note that the Find panel can perform simultaneous searches of catalogs only; it cannot search more than one volume at a time.

`initialFocus`
A constant that specifies whether there should be a focus rectangle in the Find panel, and if so, whether it should be around the scrolling list in the Find panel or around the text-input field (the Find field; see Figure 4-5 on page 4-18). You can use the `SDSetFindPanelFocus` function to change the location of the focus rectangle.

`refCon` A reference constant. The function places this value in the `refCon` field of the `SDPPanelRecord` structure pointed to by the `newPanel` parameter. You can use the `refCon` parameter for whatever you wish.

DESCRIPTION

You use the `SDPNewFindPanel` function to create a new Find panel in your window. You specify the location of the Find panel and can specify that it is to be initially visible or invisible and initially enabled or disabled.

The `typesList` and `matchTypeHow` parameters let you specify what types of records the Find panel should display. The possible values for the `matchTypeHow` parameter are as follows:

```
enum {
   kMatchAll,
   kExactMatch,
   kBeginsWith,
   kEndingWith,
   kContaining
};
```

Constant descriptions

`kMatchAll`	Display all record types. The function ignores the `typesList` and `typeCount` parameters.
`kExactMatch`	Display only the record types in the types list.
`kBeginsWith`	Display only records whose types begin with the string in the `typesList` parameter. The `typesList` parameter can contain only one string; if the `typeCount` parameter is not equal to 1, the function returns an error.
`kEndingWith`	Display only records whose types end with the string in the `typesList` parameter. The `typesList` parameter can contain only one string; if the `typeCount` parameter is not equal to 1, the function returns an error.
`kContaining`	Display only records whose types contain the string in the `typesList` parameter. The `typesList` parameter can contain only one string; if the `typeCount` parameter is not equal to 1, the function returns an error.

You can use the `SDPGetCategories` and `SDPGetCategoryTypes` functions to determine what record types are available and use that information to let the user select which types of records the Find panel should display. Then you can use the `typesList` parameter to restrict the panel to those record types.

In the `layoutResourceID` parameter you provide the resource ID of a Find-panel layout resource, defined by the following Rez type:

```
type 'find' {
   pstring;       /* "Find" text */
   align word;
   pstring;       /* "Search" text */
   align word;
```

```
   array [5] {    /* specifications for text-entry box label,
                     search menu label, text-entry box,
                     search menu, scrolling list */
      integer sysFont, appFont, portFont;
      integer;    /* face */
      integer;    /* size */
      rect;       /* bounds */
   };
};
#define kSDPFindPanelResourceType   'find'
```

Field descriptions

`"Find" text` The label for the text-entry field. You must supply a value for this field; there is no default label.

`"Search" text` The label for the search menu. You must supply a value for this field; there is no default label.

`array` An array of specifications for the elements of the Find panel in the following order: The label for the text-entry box, the label for the search menu, the text-entry box, the search menu, and the scrolling list. The specifications include the following information:

- `font` A choice of system font, application font, or graphics port font.
- `face` The QuickDraw font style.
- `size` The type size, in points.
- `bounds` The boundary, in local coordinates, of the element.

Subsequently, you can use the other routines in this and the following section to hide and show the Find panel, enable and disable it, and handle events that occur within the Find panel.

You may specify any of the following values for the `initialFocus` parameter:

```
enum {
   kSDPFindPanelNoFocus,
   kSDPFindPanelListHasFocus,
   kSDPFindPanelTextHasFocus
};

typedef unsigned short SDPFindPanelFocus;
```

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0014	$08FC

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

You can use the `SDPHideFindPanel` function (page 4-67) and `SDPShowFindPanel` function (page 4-68) to hide and show a Find panel.

You can use the `SDPEnableFindPanel` function (page 4-69) to enable and disable the Find panel.

You can use the `SDSetFindPanelFocus` function (page 4-69) to change the location of the focus rectangle.

Use the other routines in this section to dispose, draw, and move Find panels. Use the routines in "Handling Events in a Find Panel," beginning on page 4-75, to handle events that occur in the Find panel and to determine what record the user selected.

You can use the `SDPPromptForID` function (page 4-25) to get a specific or local identity.

You can use the `SDPGetCategories` function (page 4-91) to obtain a list of all the record-type categories currently available and the `SDPGetCategoryTypes` function (page 4-92) to list all the types of records included in a specific category.

SDPInstallFindPanelBusyProc

The `SDPInstallFindPanelBusyProc` function installs an application-defined routine that the Find panel calls when it is busy.

```
pascal OSErr SDPInstallFindPanelBusyProc (
                                    SDPFindPanelHandle findPanel,
                                    FindPanelBusyProc busyProc);
```

`findPanel` A handle for the Find panel for which you want to install a Find-panel-busy callback routine.

`busyProc` A pointer to your callback routine. To remove a callback routine that you installed previously, specify `nil` for this parameter.

DESCRIPTION

If you use the `SDPInstallFindPanelBusyProc` function to install a Find-panel-busy callback routine, the Find panel calls your routine whenever the panel is busy. For

example, if the user initiates a search, the Find panel calls your callback routine while it conducts the search.

The Find panel passes to your callback routine the handle to the panel and a Boolean value that indicates whether the panel is busy. You can use your callback routine to provide feedback to the user that indicates that the Find panel is busy. One good way to do this is to display the Standard Catalog Package's spinning arrow icon.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$090C

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

The Find-panel-busy callback routine is described on page 4-95.

SDPSetFindPanelBalloonHelp

The `SDPSetFindPanelBalloonHelp` function enables Balloon Help for the Find panel.

```
pascal OSErr SDPSetFindPanelBalloonHelp (
                                    SDPFindPanelHandle findPanel,
                                    short balloonHelpID);
```

`findPanel` A handle for the panel for which you want to enable Balloon Help.

`balloonHelpID`
The resource ID of a `'STR#'` resource that contains Balloon Help strings for the Find panel.

DESCRIPTION

You must use the `SDPSetFindPanelBalloonHelp` function to provide text strings for Balloon Help and to activate Balloon Help for the Find panel. You must provide three text strings, one for each element of the Find panel, in the following order:

1. the text-input field (the Find field)
2. the search menu
3. the scrolling list

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0003	$090A

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Balloon Help is described in the chapter “Help Manager” of *Inside Macintosh: More Macintosh Toolbox*.

SDPHideFindPanel

The `SDPHideFindPanel` function hides a Find panel.

```
pascal OSErr SDPHideFindPanel (SDPFindPanelHandle findPanel);
```

`findPanel` The Find panel you wish to hide.

DESCRIPTION

If a Find panel is visible, this function makes it invisible by hiding the menu, erasing and invalidating the list’s rectangles for the list and text fields, and causing the panel to not be drawn. If the Find panel is already hidden, this function does nothing.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$0903

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Use the `SDPShowFindPanel` function (described next) to display a Find panel that you have hidden.

SDPShowFindPanel

The `SDPShowFindPanel` function displays a Find panel.

```
pascal OSErr SDPShowFindPanel (SDPFindPanelHandle findPanel);
```

`findPanel` The Find panel you wish to display.

DESCRIPTION

You can hide a Find panel by setting the `visible` parameter in the `SDPNewFindPanel` to `false` or by calling the `SDPHideFindPanel` function. If the Find panel is hidden, the `SDPShowFindPanel` function makes it visible. If the Find panel is already visible, this function does nothing. Use the `SDPUpdateFindPanel` function to handle an update event.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$0902

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

When you create a Find panel with the `SDPNewFindPanel` function (page 4-61), you can specify that it be initially hidden.

Use the `SDPHideFindPanel` function (page 4-67) to hide a Find panel.

Use the `SDPUpdateFindPanel` function (page 4-72) to handle an update event.

SDPEnableFindPanel

The `SDPEnableFindPanel` function enables or disables a Find panel.

```
pascal OSErr SDPEnableFindPanel (SDPFindPanelHandle findPanel,
                                 Boolean enabled);
```

`findPanel` The Find panel you wish to enable or disable.

`enabled` A Boolean value that specifies whether you wish to enable or disable the Find panel. Specify `true` to enable the Find panel.

DESCRIPTION

This function disables the list, menu, and text field so that they do not accept any commands or text, or enables a disabled Find panel.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0003	$0905

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

When you call the SDPNewFindPanel function (page 4-61) to create a Find panel, you specify whether it should be enabled or disabled initially.

SDPSetFindPanelFocus

The SDPSetFindPanelFocus function changes the focus rectangle for a Find panel.

```
pascal OSErr SDPSetFindPanelFocus (SDPFindPanelHandle findPanel,
                                   SDPFindPanelFocus newFocus);
```

findPanel The Find panel for which you wish to change the focus rectangle.

newFocus A constant that specifies whether there should be a focus in the Find panel, and if so, whether it should be the scrolling list in the Find panel or the text-input field (the Find field; see Figure 4-5 on page 4-18).

DESCRIPTION

When the user uses the Tab key to select the Find panel (assuming you support this feature), you can use the SDPSetFindPanelFocus function to specify which portion of the Find panel (the scrolling list or the text-entry field) is the focus; that is, to which portion of the Find panel any keyboard equivalents are applied. If the scrolling list is the focus, the Find panel draws a heavy border (called a *focus rectangle*) around the list. If the text-entry field is the focus, the Find panel displays a blinking insertion point in the field.

If you want to integrate the action of the focus in the Find panel with one in your application window, you can use the SDPSetFindPanelFocus function to control whether the focus is in the Find panel and which element has the focus.

For example, if you have three text-entry fields in your application window and the user presses the Tab key repeatedly, you can use this function to cycle the focus rectangle sequentially through your three fields, then to the text-entry field in the Find panel, then the scrolling list, and then back to the first of your text fields.

To implement this feature, you must interpret events before calling the SDPFindPanelEvent function. Otherwise, when the user is working in the Find panel and you call the SDPFindPanelEvent function in response to a press of the Tab key, the Find panel toggles the focus between the text-entry field and the scrolling list.

You may specify any of the following values for the `newFocus` parameter:

```
enum {
   kSDPFindPanelNoFocus,
   kSDPFindPanelListHasFocus,
   kSDPFindPanelTextHasFocus
};
typedef unsigned short SDPFindPanelFocus;
```

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0003	$0906

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

When you call the `SDPNewFindPanel` function (page 4-61) to create a Find panel, you specify the initial focus rectangle for the Find panel.

You should call the `SDPFindPanelEvent` function (page 4-76) to handle events that occur in your window when the Find panel is open.

SDPSetFindIdentity

The `SDPSetFindIdentity` function changes the authentication identity used by the Find panel.

```
pascal OSErr SDPSetFindIdentity (SDPFindPanelHandle findPanel,
                                 AuthIdentity identity)
```

`findPanel` The Find panel for which you want to change the caller's authentication identity.

`identity` The new authentication identity. Specify 0 for guest access.

DESCRIPTION

You can use the `SDPSetFindIdentity` function to change the authentication identity of the user without closing the Find panel and opening a new one.

SPECIAL CONSIDERATIONS

If a search is in progress when you call the `SDPSetFindIdentity` function, the Find panel continues to use the old identity until the search is complete.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$090B

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

The `SDPPromptForID` function (page 4-25) prompts the user for a password and returns an authentication identity.

SDPUpdateFindPanel

The `SDPUpdateFindPanel` function draws all or part of a Find panel.

```
pascal OSErr SDPUpdateFindPanel (SDPFindPanelHandle findPanel,
                                 RgnHandle theRgn);
```

`findPanel` The Find panel you wish to draw.

`theRgn` The region of the window you wish to draw. Any component of the Find panel that intersects this region is redrawn. For example, if any portion of the list of the Find panel intersects this region, the Standard Catalog Package redraws the whole list. Pass `nil` in this parameter to draw the entire Find panel.

DESCRIPTION

You must call the `SDPUpdateFindPanel` function to draw a Find panel in response to an update event. (Do not pass an update event to the `SDPFindPanelEvent` function.)

You can also use the SDPUpdateFindPanel function to draw a Find panel when your application needs to redraw its window because of some activity other than an update event.

In response to an update event, you should first call the BeginUpdate routine, which takes a window pointer as a parameter. Then use the value in the visRgn field of the window's grafPort structure for the theRgn parameter of the SDPUpdateFindPanel function.

Note that the SDPUpdateFindPanel function does not use the region specified by the parameter theRgn as a clipping region. To clip the drawing region, you must first call the SetClip routine. (Remember to save the old clipping region so that you can restore it when you have finished clipping.)

If you have hidden the Find panel (either by setting the visible parameter in the SDPNewFindPanel function to false or by calling the SDPHideFindPanel function), then the SDPUpdateFindPanel function does not display the Find panel. To display a hidden Find panel, you must call the SDPShowFindPanel function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0901

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

Use the SDPShowFindPanel function (page 4-68) to display a Find panel that you have hidden with the SDPNewFindPanel function (page 4-61) or SDPHideFindPanel function (page 4-67).

Use the BeginUpdate routine to begin the process of updating your window. The chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* describes how to respond to update events. The BeginUpdate routine is described in the chapter "Window Manager" in *Inside Macintosh: Macintosh Toolbox Essentials*.

Use the SetClip routine to set a clipping region. The SetClip routine is described in *Inside Macintosh: Imaging With QuickDraw*.

SDPMoveFindPanel

The `SDPMoveFindPanel` function moves the Find panel to a location you specify.

```
pascal OSErr SDPMoveFindPanel (SDPFindPanelHandle findPanel,
                               short h,
                               short v);
```

`findPanel`	The Find panel you wish to move.
`h`	The horizontal coordinate to which you want to move the upper-left corner of the Find panel, in the local coordinates of the Find panel's window.
`v`	The vertical coordinate to which you want to move the upper-left corner of the Find panel, in the local coordinates of the Find panel's window.

DESCRIPTION

Use the `SDPMoveFindPanel` function to move the upper-left corner of the Find panel to (h, v), given in local coordinates of the Find panel's window. You set the original location of the Find panel with the `upperLeft` parameter in the `SDPNewFindPanel` function.

If you have hidden the Find panel (either by setting the `visible` parameter in the `SDPNewFindPanel` function to `false` or by calling the `SDPHideFindPanel` function), then the `SDPMoveFindPanel` function does not display the Find panel. To display a hidden Find panel, you must call the `SDPShowFindPanel` function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0904

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

You use the `SDPNewFindPanel` function (page 4-61) to set the original location of the Find panel.

Use the SDPShowFindPanel function (page 4-68) to display a Find panel that you have hidden with the SDPNewFindPanel function (page 4-61) or SDPHideFindPanel function (page 4-67).

SDPDisposeFindPanel

The SDPDisposeFindPanel function deallocates all of the data structures associated with a Find panel.

```
pascal OSErr SDPDisposeFindPanel (SDPFindPanelHandle findPanel);
```

findPanel The handle of the Find panel you wish to deallocate.

DESCRIPTION

Call this function when you have completely finished using a Find panel.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$08FD

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

Call the SDPNewFindPanel function (page 4-61) to open a new Find panel.

Handling Events in a Find Panel

The routines in this section let you find out what action the user has taken in the Find panel and handle the resulting event. When you receive an event for a window in your application that contains a Find panel, you must first determine whether it took place in the panel. For mouse-down events, you can check the coordinates of the event to see if it was in the Find panel. You must keep track of where the user is working to know how to handle key-down events; for example, you can place a focus rectangle in the Find panel

or around the content portion of the window, according to the last location of a mouse-down event. Then you can assume that any key-down event pertains to the portion of the window inside the focus rectangle.

If an event takes place in the Find panel, you call the first function in this section, `SDPFindPanelEvent`, to handle the event. You must also pass regular null events to the `SDPFindPanelEvent` function. This function returns a value that tells you how the function handled the event. If the user has double-clicked a record, for example, the function returns the value `kSDPSelectedAFindItem` and you can call the `SDPGetFindPanelSelectionSize` (page 4-80) and `SDPGetFindPanelSelection` (page 4-82) functions to determine what record the user has selected.

You must call the `SDPUpdateFindPanel` function (page 4-72) to handle update events.

You can use the `SDPStartFind` function (page 4-83) to implement a Find button. If you do so, you should also provide a Cancel button, which you can implement by calling the `SDPStopFind` function (page 4-84).

If you are implementing menu items or controls in addition to those in the Find panel, you can use the `SDPGetFindPanelState` function (page 4-79) to determine what the user is doing in order to decide whether to enable your menu items or buttons and what their labels should be. For example, if the user has clicked a record once to highlight it, you can enable a button labeled Choose.

SDPFindPanelEvent

The `SDPFindPanelEvent` function handles events intended for the Find panel.

```
pascal OSErr SDPFindPanelEvent (SDPFindPanelHandle findPanel,
                                const EventRecord *event,
                                SDPFindPanelResult *whatHappened);
```

`findPanel` The Find panel in which the event occurred.

`event` The event record for the event.

`whatHappened`
A return value that tells you how the `SDPFindPanelEvent` function handled the event.

DESCRIPTION

Because the Find panel is in your window, the Event Manager passes all events that occur in the Find panel to you. If you determine that the event occurred in the Find panel, you can use the `SDPFindPanelEvent` function to handle the event.

You should use this function to handle mouse-up and mouse-down events in the Find panel, any key-down and auto-key events you want the Find panel to process, and activate events for the Find panel's window. It treats all other events as null events. You must call the `SDPUpdateFindPanel` function to handle update events.

Before you call the SDPFindPanelEvent function for a key-down event, you should provide the appropriate feedback to the user. For example, if you provide a button labeled Open for opening an item in the Find panel and the user presses the Return key, you should highlight the Open button before calling the SDPFindPanelEvent function.

You must regularly provide the Find panel with null events so that it can search for the records the user specifies.

The whatHappened parameter can have any of the following values:

```
enum {
   kSDPSelectedAFindItem,
   kSDPFindSelectionChanged,
   kSDPFindCompleted,
   kSDPTextStateChanged,
   kSDPFocusChanged,
   kSDPSelectionAndFocusChanged,
   kSDPMenuChanged,
   kSDPSelectionAndMenuChanged,
   kSDPProcessedFind
};
typedef unsigned short SDPFindPanelResult;
```

Constant descriptions

kSDPSelectedAFindItem
: The user wants to choose a record. The function returns this value, for example, when a user double-clicks a record or clicks a record once and presses Return or Enter.

kSDPFindSelectionChanged
: The item selected in the Find panel has changed because the user clicked a new item (which may mean that no item is selected), pressed an arrow key, or typed the beginning letters of the name of the record (“type-selecting” the record).

kSDPFindCompleted
: The Find panel has completed a search. When the Find text-entry field is the focus and the user presses the Return or Enter key, or when you call the SDPStartFind function, the Find panel starts a search. Each time you pass an event (including null events) to the SDPFindPanelEvent function, the Find panel continues the search for a short time and then returns control to your application. The first time you call the SDPFindPanelEvent function after the search is complete, the function returns the value kSDPFindCompleted.

kSDPTextStateChanged
: The user has entered text in a previously empty Find field or has deleted all text from the field.

`kSDPFocusChanged`
: The focus has moved from the Find text field to the scrolling list or vice versa.

`kSDPSelectionAndFocusChanged`
: The user has changed the focus rectangle to the scrolling list by clicking a new item.

`kSDPMenuChanged`
: The user has changed the item chosen in the Search menu. The Search menu lists the catalogs and volumes available for searching.

`kSDPSelectionAndMenuChanged`
: The user has changed both the item chosen in the search menu and the item selected in the scrolling list.

`kSDPProcessedFind`
: The event resulted in no state change.

SPECIAL CONSIDERATIONS

The `SDPFindPanelEvent` function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0006	$0900

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Call the `SDPUpdateFindPanel` function (page 4-72) to handle update events.

Call the `SDPGetFindPanelSelectionSize` function (page 4-80) to determine the size of the packed `DSSpec` structure of a selected item and the `SDPGetFindPanelSelection` function (page 4-82) to get a pointer to the packed `DSSpec` structure.

You can call the `SDPStartFind` function (page 4-83) to initiate a search and the `SDPStopFind` function (page 4-84) to cancel one.

SDPGetFindPanelState

The SDPGetFindPanelState function tells you what action the user has taken in the Find panel.

```
pascal OSErr SDPGetFindPanelState (SDPFindPanelHandle findPanel,
                                   SDPFindPanelState *itsState);
```

`findPanel` The Find panel on which you want this function to act.

`itsState` The state of the Find panel.

DESCRIPTION

The Find panel (see Figure 4-5 on page 4-18) does not provide buttons or menu items to let the user choose records or initiate searches; you must provide these buttons and menu items. You can use the SDPGetFindPanelState function to determine what the user is doing in order to decide whether to enable your menu items or buttons and what their labels should be. For example, if the user has clicked a record to highlight it, you can enable a button labeled Choose. If the user has entered text in the Find field, you can enable a button labeled Find.

You can use the following bit masks to test the value returned in the itsState parameter:

```
enum {
   kSDPItemIsSelectedBit,
   kSDPFindTextExistsBit
};

/* values of SDPFindPanelState */
enum {
   kSDPItemIsSelectedMask = 1<<kSDPItemIsSelectedBit,
   kSDPFindTextExistsMask = 1<<kSDPFindTextExistsBit
};
typedef unsigned short SDPFindPanelState;
```

Constant descriptions

`kSDPItemIsSelectedMask`
The user has selected a record or the alias of a record.

`kSDPFindTextExistsMask`
The user has entered text in the Find field.

If the user has selected a record or the alias of a record, you can use the SDPGetFindPanelSelectionSize function to determine the size of the PackedDSSpec structure for the record and the SDPGetFindPanelSelection function to get the PackedDSSpec structure.

If the user has entered text in the Find field, you can activate a button labeled Find. If the user clicks your Find button, you should call the `SDPStartFind` function to start the search. You should also provide a Stop button, which you can implement by calling the `SDPStopFind` function. Also, if the user presses the Return or Enter key during a search, the Find panel stops the search. Therefore, you should highlight the Stop button during a search to indicate that it is the default button.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0907

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

See the description of the `SDPFindPanelEvent` function (page 4-76) for more information on handling user selections in panels.

Call the `SDPGetFindPanelSelectionSize` function (described next) to determine the size of the `PackedDSSpec` structure of a selected record and the `SDPGetFindPanelSelection` function (page 4-82) to get a pointer to the packed `DSSpec` structure.

SDPGetFindPanelSelectionSize

The `SDPGetFindPanelSelectionSize` function returns the size of the `PackedDSSpec` structure containing the packed record ID of the currently selected record in the Find panel.

```
pascal OSErr SDPGetFindPanelSelectionSize (
                                    SDPFindPanelHandle findPanel,
                                    unsigned short *size);
```

`findPanel` The Find panel on which you want this function to act.

`size` A pointer to the size of the `PackedDSSpec` structure containing the record ID and location information for the record that the user has selected.

DESCRIPTION

If the `SDPGetFindPanelState` function returns the value `kSDPItemIsSelectedMask`, you can use the `SDPGetFindPanelSelectionSize` and `SDPGetFindPanelSelection` functions to determine which record the user selected. If the user has selected an alias of a record, these functions tell you which record the alias is for.

You can use the length returned in the `dsSpecSize` parameter to allocate a buffer to hold a `PackedDSSpec` structure. You can then use a pointer to this buffer as the value of the `selection` parameter when you call the `SDPGetFindPanelSelection` function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0908

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Use the `SDPGetFindPanelState` function (page 4-79) to determine if the user has selected a record in the Find panel.

Use the `SDPGetFindPanelSelection` function, described next, to get the `DSSpec` structure identifying the record the user has selected.

SDPGetFindPanelSelection

The `SDPGetFindPanelSelection` function returns a `DSSpec` structure for the record the user has selected in the Find panel.

```
pascal OSErr SDPGetFindPanelSelection (
                                    SDPFindPanelHandle findPanel,
                                    PackedDSSpec *selection);
```

`findPanel` The Find panel on which you want this function to act.

`selection` A pointer to a `PackedDSSpec` structure, which contains location information for a record. You must allocate memory and provide this pointer before you call the function.

DESCRIPTION

If the `SDPGetFindPanelState` function returns the value `kSDPItemIsSelectedMask`, you can use the `SDPGetFindPanelSelection` function to determine which record the user selected.

If the user has selected the alias of a record, the `SDPGetFindPanelSelection` function returns the `DSSpec` structure for the record referred to by the alias, not for the alias itself.

Before you call the `SDPGetFindPanelSelection` function, you can call the `SDPGetFindPanelSelectionSize` function to get the size of the `PackedDSSpec` structure and use it to allocate the buffer for the `selection` parameter.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0909

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Use the `SDPGetFindPanelState` function (page 4-79) to determine whether the user has selected a record.

Use the `SDPGetFindPanelSelectionSize` function (page 4-80) to determine the size of the packed `DSSpec` structure.

SDPStartFind

The `SDPStartFind` function searches for the record or records the user specifies in the Find text field of the Find panel.

```
pascal OSErr SDPStartFind (SDPFindPanelHandle findPanel);
```

`findPanel` The Find panel on which you want this function to act.

DESCRIPTION

The Find panel allows the user to enter text in the Find field and then press Return or Enter to initiate a search but provides no button to start a search. The *Macintosh Human Interface Guidelines* suggest that you should provide clearly labeled buttons to let the user start and cancel searches. You can use the `SDPStartFind` function to provide a button (or other interface) that lets the user start a search for records.

The `SDPStartFind` function returns control to your application as soon as it initiates the search. Each time you call the `SDPFindPanelEvent` function, the Find panel continues the search for a short time before returning control to you. When it completes the search, the `SDPFindPanelEvent` function returns the value `kSDPFindCompleted` in the `whatHappened` parameter.

You can use the `SDPStopFind` function to cancel a search.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$08FE

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

Call the SDPFindPanelEvent function (page 4-76) to determine when the SDPStartFind function has completed a search.

Call the SDPStopFind function (described next) to cancel a search.

SDPStopFind

The SDPStopFind function cancels a search that you initiated with the SDPStartFind function.

```
pascal OSErr SDPStopFind (SDPFindPanelHandle findPanel);
```

findPanel The Find panel on which you want this function to act.

DESCRIPTION

When you use the SDPStartFind function to initiate a search for records, you should enable a Stop button. If the user clicks this button, you should call the SDPStopFind function to stop the search. Also, if the user presses the Return or Enter key during a search, the Find panel stops the search. Therefore, you should highlight the Stop button during a search to indicate that it is the default button.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$08FF

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

Call the `SDPStartFind` function (page 4-83) to initiate a search.

Resolving Aliases

The functions in this section resolve HFS and AOCE catalog-system aliases to catalog objects. The user can use the Finder to create an alias for any item in the catalog system, including records, dNodes, catalogs, personal catalogs, and so forth. If the alias is an HFS file, as in the case of a file-system alias for a personal catalog, you can use the `SDPResolveAliasFile` function (described next) to resolve the alias. If the alias is contained in a record, you can use the `SDPResolveAliasDSSpec` function (page 4-87) to resolve it.

SDPResolveAliasFile

The `SDPResolveAliasFile` function resolves a file-system alias of an AOCE catalog object.

```
pascal OSErr SDPResolveAliasFile (FSSpecPtr fileSpec,
                                  PackedDSSpecHandle resolvedDSSpec,
                                  AuthIdentity identity,
                                  Boolean mayPromptUser);
```

`fileSpec` The file specification structure of the alias file you wish to resolve.

`resolvedDSSpec`
A handle to a packed `DSSpec` structure that contains the resolved alias. You must allocate a handle of any size and provide it to the function. The function resizes the handle as necessary and uses it to return the resolved alias of you.

`identity` The authentication identity of the caller.

`mayPromptUser`
A Boolean value that specifies whether the function should present dialog boxes to the user as necessary. If you set this parameter to `true`, the function displays any dialog boxes that are necessary to complete the resolution of the alias; for example, the user might be requested to insert a disk or log on to a file server. If you set this parameter to `false`, the function does not present the user with any dialog boxes but returns with an error if it can't resolve the alias.

DESCRIPTION

A file containing an AOCE catalog-system alias has the `isAlias` bit set in the file's Finder flags field and has one of the following file types:

File type	Constant
`'pabt'`	`kPersonalCatalogFileType`
`'bust'`	`kBusinessCardFileType`
`'dirt'`	`kCatalogFileType`
`'dnod'`	`kDNodeFileType`
`'rcrd'`	`kRecordFileType`

If it is appropriate to present the user with dialog boxes, specify `true` for the `mayPromptUser` parameter to make sure that the function can resolve the alias. If your function is running in the background or for some other reason it is not appropriate to display dialog boxes, set this parameter to `false`.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0007	$0E74

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

HFS aliases and the Finder flags field are described in the chapter "Finder Interface" in the book *Inside Macintosh: Macintosh Toolbox Essentials.*

To resolve an alias in an AOCE catalog record, use the `SDPResolveAliasDSSpec` function, described next.

SDPResolveAliasDSSpec

The SDPResolveAliasDSSpec function resolves an alias of an AOCE catalog-system alias that is contained in a PackedDSSpec structure.

```
pascal OSErr SDPResolveAliasDSSpec (
                                PackedDSSpecHandle theAliasDSSpec,
                                AuthIdentity identity,
                                Boolean mayPromptUser);
```

theAliasDSSpec
A handle to the PackedDSSpec structure of the alias you wish to resolve. The function returns, in this same handle, the fully resolved alias of the record to which that alias refers.

identity The authentication identity of the caller.

mayPromptUser
A Boolean value that specifies whether the function should present dialog boxes to the user as necessary. If you set this parameter to true, the function displays any dialog boxes that are necessary to complete the resolution of the alias; for example, the user might be requested to log on to a catalog server. If you set this parameter to false, the function does not present the user with any dialog boxes but returns with an error if it can't resolve the alias.

DESCRIPTION

When the enumeration specification structure returned by the DirEnumerateGet function indicates that a record contains an alias, that record includes an attribute of type kAliasAttrTypeBody. The attribute value of such an attribute is a packed DSSpec structure. The SDPResolveAliasDSSpec function returns the PackedDSSpec structure of the fully resolved record to which that alias refers.

If it is appropriate to present the user with dialog boxes, specify true for the mayPromptUser parameter to make sure that the function can resolve the alias. If your function is running in the background or for some other reason it is not appropriate to display dialog boxes, set this parameter to false.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0005	$0E75

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

To resolve an HFS file containing an alias of an AOCE catalog object, use the `SDPResolveAliasFile` function (page 4-85).

Obtaining Icons and Lists of Catalog-Item Categories and Types

When you use the Catalog-Browsing panel or Find panel to let the user select records, you might want to display a list of types or categories of records that the user can search for, a list of records the user has selected, or other information about the catalog structure and contents. The functions in this section return icons for records and other AOCE catalog items (such as catalogs or dNodes) and provide lists of the record types and categories available.

SDPGetIconByType

The `SDPGetIconByType` function returns the icon suite for a specific type of record.

```
pascal OSErr SDPGetIconByType (const RString *recordType,
                               IconSelectorValue whichIcons,
                               Handle *iconSuite);
```

`recordType`
A pointer to the record type whose icon suite you want.

`whichIcons`
A selector mask that indicates which icons you want.

`iconSuite` A handle to the suite of icons you requested. The function allocates this handle. Use the Macintosh Toolbox icon utilities to handle these icons.

DESCRIPTION

If you want to display the icon for a record, you can use the `SDPGetIconByType` function to obtain one or more icons for the record. You must specify the types of icon resource you want and the type of record for which you want an icon.

To specify which icons you want for a record type, use the following mask values:

```
typedef unsigned long   IconSelectorValue;
#define svLarge1Bit     0x00000001
#define svLarge4Bit     0x00000002
#define svLarge8Bit     0x00000004
#define svSmall1Bit     0x00000100
#define svSmall4Bit     0x00000200
#define svSmall8Bit     0x00000400
#define svMini1Bit      0x00010000
#define svMini4Bit      0x00020000
#define svMini8Bit      0x00040000
#define svAllLargeData  0x000000ff
#define svAllSmallData  0x0000ff00
#define svAllMiniData   0x00ff0000
#define svAll1BitData   (svLarge1Bit | svSmall1Bit | svMini1Bit)
#define svAll4BitData   (svLarge4Bit | svSmall4Bit | svMini4Bit)
#define svAll8BitData   (svLarge8Bit | svSmall8Bit | svMini8Bit)
#define svAllAvailableData 0xffffffff
```

To dispose of the icon suite, call the icon utility routine `DisposeIconSuite`.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Unlike most Standard Catalog Package routines, the `SDPGetIconByType` function uses the $AA5C trap and does not require a parameter word count.

Trap	Routine selector
$AA5C	$0400

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Icons and icon resources are described in the chapter “Finder Interface” in *Inside Macintosh: Macintosh Toolbox Essentials.*

To dispose of the icon suite, call the icon utility routine `DisposeIconSuite`. The icon utilities are described in the chapter "Icon Utilities" in *Inside Macintosh: More Macintosh Toolbox*.

SDPGetDSSpecIcon

The `SDPGetDSSpecIcon` function returns the icon for any object in the AOCE catalog system.

```
pascal OSErr SDPGetDSSpecIcon (const PackedDSSpec *packedDSSpec,
                               IconSelectorValue whichIcons,
                               Handle *iconSuite);
```

`packedDSSpec`
A pointer to the specifier of the object whose icon you want.

`whichIcons`
A selector mask that indicates which icons you want.

`iconSuite` A handle to the suite of icons you requested. The function allocates this handle. Use the Macintosh Toolbox icon utilities to handle these icons.

DESCRIPTION

If you want to display an icon for a specific attribute, a record, or a container (volume, folder, AOCE catalog, dNode, or personal catalog) in the AOCE catalog system, you can use the `SDPGetDSSpecIcon` function to obtain the icon. You must specify the types of icon resource you want and the `DSSpec` structure for the item for which you want an icon. You can use the `SDPGetPanelSelection` or `SDPGetFindPanelSelection` functions to get the `DSSpec` structure for an item the user selected.

If the object you specify does not have an icon, the system returns the default icon for objects of that type.

To dispose of the icon suite, call the icon utility routine `DisposeIconSuite`.

The selector masks for the `whichIcons` parameter are described on page 4-89.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Unlike most Standard Catalog Package routines, the `SDPGetDSSpecIcon` function uses the $AA5C trap and does not require a parameter word count.

Trap	Routine selector
$AA5C	$0401

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

You can obtain a record type from the `DSSpec` structure returned by the `SDPGetPanelSelection` function (page 4-58) or the `SDPGetFindPanelSelection` function (page 4-82).

The routines for expanding `DSSpec` structures are described in the chapter "AOCE Utilities" in this book.

Icons and icon resources are described in the chapter "Finder Interface" in the book *Inside Macintosh: Macintosh Toolbox Essentials.*

To dispose of the icon suite, call the icon utility routine `DisposeIconSuite`. The icon utilities are described in the chapter "Icon Utilities" in *Inside Macintosh: More Macintosh Toolbox*.

SDPGetCategories

The `SDPGetCategories` function returns a list of the AOCE catalog-item categories known to the Catalog Browser.

```
pascal OSErr SDPGetCategories (
                              PackedRStringListHandle *categories,
                              PackedRStringListHandle *displayNames);
```

`categories` A pointer to a list of categories.

`displayNames`
A pointer to a list of user-readable category names.

DESCRIPTION

AOCE templates can group catalog records into categories. For example, the separate record types for LaserWriter, ImageWriter, and ImageWriter LC printers can be grouped into the category "printers." You can use the `SDPGetCategories` function to obtain a list of all the categories currently available.

SPECIAL CONSIDERATIONS

The size of the structures referenced by the handles returned by this function may be larger than a packed pathname.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Unlike most Standard Catalog Package routines, the `SDPGetCategories` function uses the $AA5C trap and does not require a parameter word count.

Trap	Routine selector
$AA5C	$0402

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter

SEE ALSO

Use the `SDPGetCategoryTypes` function (described next) to list all the types of items available within a specific category.

The `PackedRStringListHandle` structure is described in “RString List” on page 4-23.

SDPGetCategoryTypes

The `SDPGetCategoryTypes` function returns a list of the types of items within a catalog-item category known to the Catalog Browser.

```
pascal OSErr SDPGetCategoryTypes (const RString *category,
                                  PackedRStringListHandle *types);
```

`category` The catalog-item category for which you want a list of types.

`types` A pointer to a list of catalog-item types included in the category you specified.

DESCRIPTION

After you use the SDPGetCategories function to obtain a list of all the catalog-item categories currently available, you can call the SDPGetCategoryTypes function to list all the types of records included in a specific category. You can use this information, for example, to allow the user to choose which record types to include in the Catalog-Browsing panel or the Find panel.

Note

Before you pass the list of record types to the SDPNewPanel or SDPGetNewPanel function, you must add a record type of "DNode" (the value kDNodeRecTypeNum) to the types list to allow the panel to show aliases to dNodes. u

SPECIAL CONSIDERATIONS

The size of the structure referenced by the handle returned by this function may be larger than a packed pathname.

This function may move or purge memory; you should not call this function at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Unlike most Standard Catalog Package routines, the SDPGetCategoryTypes function uses the $AA5C trap and does not require a parameter word count.

Trap	Routine selector
$AA5C	$0403

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter

SEE ALSO

Use the SDPGetCategories function (page 4-91) to list all the record-type categories available.

The PackedRStringListHandle structure is described in "RString List" on page 4-23.

You can restrict the Catalog-Browsing panel and the Find panel to display records of specific types. For more information on this feature, see the descriptions of the SDPNewPanel (page 4-30), SDPGetNewPanel (page 4-34), and SDPNewFindPanel (page 4-61) functions.

Application-Defined Functions

This section describes the callback routines that you may provide for the Standard Catalog Package. Your `MyPanelBusyProc` and `MyFindPanelBusyProc` routines enable you to provide feedback to the user when the Catalog-Browsing panel and Find panel are busy.

MyPanelBusyProc

You can install a panel-busy callback routine to tell the user that the Catalog-Browsing panel is busy.

```
pascal void MyPanelBusyProc (SDPPanelHandle panel,
                             Boolean busy);
```

`panel` A handle to the panel for which you installed this routine.

`busy` A Boolean value indicating whether the panel is busy.

DESCRIPTION

You can use the `SDPInstallPanelBusyProc` function to install a panel-busy callback routine that the panel calls whenever the Catalog-Browsing panel is busy. You can use your panel-busy routine, for example, to display the Standard Catalog Package's spinning arrow icon while the panel is busy. The panel calls your routine one last time with the `busy` parameter set to `false` when the panel is no longer busy; you can then take down the spinning cursor.

You can use the following constants to obtain the icon IDs of the spinning arrows:

```
#define kFirstSpinnerIcon  -16745

#define kLastSpinnerIcon   -16738
```

SEE ALSO

Use the `SDPInstallPanelBusyProc` function (page 4-35) to install and remove panel-busy callback routines.

MyFindPanelBusyProc

You can install a Find-panel-busy callback routine that tells the user that the Find panel is busy.

```
pascal void MyFindPanelBusyProc (SDPFindPanelHandle findPanel,
                                 Boolean busy);
```

`findPanel`	A handle to the Find panel for which you installed this routine.
`busy`	A Boolean value indicating whether the Find panel is busy.

DESCRIPTION

You can use the `SDPInstallFindPanelBusyProc` function to install a Find-panel-busy callback routine that the Find panel calls whenever the panel is busy. You can use your callback routine, for example, to display the Standard Catalog Package's spinning arrow icon while the Find panel is busy. The Find panel calls your routine one last time with the `busy` parameter set to `false` when the Find panel is no longer busy; you can then take down the spinning cursor.

You can use the following constants to obtain the icon IDs of the spinning arrows:

```
#define kFirstSpinnerIcon  -16745

#define kLastSpinnerIcon   -16738
```

SEE ALSO

Use the `SDPInstallFindPanelBusyProc` function (page 4-65) to install and remove Find-panel-busy callback routines.

Summary of the Standard Catalog Package

C Summary

Constants and Data Types

```
#define gestaltSDPStandardDirectoryVersion   'sdvr'
#define gestaltSDPFindVersion                'dfnd'
#define gestaltSDPPromptVersion              'prpv'

/* selector mask values */
typedef unsigned long IconSelectorValue;
#define svLarge1Bit        0x00000001
#define svLarge4Bit        0x00000002
#define svLarge8Bit        0x00000004
#define svSmall1Bit        0x00000100
#define svSmall4Bit        0x00000200
#define svSmall8Bit        0x00000400
#define svMini1Bit         0x00010000
#define svMini4Bit         0x00020000
#define svMini8Bit         0x00040000
#define svAllLargeData     0x000000ff
#define svAllSmallData     0x0000ff00
#define svAllMiniData      0x00ff0000
#define svAll1BitData      (svLarge1Bit | svSmall1Bit | svMini1Bit)
#define svAll4BitData      (svLarge4Bit | svSmall4Bit | svMini4Bit)
#define svAll8BitData      (svLarge8Bit | svSmall8Bit | svMini8Bit)
#define svAllAvailableData    0xffffffff

/* generic icon suites */
/* icl8, icl4, ICN#, ics#, ics4, ics8, sicn */
#define genericDirectoryIconResource            -16721
/* icl8, icl4, ICN#, ics#, ics4, ics8, sicn */
#define genericLockedDirectoryIconResource      -16716
/* icl8, icl4, ICN#, ics#, ics4, ics8, sicn */
#define genericRecordIconResource               -16722
/* icl8, icl4, ICN#, ics#, ics4, ics8, sicn */
```

```
#define genericAttributeIconResource            -16723
/* icl8, icl4, ICN#, ics#, ics4, ics8 */
#define genericTemplateIconResource             -16746

/* standard icon suites */
/* icl8, icl4, ICN#, ics#, ics4, ics8, sicn */
#define directoryFolderIconResource             -16720
/* icl8, icl4, ICN#, ics#, ics4, ics8, sicn */
#define lockedDirectoryFolderIconResource       -16719
/* icl8, icl4, ICN#, ics#, ics4, ics8, sicn */
#define personalDirectoryIconResource           -16718
/* icl8, icl4, ICN#, ics#, ics4, ics8, sicn */
#define directoriesIconResource                 -16717
/* icl8, icl4, ICN#, ics#, ics4, ics8, sicn */
#define preferredPersonalDirectoryIconResource  -16724

/* icon IDs for spinning-arrow cursor */
#define kFirstSpinnerIcon                       -16745
#define kLastSpinnerIcon                        -16738

/* resource types */
#define kSDPPanelResourceType       'panl'
#define kSDPFindPanelResourceType   'find'

/* standard FindPanel resource */
#define kStandardFindLayout   -16700

enum {
   kSDPGuestBit,
   kSDPSpecificIdentityBit,
   kSDPLocalIdentityBit
};
/* values of SDPIdentityKind */
enum {
   kSDPGuestMask = 1<<kSDPGuestBit,
   kSDPSpecificIdentityMask = 1<<kSDPSpecificIdentityBit,
   kSDPLocalIdentityMask = 1<<kSDPLocalIdentityBit
};
typedef unsigned short SDPIdentityKind;

enum {
   kSDPSuggestionOnly,
   kSDPRestrictToDirectory,
```

```
   kSDPRestrictToRecord
};
typedef unsigned short SDPLoginFilterKind;

/* values of SDPSelectionState */
enum {
   kSDPNothingSelected,
   kSDPLockedContainerSelected,
   kSDPContainerSelected,
   kSDPRecordSelected,
   kSDPRecordAliasSelected,
   kSDPContainerAliasSelected
};
typedef unsigned short SDPSelectionState;

/* values of SDPPanelState */
enum {
   kSDPProcessed,
   kSDPSelectedAnItem,
   kSDPChangedSelection
};
typedef unsigned short SDPPanelState;

enum {
   kSDPItemIsSelectedBit,
   kSDPFindTextExistsBit
};
/* values of SDPFindPanelState */
enum {
   kSDPItemIsSelectedMask = 1<<kSDPItemIsSelectedBit,
   kSDPFindTextExistsMask = 1<<kSDPFindTextExistsBit
};
typedef unsigned short SDPFindPanelState;

/* values of SDPFindPanelFocus */
enum {
   kSDPFindPanelNoFocus,
   kSDPFindPanelListHasFocus,
   kSDPFindPanelTextHasFocus
};
typedef unsigned short SDPFindPanelFocus;
```

```
/* values of SDPFindPanelResult */
enum {
   kSDPSelectedAFindItem,
   kSDPFindSelectionChanged,
   kSDPFindCompleted,
   kSDPTextStateChanged,
   kSDPFocusChanged,
   kSDPSelectionAndFocusChanged,
   kSDPMenuChanged,
   kSDPSelectionAndMenuChanged,
   kSDPProcessedFind
};
typedef unsigned short SDPFindPanelResult;

/* Your application may read any of the fields in an SDPPanelRecord, but it
should use the appropriate routines to make changes to the records with the
exception of the refCon field, which your application may read or write at
will. Private information follows the SDPPanelRecord, so the handle must
not be resized. */

struct SDPPanelRecord {
   Rect              bounds;        /* rectangle around panel */
   Boolean           visible;       /* Is panel visible? */
   Boolean           enabled;       /* Is panel enabled? */
   Boolean           focused;       /* Is focus rectangle around panel? */
   Byte              padByte;       /* reserved */
   AuthIdentity      identity;      /* auth ID of caller of panel */
   long              refCon;        /* for your use */
   Rect              listRect;      /* rectangle around scrolling list */
   Rect              popupRect;     /* rectangle around pop-up menu */
   short             numberOfRows;  /* number of rows in scrolling list */
   short             rowHeight;     /* height of each row in list (points) */
   Boolean           pabMode;       /* Is panel in personal catalog mode? */
};

typedef struct SDPPanelRecord SDPPanelRecord;
typedef SDPPanelRecord *SDPPanelPtr, **SDPPanelHandle;

struct SDPFindPanelRecord {
   Point             upperLeft;                /* reserved */
   Boolean           visible;                  /* reserved */
   Boolean           enabled;                  /* reserved */
   Boolean           nowFinding;               /* reserved */
   Byte              padByte;                  /* reserved */
```

```
   SDPFindPanelFocus currentFocus;              /* reserved */
   AuthIdentity      identity;                  /* reserved */
   short             simultaneousSearchCount;   /* reserved */
   long              refCon;                    /* for your use */
};

typedef struct SDPFindPanelRecord SDPFindPanelRecord;
typedef SDPFindPanelRecord *SDPFindPanelPtr, **SDPFindPanelHandle;

/* pointers to functions for application-defined callback routines */

typedef pascal void (*PanelBusyProc) (SDPPanelHandle Panel,
                                      Boolean busy);

typedef pascal void (*FindPanelBusyProc) (SDPFindPanelHandle findPanel,
                                          Boolean busy);
```

Standard Catalog Package Functions

Authenticating a User

```
pascal OSErr SDPPromptForID (AuthIdentity *id,
                             ConstStr255Param guestPrompt,
                             ConstStr255Param specificIDPrompt,
                             ConstStr255Param localIDPrompt,
                             const RString *recordType,
                             SDPIdentityKind permittedKinds,
                             SDPIdentityKind *selectedKind,
                             const RecordID *loginFilter,
                             SDPLoginFilterKind filterKind);
```

Sorting a Personal Catalog

```
pascal OSErr SDPRepairPersonalDirectory
                            (FSSpec *pd, Boolean showProgress);
```

Creating, Displaying, and Disposing of a Catalog-Browsing Panel

```
pascal OSErr SDPNewPanel    (SDPPanelHandle *newPanel,
                             WindowPtr window,
                             const Rect *bounds,
                             Boolean visible,
                             Boolean enabled,
                             const PackedRLI *initialRLI,
                             const RStringPtr *typesList,
                             unsigned long typeCount,
```

```
                              AuthIdentity identity,
                              DirEnumChoices enumFlags,
                              DirMatchWith matchTypeHow,
                              long refCon);
pascal OSErr SDPGetNewPanel (SDPPanelHandle *newPanel,
                              short resourceID,
                              WindowPtr window,
                              const PackedRLI *initialRLI,
                              AuthIdentity identity);
pascal OSErr SDPInstallPanelBusyProc
                             (SDPPanelHandle panel,
                              PanelBusyProc busyProc);
pascal OSErr SDPSetPanelBalloonHelp
                             (SDPPanelHandle panel,
                              short balloonHelpID);
pascal OSErr SDPSetIdentity (SDPPanelHandle panel,
                              AuthIdentity identity);
pascal OSErr SDPSetPath      (SDPPanelHandle panel,
                              const PackedRLI *prli);
pascal OSErr SDPGetPathLength
                             (SDPPanelHandle panel,
                              unsigned short *pathNameLength);
pascal OSErr SDPGetPath      (SDPPanelHandle panel,
                              PackedRLI *prli,
                              short *dsRefNum);
pascal OSErr SDPSelectString
                             (SDPPanelHandle panel,
                              const RString *name);
pascal OSErr SDPHidePanel    (SDPPanelHandle panel);
pascal OSErr SDPShowPanel    (SDPPanelHandle panel);
pascal OSErr SDPEnablePanel (SDPPanelHandle panel,
                              Boolean enable);
pascal OSErr SDPSetFocus     (SDPPanelHandle panel,
                              Boolean focus);
pascal OSErr SDPUpdatePanel (SDPPanelHandle panel,
                              RgnHandle theRgn);
pascal OSErr SDPMovePanel    (SDPPanelHandle panel,
                              short h,
                              short v);
pascal OSErr SDPSizePanel    (SDPPanelHandle panel,
                              short width,
                              short height);
```

```
pascal OSErr SDPDisposePanel
                            (SDPPanelHandle panel);
```

Handling Events in a Catalog-Browsing Panel

```
pascal OSErr SDPPanelEvent  (SDPPanelHandle panel,
                             const EventRecord *theEvent,
                             SDPPanelState *whatHappened);
pascal OSErr SDPGetPanelSelectionState
                            (SDPPanelHandle panel,
                             SDPSelectionState *itsState);
pascal OSErr SDPGetPanelSelectionSize
                            (SDPPanelHandle panel,
                             unsigned short *dsSpecSize);
pascal OSErr SDPGetPanelSelection
                            (SDPPanelHandle panel,
                             PackedDSSpec *selection);
pascal OSErr SDPOpenSelectedItem
                            (SDPPanelHandle panel,
                             SDPPanelState *whatHappened);
```

Creating, Displaying, and Disposing of a Find Panel

```
pascal OSErr SDPNewFindPanel
                            (SDPFindPanelHandle *newPanel,
                             WindowPtr window,
                             Point upperLeft,
                             short layoutResourceID,
                             Boolean visible,
                             Boolean enabled,
                             const RStringPtr *typesList,
                             long typeCount,
                             DirMatchWith matchTypeHow,
                             AuthIdentity identity,
                             short simultaneousSearchCount,
                             SDPFindPanelFocus initialFocus,
                             long refCon);
pascal OSErr SDPInstallFindPanelBusyProc
                            (SDPFindPanelHandle findPanel,
                             FindPanelBusyProc busyProc);
pascal OSErr SDPSetFindPanelBalloonHelp
                            (SDPFindPanelHandle findPanel,
                             short balloonHelpID);
pascal OSErr SDPHideFindPanel
                            (SDPFindPanelHandle findPanel);
```

```
pascal OSErr SDPShowFindPanel
                            (SDPFindPanelHandle findPanel);
pascal OSErr SDPEnableFindPanel
                            (SDPFindPanelHandle findPanel,
                             Boolean enabled);
pascal OSErr SDPSetFindPanelFocus
                            (SDPFindPanelHandle findPanel,
                             SDPFindPanelFocus newFocus);
pascal OSErr SDPSetFindIdentity
                            (SDPFindPanelHandle findPanel,
                             AuthIdentity identity)
pascal OSErr SDPUpdateFindPanel
                            (SDPFindPanelHandle findPanel,
                             RgnHandle theRgn);
pascal OSErr SDPMoveFindPanel
                            (SDPFindPanelHandle findPanel,
                             short h,
                             short v);
pascal OSErr SDPDisposeFindPanel
                            (SDPFindPanelHandle findPanel);
```

Handling Events in a Find Panel

```
pascal OSErr SDPFindPanelEvent
                            (SDPFindPanelHandle findPanel,
                             const EventRecord *event,
                             SDPFindPanelResult *whatHappened);
pascal OSErr SDPGetFindPanelState
                            (SDPFindPanelHandle findPanel,
                             SDPFindPanelState *itsState);
pascal OSErr SDPGetFindPanelSelectionSize
                            (SDPFindPanelHandle findPanel,
                             unsigned short *size);
pascal OSErr SDPGetFindPanelSelection
                            (SDPFindPanelHandle findPanel,
                             PackedDSSpec *selection);
pascal OSErr SDPStartFind   (SDPFindPanelHandle findPanel);
pascal OSErr SDPStopFind    (SDPFindPanelHandle findPanel);
```

Resolving Aliases

```
pascal OSErr SDPResolveAliasFile
                            (FSSpecPtr fileSpec,
                             PackedDSSpecHandle resolvedDSSpec,
                             AuthIdentity identity,
                             Boolean mayPromptUser);
pascal OSErr SDPResolveAliasDSSpec
                            (PackedDSSpecHandle theAliasDSSpec,
                             AuthIdentity identity,
                             Boolean mayPromptUser);
```

Obtaining Icons and Lists of Catalog-Item Categories and Types

```
pascal OSErr SDPGetIconByType
                            (const RString *recordType,
                             IconSelectorValue whichIcons,
                             Handle *iconSuite);
pascal OSErr SDPGetDSSpecIcon
                            (const PackedDSSpec *packedDSSpec,
                             IconSelectorValue whichIcons,
                             Handle *iconSuite);
pascal OSErr SDPGetCategories
                            (PackedRStringListHandle *categories,
                             PackedRStringListHandle *displayNames);
pascal OSErr SDPGetCategoryTypes
                            (const RString *category,
                             PackedRStringListHandle *types);
```

Application-Defined Functions

```
pascal void MyPanelBusyProc (SDPPanelHandle panel,
                             Boolean busy);
pascal void MyFindPanelBusyProc
                            (SDPFindPanelHandle findPanel,
                             Boolean busy);
```

Pascal Summary

Constants

```
CONST
gestaltSDPStandardDirectoryVersion= 'sdvr';
gestaltSDPFindVersion= 'dfnd';
gestaltSDPPromptVersion= 'prpv';

{ selector mask values }
svLarge1Bit= $00000001;
svLarge4Bit= $00000002;
svLarge8Bit= $00000004;
svSmall1Bit= $00000100;
svSmall4Bit= $00000200;
svSmall8Bit= $00000400;
svMini1Bit= $00010000;
svMini4Bit= $00020000;
svMini8Bit= $00040000;
svAllLargeData= $000000ff;
svAllSmallData= $0000ff00;
svAllMiniData= $00ff0000;
svAll1BitData= (svLarge1Bit + svSmall1Bit + svMini1Bit);
svAll4BitData= (svLarge4Bit + svSmall4Bit + svMini4Bit);
svAll8BitData= (svLarge8Bit + svSmall8Bit + svMini8Bit);
svAllAvailableData= $ffffffff;

{ generic icon suites }
genericDirectoryIconResource= -16721;
{ icl8, icl4, ICN#, ics#, ics4, ics8, sicn }
genericLockedDirectoryIconResource= -16716;
{ icl8, icl4, ICN#, ics#, ics4, ics8, sicn }
genericRecordIconResource= -16722;
{ icl8, icl4, ICN#, ics#, ics4, ics8, sicn }
genericAttributeIconResource= -16723;
{ icl8, icl4, ICN#, ics#, ics4, ics8, sicn }
genericTemplateIconResource= -16746;
{ icl8, icl4, ICN#, ics#, ics4, ics8 }

{ standard icon suites }
directoryFolderIconResource= -16720;
{ icl8, icl4, ICN#, ics#, ics4, ics8, sicn }
lockedDirectoryFolderIconResource= -16719;
```

```
{ icl8, icl4, ICN#, ics#, ics4, ics8, sicn }
personalDirectoryIconResource= -16718;
{ icl8, icl4, ICN#, ics#, ics4, ics8, sicn }
directoriesIconResource= -16717;
{ icl8, icl4, ICN#, ics#, ics4, ics8, sicn }
preferredPersonalDirectoryIconResource= -16724;
{ icl8, icl4, ICN#, ics#, ics4, ics8, sicn }

{ icon IDs for spinning-arrow cursor }
kFirstSpinnerIcon = -16745;
kLastSpinnerIcon  = -16738;

{ resource types }
kSDPPanelResourceType= 'panl';
kSDPFindPanelResourceType= 'find';

{ standard FindPanel resource }
kStandardFindLayout= -16700;

kSDPGuestBit = 0;
kSDPSpecificIdentityBit = 1;
kSDPLocalIdentityBit = 2;

{ values of SDPIdentityKind }
kSDPGuestMask              = $0001; { 1<<kSDPGuestBit }
kSDPSpecificIdentityMask   = $0002; { 1<<kSDPSpecificIdentityBit }
kSDPLocalIdentityMask      = $0004; { 1<<kSDPLocalIdentityBit }

kSDPSuggestionOnly = 0;
kSDPRestrictToDirectory = 1;
kSDPRestrictToRecord = 2;

{ values of SDPSelectionState }
kSDPNothingSelected = 0;
kSDPLockedContainerSelected = 1;
kSDPContainerSelected = 2;
kSDPRecordSelected = 3;
kSDPRecordAliasSelected = 4;
kSDPContainerAliasSelected = 5;

{ values of SDPPanelState }
kSDPProcessed = 0;
kSDPSelectedAnItem = 1;
kSDPChangedSelection = 2;
```

```
kSDPItemIsSelectedBit = 0;
kSDPFindTextExistsBit = 1;

{ values of SDPFindPanelState }
kSDPItemIsSelectedMask = $0001; { 1<<kSDPItemIsSelectedBit }
kSDPFindTextExistsMask = $0002; { 1<<kSDPFindTextExistsBit }

{ values of SDPFindPanelFocus }
kSDPFindPanelNoFocus = 0;
kSDPFindPanelListHasFocus = 1;
kSDPFindPanelTextHasFocus = 2;

{ values of SDPFindPanelResult }
kSDPSelectedAFindItem = 0;
kSDPFindSelectionChanged = 1;
kSDPFindCompleted = 2;
kSDPTextStateChanged = 3;
kSDPFocusChanged = 4;
kSDPSelectionAndFocusChanged = 5;
kSDPMenuChanged = 6;
kSDPSelectionAndMenuChanged = 7;
kSDPProcessedFind = 8;
```

Data Types

```
TYPE
IconSelectorValue = LONGINT;

SDPMatchWith = INTEGER;

SDPIdentityKind = INTEGER;

SDPLoginFilterKind = INTEGER;

SDPSelectionState = INTEGER;

SDPPanelState = INTEGER;

SDPFindPanelState = INTEGER;

SDPFindPanelFocus = INTEGER;

SDPFindPanelResult = INTEGER;
```

```
{ Your application may read any of the fields in an SDPPanelRecord, but it
should use the appropriate routines to make changes to the records with the
exception of the refCon field, which your application may read or write at
will. Private information follows the SDPPanelRecord, so the handle must
not be resized. }

SDPPanelRecord = RECORD
   bounds: Rect;
   visible: BOOLEAN;
   enabled: BOOLEAN;
   focused: BOOLEAN;
   {padByte: Byte;}
   identity: AuthIdentity;
   refCon: LONGINT;
   listRect: Rect;
   popupRect: Rect;
   numberOfRows: INTEGER;
   rowHeight: INTEGER;
   pabMode: BOOLEAN;
   END;

SDPPanelPtr = ^SDPPanelRecord;
SDPPanelHandle = ^SDPPanelPtr;

SDPFindPanelRecord = RECORD
   upperLeft: Point;
   visible: BOOLEAN;
   enabled: BOOLEAN;
   nowFinding: BOOLEAN;
   {padByte: Byte;}
   currentFocus: SDPFindPanelFocus;
   identity: AuthIdentity;
   simultaneousSearchCount: INTEGER;
   refCon: LONGINT;
   END;

SDPFindPanelPtr = ^SDPFindPanelRecord;
SDPFindPanelHandle = ^SDPFindPanelPtr;

PackedRStringListHandle = ^PackedPathNamePtr;
```

```
PackedDSSpecHandle = ^PackedDSSpecPtr;

PanelBusyProc = ProcPtr;
   { PROCEDURE PanelBusyProc(Panel: SDPPanelHandle; busy: BOOLEAN); }

FindPanelBusyProc = ProcPtr;
   {PROCEDURE FindPanelBusyProc(findPanel: SDPFindPanelHandle;
                                                busy: BOOLEAN); }
```

Standard Catalog Package Functions

Authenticating a User

```
FUNCTION SDPPromptForID     (VAR id: AuthIdentity;
                             guestPrompt: StringPtr;
                             specificIDPrompt: StringPtr;
                             localIDPrompt: StringPtr;
                             recordType: RStringPtr;
                             permittedKinds: SDPIdentityKind;
                             VAR selectedKind: SDPIdentityKind;
                             loginFilter: RecordIDPtr;
                             filterKind: SDPLoginFilterKind): OSErr;
```

Sorting a Personal Catalog

```
FUNCTION SDPRepairPersonalDirectory
                            (pd: FSSpecPtr; showProgress: BOOLEAN): OSErr;
```

Creating, Displaying, and Disposing of a Catalog-Browsing Panel

```
FUNCTION SDPNewPanel        (VAR newPanel: SDPPanelHandle;
                             window: WindowPtr;
                             bounds: Rect;
                             visible: BOOLEAN;
                             enabled: BOOLEAN;
                             initialRLI: PackedRLIPtr;
                             typesList: RStringHandle;
                             typeCount: LONGINT;
                             identity: AuthIdentity;
                             enumFlags: DirEnumChoices;
                             matchTypeHow: SDPMatchWith;
                             refCon: LONGINT): OSErr;
```

```
FUNCTION SDPGetNewPanel      (VAR newPanel: SDPPanelHandle;
                              resourceID: INTEGER;
                              window: WindowPtr;
                              initialRLI: PackedRLIPtr;
                              identity: AuthIdentity): OSErr;
FUNCTION SDPInstallPanelBusyProc
                             (panel: SDPPanelHandle;
                              busyProc: PanelBusyProc): OSErr;
FUNCTION SDPSetPanelBalloonHelp
                             (panel: SDPPanelHandle;
                              balloonHelpID: INTEGER): OSErr;
FUNCTION SDPSetIdentity      (panel: SDPPanelHandle; identity:
                              AuthIdentity): OSErr;
FUNCTION SDPSetPath          (panel: SDPPanelHandle; prli: PackedRLIPtr):
                              OSErr;
FUNCTION SDPGetPathLength    (panel: SDPPanelHandle;
                              VAR pathNameLength: INTEGER): OSErr;
FUNCTION SDPGetPath          (panel: SDPPanelHandle; prli: PackedRLIPtr;
                              VAR dsRefNum: INTEGER): OSErr;
FUNCTION SDPSelectString     (panel: SDPPanelHandle; name: RStringPtr):
                              OSErr;
FUNCTION SDPHidePanel        (panel: SDPPanelHandle): OSErr;
FUNCTION SDPShowPanel        (panel: SDPPanelHandle): OSErr;
FUNCTION SDPEnablePanel      (panel: SDPPanelHandle; enable: BOOLEAN): OSErr;
FUNCTION SDPSetFocus         (panel: SDPPanelHandle; focus: BOOLEAN): OSErr;
FUNCTION SDPUpdatePanel      (panel: SDPPanelHandle; theRgn: RgnHandle):
                              OSErr;
FUNCTION SDPMovePanel        (panel: SDPPanelHandle; h: INTEGER; v:
                              INTEGER): OSErr;
FUNCTION SDPSizePanel        (panel: SDPPanelHandle; width: INTEGER;
                              height: INTEGER): OSErr;
FUNCTION SDPDisposePanel     (panel: SDPPanelHandle): OSErr;
```

Handling Events in a Catalog-Browsing Panel

```
FUNCTION SDPPanelEvent      (panel: SDPPanelHandle;
                             theEvent: EventRecord;
                             VAR whatHappened: SDPPanelState): OSErr;
FUNCTION SDPGetPanelSelectionState
                            (panel: SDPPanelHandle;
                             VAR itsState: SDPSelectionState): OSErr;
FUNCTION SDPGetPanelSelectionSize
                            (panel: SDPPanelHandle;
                             VAR dsSpecSize: INTEGER): OSErr;
FUNCTION SDPGetPanelSelection
                            (panel: SDPPanelHandle;
                             selection: PackedDSSpecPtr): OSErr;
FUNCTION SDPOpenSelectedItem
                            (panel: SDPPanelHandle;
                             VAR whatHappened: SDPPanelState): OSErr;
```

Creating, Displaying, and Disposing of a Find Panel

```
FUNCTION SDPNewFindPanel    (VAR newPanel: SDPFindPanelHandle;
                             window: WindowPtr;
                             upperLeft: Point;
                             layoutResourceID: INTEGER;
                             visible: BOOLEAN;
                             enabled: BOOLEAN;
                             typesList: RStringHandle;
                             typeCount: LONGINT;
                             matchTypeHow: SDPMatchWith;
                             identity: AuthIdentity;
                             simultaneousSearchCount: INTEGER;
                             initialFocus: SDPFindPanelFocus;
                             refCon: LONGINT): OSErr;
FUNCTION SDPInstallFindPanelBusyProc
                            (findPanel: SDPFindPanelHandle; busyProc:
                             FindPanelBusyProc): OSErr;
FUNCTION SDPSetFindPanelBalloonHelp
                            (findPanel: SDPFindPanelHandle;
                             balloonHelpID: INTEGER): OSErr;
FUNCTION SDPHideFindPanel   (findPanel: SDPFindPanelHandle): OSErr;
FUNCTION SDPShowFindPanel   (findPanel: SDPFindPanelHandle): OSErr;
```

```
FUNCTION SDPEnableFindPanel (findPanel: SDPFindPanelHandle;
                             enabled: BOOLEAN): OSErr;
FUNCTION SDPSetFindPanelFocus
                            (findPanel: SDPFindPanelHandle;
                             newFocus: SDPFindPanelFocus): OSErr;
FUNCTION SDPSetFindIdentity
                            (findPanel: SDPFindPanelHandle;
                             identity: AuthIdentity): OSErr;
FUNCTION SDPUpdateFindPanel
                            (findPanel: SDPFindPanelHandle;
                             theRgn: RgnHandle): OSErr;
FUNCTION SDPMoveFindPanel   (findPanel: SDPFindPanelHandle; h: INTEGER;
                             v: INTEGER): OSErr;
FUNCTION SDPDisposeFindPanel
                            (findPanel: SDPFindPanelHandle): OSErr;
```

Handling Events in a Find Panel

```
FUNCTION SDPFindPanelEvent  (findPanel: SDPFindPanelHandle;
                             event: EventRecord;
                             VAR whatHappened: SDPFindPanelResult): OSErr;
FUNCTION SDPGetFindPanelState
                            (findPanel: SDPFindPanelHandle;
                             VAR itsState: SDPFindPanelState): OSErr;
FUNCTION SDPGetFindPanelSelectionSize
                            (findPanel: SDPFindPanelHandle;
                             VAR size: INTEGER): OSErr;
FUNCTION SDPGetFindPanelSelection
                            (findPanel: SDPFindPanelHandle;
                             selection: PackedDSSpecPtr): OSErr;
FUNCTION SDPStartFind       (findPanel: SDPFindPanelHandle): OSErr;
FUNCTION SDPStopFind        (findPanel: SDPFindPanelHandle): OSErr;
```

Resolving Aliases

```
FUNCTION SDPResolveAliasFile
                            (fileSpec: FSSpecPtr;
                             resolvedDSSpec: PackedDSSpecHandle;
                             identity: AuthIdentity;
                             mayPromptUser: BOOLEAN): OSErr;
FUNCTION SDPResolveAliasDSSpec
                            (theAliasDSSpec: PackedDSSpecHandle;
                             identity: AuthIdentity;
                             mayPromptUser: BOOLEAN): OSErr;
```

Obtaining Icons and Lists of Catalog-Item Categories and Types

```
FUNCTION SDPGetIconByType   (recordType: RStringPtr;
                             whichIcons: IconSelectorValue;
                             VAR iconSuite: Handle): OSErr;
FUNCTION SDPGetDSSpecIcon   (packedDSSpec: PackedDSSpecPtr;
                             whichIcons: IconSelectorValue;
                             VAR iconSuite: Handle): OSErr;
FUNCTION SDPGetCategories   (VAR catagories: PackedRStringListHandle;
                             VAR displayNames: PackedRStringListHandle):
                             OSErr;
FUNCTION SDPGetCategoryTypes
                            (category: RStringPtr;
                             VAR types: PackedRStringListHandle): OSErr;
```

Application-Defined Functions

```
PROCEDURE MyPanelBusyProc   (panel: SDPPanelHandle; busy: BOOLEAN);
PROCEDURE MyFindPanelBusyProc
                            (findPanel: SDPFindPanelHandle; busy: BOOLEAN);
```

Assembly-Language Summary

Trap Macros

Trap Macros Requiring Routine Selectors

$AA5D

Selector	Count	Function
$0388	$0010	SDPPromptForID
$1A2C	$0003	SDPRepairPersonalDirectory
$0064	$0015	SDPNewPanel
$0065	$0009	SDPGetNewPanel
$0079	$0004	SDPInstallPanelBusyProc
$0078	$0003	SDPSetPanelBalloonHelp
$0073	$0004	SDPSetIdentity
$0070	$0004	SDPSetPath
$0075	$0004	SDPGetPathLength
$0076	$0006	SDPGetPath
$0074	$0004	SDPSelectString
$0067	$0002	SDPHidePanel
$0068	$0002	SDPShowPanel
$0069	$0003	SDPEnablePanel
$0077	$0003	SDPSetFocus
$006A	$0004	SDPUpdatePanel
$006B	$0004	SDPMovePanel
$006C	$0004	SDPSizePanel
$0066	$0002	SDPDisposePanel
$0071	$0006	SDPPanelEvent
$006E	$0004	SDPGetPanelSelectionState
$0072	$0004	SDPGetPanelSelectionSize
$006F	$0004	SDPGetPanelSelection
$006D	$0004	SDPOpenSelectedItem
$08FC	$0014	SDPNewFindPanel
$090C	$0004	SDPInstallFindPanelBusyProc
$090A	$0003	SDPSetFindPanelBalloonHelp
$0903	$0002	SDPHideFindPanel
$0902	$0002	SDPShowFindPanel
$0905	$0003	SDPEnableFindPanel

Selector	Count	Function
$0906	$0003	SDPSetFindPanelFocus
$090B	$0004	SDPSetFindIdentity
$0901	$0004	SDPUpdateFindPanel
$0904	$0004	SDPMoveFindPanel
$08FD	$0002	SDPDisposeFindPanel
$0900	$0006	SDPFindPanelEvent
$0907	$0004	SDPGetFindPanelState
$0908	$0004	SDPGetFindPanelSelectionSize
$0909	$0004	SDPGetFindPanelSelection
$08FE	$0002	SDPStartFind
$08FF	$0002	SDPStopFind
$0E74	$0007	SDPResolveAliasFile
$0E75	$0005	SDPResolveAliasDSSpec

$AA5C

Selector	Function
$0400	SDPGetIconByType
$0401	SDPGetDSSpecIcon
$0402	SDPGetCategories
$0403	SDPGetCategoryTypes

Result Codes

The allocated range of result codes for the Standard Catalog Package is –1950 through –1969. Routines may also return standard Macintosh result codes such as noErr 0 (no error) and fnfErr –43 (file not found).

kSDPNoSearchText	–1950	No text is in Find panel text field
kSDPTooManyLoginAttempts	–1951	User tried more than 3 incorrect passwords
kSDPNoSelection	–1952	No selection is in Catalog-Browsing panel
kSDPPersonalDirectoryRepairFailed	–1953	Cannot sort personal catalog

CHAPTER 5

AOCE Templates

Contents

This chapter describes how to expand the capabilities of the AOCE Catalogs Extension to the Finder. The AOCE Catalogs Extension (CE) allows users to use the Finder's iconographic interface to search through AOCE catalogs, examine records, and edit records. You can provide extensions to the CE that make it possible for the Finder to handle new types of records and attributes, to group record types in a new way, or to present the content of records and record attributes in a new way. These extensions to the AOCE Catalogs Extension are called **AOCE templates**.

This chapter describes the various types of AOCE templates, tells you how to write an AOCE template, and defines the resource types that you use to create an AOCE template. How you apply AOCE templates is up to you. Indeed, the entire point of AOCE templates is to allow developers to extend the CE in ways that could not be foreseen.

This chapter is intended for developers who are providing new record types or new attribute types and who want to allow users to use the AOCE Catalogs Extension to the Finder to view, create, or modify these records and attributes. The chapter is also for anyone who wants to extend the capabilities of the CE in any other way. This chapter is also required reading along with the chapter "Service Access Module Setup" in *Inside Macintosh: AOCE Service Access Modules* for anyone writing a service access module (SAM).

To use this chapter, you should be familiar with the structure of AOCE catalogs as described in the introduction to the chapter "Catalog Manager" in this book. You also must be familiar with Macintosh resources as described in the chapter on the Resource Manager in *Inside Macintosh: More Macintosh Toolbox*. The sample code listings in this chapter include code written for the Rez resource compiler.

This chapter first briefly describes the human interface of the AOCE Catalogs Extension and the way in which AOCE templates work together to modify the CE. Next it describes the use of each type of AOCE template in more detail. Finally, the chapter presents the details of the implementation of AOCE templates, including definitions of the various resource types you use to create AOCE templates.

Introduction to the AOCE Catalogs Extension

AOCE catalogs, described in the chapter "Catalog Manager" in this book, contain information arranged in a hierarchical structure similar to the Macintosh hierarchical file system (HFS). At the root level of the hierarchy is the AOCE catalog itself. Each catalog can contain any number of dNodes, and each dNode can contain dNodes and records, which contain the data.

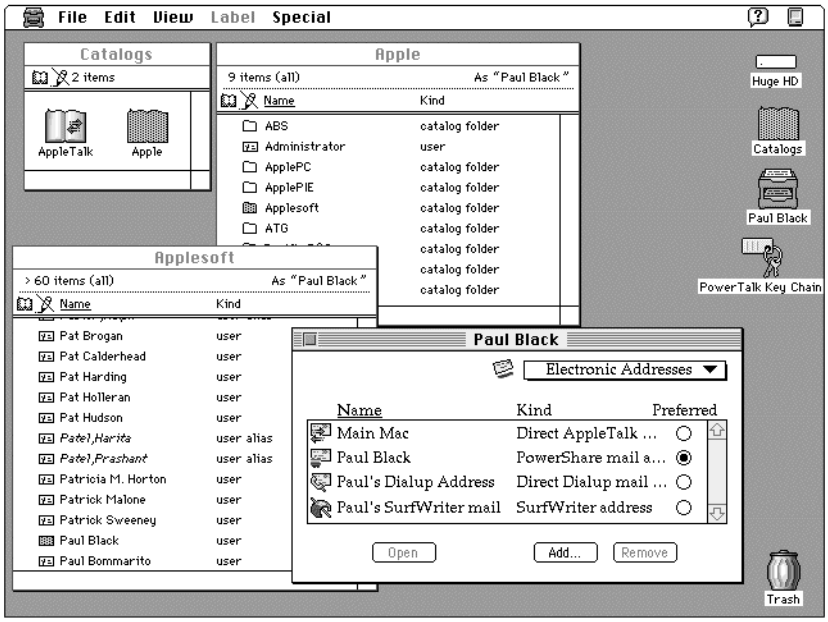
The Catalog Manager provides access to server-based AOCE catalogs, including those implemented by Apple Computer, Inc.'s PowerShare servers and those implemented by third parties through the CSAM interface. The Catalog Manager also provides access to catalogs on the local Macintosh computer: personal catalog files and information cards (individual records on disk).

Catalogs are organized in a hierarchy of dNodes, much like Macintosh HFS folders. Each dNode has a name, so that a particular location within the catalog can be specified by listing the dNode names leading from the catalog through the hierarchy to the particular dNode of interest. At the bottom of the hierarchy are records, just as Macintosh files are at the bottom of the HFS hierarchy. Each record has a name and a type. The data within records is organized into attributes, each of which can contain an arbitrary number of values. Each attribute has a type, and each attribute value has a *tag* that indicates its format. Attribute tags are of type `OSType` and are therefore 4 bytes long. Attribute values are blocks of data up to 64 KB in size.

PowerShare catalogs and the CE also support stand-alone attributes. A **stand-alone attribute** is 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. For example, if a user drags a single mail address from a User record and drops it in a personal catalog, the CE creates a stand-alone attribute containing that address. When the user drops that stand-alone attribute onto another User record, the AOCE software adds the address to the User record as an attribute.

The AOCE Catalogs Extension to the Finder places all PowerShare catalogs and CSAM catalogs within an icon (the “Catalogs” icon) that must remain on the desktop like the icon for a disk. To the user, each catalog within the Catalogs icon appears to be a folder, and each dNode within the catalog appears to be a folder inside another folder. Figure 5-1 shows what a user’s desktop might look like with catalogs and dNodes open.

Figure 5-1 The AOCE Catalogs Extension in use



From the user's perspective, browsing the catalog system is very much like browsing the Macintosh file system. Catalog dNodes look like file folders. They contain lists of enclosed dNodes and records. Opening a dNode produces a new window showing the contents of that dNode.

There are three major differences between browsing HFS directories and browsing AOCE catalogs:

- The AOCE Catalogs Extension can display the contents of catalogs that contain many more items than could be browsed in the file system. While file system browsing is limited to hundreds of items, catalog dNodes can contain tens of thousands of items.
- Finder windows that display lists of AOCE catalogs, contents of catalogs, and contents of dNodes include a column labeled "Kind," just as Finder windows for HFS folders include a "Kind" column. Whereas HFS windows differentiate such items as application documents, application programs, and folders, AOCE windows differentiate between PowerShare catalogs and CSAM catalogs, and among records of various types. Unlike the file system, AOCE catalog items are grouped into categories such as people, mail servers, and AppleShare. For example, the separate record types for LaserWriters, ImageWriters, and ImageWriter LCs could be grouped into the category "printers." The user can use the View menu to select the categories to be displayed (Figure 5-2).
- Whereas the Finder launches an application to display the contents of HFS files, the AOCE Catalogs Extension itself can display and be used to edit the contents of records.

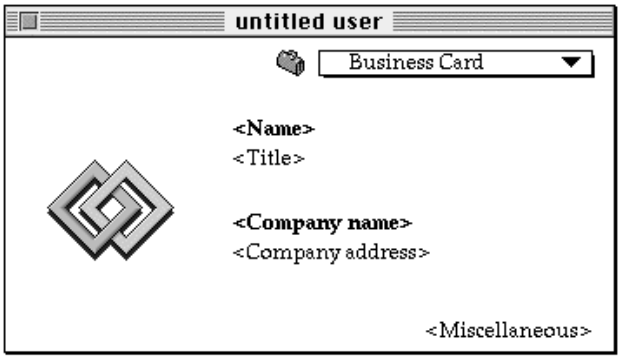
Figure 5-2 View menu seen with the AOCE Catalogs Extension to the Finder

When the user opens a record, the AOCE Catalogs Extension opens a window that lets the user select any of a series of information pages. Each **information page** (sometimes called an *info-page*) shows a portion of the contents of the record. The user moves to different information pages by using a pop-up menu.

Depending on the type of record and the design of the information pages, the user might be able to select various options such as different displays of the same data, perform functions such as dialing the telephone, and edit certain fields of the information page or

even create a new record. Figure 5-3 shows an information page for an address catalog. Notice the pop-up menu and editable text fields.

Figure 5-3 Information page



Any information page for a record can include a list of attribute values. The information page template specifies which attribute types are included in the list. Because the list includes a subset of the attribute types in the record and appears as a distinct portion of the information page window, these lists are referred to as **sublists**. Each sublist entry includes information from within the attribute value as specified by the template. After opening an item in a sublist, the user is presented with a new information page window displaying the contents of the opened attribute. Similarly, a dNode window can include a sublist of records contained in the dNode. Figure 5-4 shows an information page with a sublist, and Figure 5-5 shows the information page that appears when the user opens one of the items in that sublist. (The information page in Figure 5-5 also appears when the user opens a stand-alone attribute created by dragging the item from the sublist and dropping it in a catalog or on the desktop.)

Figure 5-4 Information page with a sublist

Figure 5-5 Information page for an item in a sublist

You can extend the ability of the AOCE Catalogs Extension to display the contents of records and attributes by providing AOCE templates.

Introduction to AOCE Templates

The AOCE Catalogs Extension to the Finder provided with the PowerTalk system software can display a certain number of record types and attribute types. If you want to provide users with the ability to work with other record types or attribute types, you can extend the AOCE Catalogs Extension (CE) by writing AOCE templates.

Templates allow developers to extend the browsing capabilities of the system in several ways: adding new types of visible records, defining the kind and category for a particular record type or attribute type shown in a list, and extending the available information pages for displaying the record or attribute contents to the user.

The data in a record is stored in data structures known as **attributes**. Each attribute can contain any number of **attribute values**. Each attribute has a *type*, and each attribute value has a *tag* that indicates its format. Attribute values can be up to 64 KB in size. The `attribute` structure is defined in the chapter “AOCE Utilities” in this book.

There is not necessarily a one-to-one correspondence between attribute values and data items of interest to a user. For example, the name and address of a person could be stored as a single attribute value, as two attribute values (one containing the name and one the address), or as several attribute values (one containing the first name, one the last name, one the house number and street, one the state, and so forth). There are no restrictions on the type of data that can be placed in attributes, and, except for a few standard attribute types, there is no way for the CE to determine how to display or interpret an attribute. For this reason, for each new record type or attribute type that you

add to a catalog, you must provide templates that tell the CE how to display the data contained in records or attributes of that type.

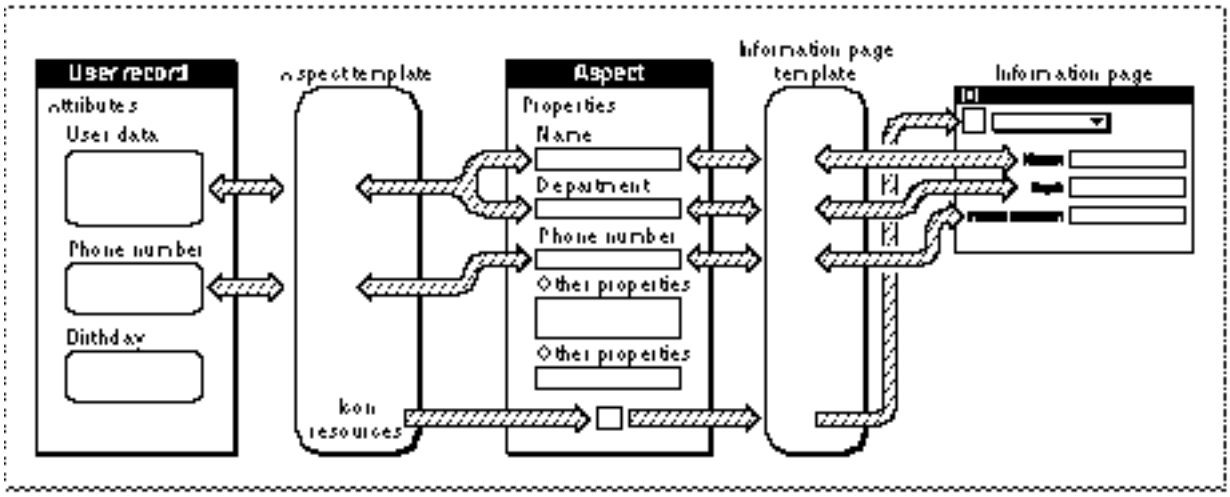
To display the data contained in records, AOCE templates must do two things: parse attribute values into the individual data items of interest to the user (referred to as **properties**) and specify how the Finder should display each property. For example, an attribute value could contain three strings: a house number, a street name, and a city name. To display each string as a separate item on the screen, you would provide two AOCE templates: an aspect template and an information page template. The aspect template would specify that the house number, street name, and city name each constitutes a property, and the information page template would specify that each of these items is to be displayed in an editable text box and would specify the size and location of each text box in the information page.

Just as an information page template has an associated information page, an aspect template has an associated **aspect**; a structure in memory that contains properties provided by the aspect template. Each information page displays properties taken from a single aspect. Note, however, that a given information page template need not use *all* of the properties in an aspect, and any number of information page templates can use properties from a single aspect.

Note
In the terminology of object-oriented programming, the information page is the view/controller and the attribute value is the persistent storage; the aspect template is the class for the aspect instance, and the information page template is the class for the information page instance. u

Figure 5-6 illustrates the relationships among records, aspect templates, aspects, information page templates, and information pages. As the figure shows, the aspect template processes the data in an attribute within a catalog record to create properties in an aspect. The aspect template itself might also provide data for properties, as is the case with the icon resources in the figure. The information page template specifies how the properties are displayed on an information page. The information page template can also provide such unchanging items as labels for text fields. Any editable item in the information page can also be processed in the other direction by the templates to change the data in the catalog record, as indicated by the bidirectional arrows in the figure.

Figure 5-6 From a record to an information page



There are five different types of AOCE templates:

- An **aspect template** specifies an aspect that provides information about a record or attribute of a particular type. Some of the information in an aspect is specified by the aspect template itself and therefore applies to all records or attributes of the same type. Examples of such information are the kind and category of an attribute or the icon of a record. Other information in the aspect is extracted by the template from the record, such as a string or number from the contents of an attribute value. Each aspect includes a collection of items of various types, which are known as properties. The aspect template includes instructions that tell the CE how to create properties from attributes and attributes from properties. Some aspect templates also specify how new records of a specific type are to be added to the containing dNode or how new attributes of a specific type are to be added to the containing record.

  Each aspect is independent of all other aspects, even aspects created from the same attribute or record. Therefore, two developers can provide separate aspect templates that act on the same attribute or record without causing any conflicts.

- An **information page template** specifies how the Finder should display record or attribute data. The information page template specifies which aspect to use to fill in the fields of the information page and the graphic layout of the information page.

- A **forwarder template** allows existing aspects and information pages to be used for new types of records. Using forwarders, a single information page template can be applied to several different record or attribute types.

- **Killer templates** allow developers to supersede existing templates. Killer templates identify one or more templates by name and cause the CE to ignore those templates. For example, if you want to override one of the built-in templates, you can provide a killer template that disables the existing template and a replacement template that the CE uses instead.
- In some applications of templates, all of the types of files that might contain templates cannot be known ahead of time. **File type templates** allow you to extend the list of file types that the CE searches for templates. All of the new files containing templates must, however, reside in the System Extensions folder.

The aspect and information page templates specify how to divide a record or attribute into properties and how to display and edit those properties through information pages. Forwarder, killer, and file type templates, in contrast, are concerned with which templates to use and where to find them.

Templates reside in the resource forks of files. The CE looks in several places for templates:

- template files (as indicated by a file type of `'detf'`) in the System Extensions folder
- MSAM and CSAM files in the System Extensions folder
- additional files of the types specified by any file type templates
- the PowerTalk Extension file, also in the System Extensions folder, which includes all the templates that are included as part of the PowerTalk software

Templates consist of a set of associated resources in the resource forks of these files. Each template has a **signature resource** that indicates the type of template. The ID of the signature resource is used to locate the other resources that make up the template. All additional resources are at fixed offsets from this base ID.

All templates, regardless of type, include a signature resource and a name resource. Each of the different types of templates also contain additional resources specific to that template's function. For example, an information page template includes the record and attribute types that it applies to, plus the layout of the information page to be displayed; a killer template includes a list of template names to be ignored.

Aspect Templates

An aspect template creates an aspect, which is a collection of information about a particular part of a record or an attribute. The information is divided into properties. Each property is identified by a unique number and can be any one of the following types:

Property type	Description
String	A string of text characters, stored internally as an `RString` structure
Number	A numerical value, stored internally as a long integer
Icon	An icon suite; always stored within the aspect template as a set of resources
Binary	An uninterpreted block of bytes, which can be used to store arbitrarily formatted data

Aspects serve two primary purposes:

- They provide unique identification for properties. Although within an aspect each property is identified only by a number and the same number is likely to be used for semantically different properties in different aspect templates, the combination of aspect template name and property number should be unique. To ensure that your aspect template has a unique name, you should start the template name with a 4-character application signature registered with Macintosh Developer Technical Support.
- They allow efficient access to subsets of the data in a record or attribute. For an item to appear in a sublist—such as a record in a dNode window or an attribute in a record information page—the Catalogs Extension must have icon, kind, and category information for that item. The CE takes this information from a single aspect. If you place other information about the item in other aspect templates, the CE does not create the other aspects for the item until they are needed, such as when an information page is opened for the item. Not creating aspects until they are needed saves time and memory. In addition, one aspect can often be used by more than one information page.

Property values in aspects are derived from three sources:

- the aspect template itself
- the record or attribute value
- the CE, which fetches related information not directly a part of an attribute (such as the access mask)

Some properties provide unchanging information, such as a record's kind or default values for changeable information. The CE takes these properties from resources in the aspect template itself by using the property number as an offset from the base ID of the template. For example, a resource of type `'rstr'` (`RString`) with an ID of 1013 in a template with a base ID of 1000 corresponds to a string property with property number 13.

For properties that must be taken from a record or attribute, the aspect template includes directions for dividing up attribute values to extract the properties. The aspect template

can include a lookup table with instructions for parsing attribute values into properties (“The Lookup-Table Resource” beginning on page 5-105), a code resource (“Code Resources Reference” beginning on page 5-142), or both. The CE uses the same process in reverse to revise or create attribute values in the record when the user edits a field in an information page. The lookup-table mechanism for parsing attribute values should allow you to create most of your aspect templates without writing any code. Aspect template lookup tables (also referred to as *patterns*) provide a wide range of different types of data structures that can be combined to handle almost any attribute format.

An aspect template can convert data from one type to another as it divides an attribute value into properties and also when it takes the value of a property from a field on an information page. For instance, an aspect template might convert a number in an attribute to a string property to allow the user to edit it. The property-type system is extensible, allowing the aspect template code resource to supply additional types and perform the appropriate type conversions.

Each aspect template includes a specification of the types of records and attributes to which the template applies. An aspect template can apply to a particular type of record or to a particular type of attribute found either in any record type or only in specific record types.

Information Page Templates

There is an information page template for each information page displayed to users. The template specifies the physical layout of the information page and indicates what properties are used to fill in each field (or **view**) in the information page.

For a list of possible view types, see “View Lists” on page 5-123.

Like aspect templates, information page templates apply only to a specified record or attribute type.

Forwarder Templates

Forwarder templates allow a new record or attribute type to use existing aspect and information page templates. A forwarder template includes a specification of the record type and attribute type to which it applies, just as is the case with aspect and information page templates. In addition, the forwarder template contains a list of aspect and information page template names to be used with the specified record or attribute type.

Killer Templates

Killer templates disable existing templates. A killer template consists of a list of names of the templates to be disabled.

Killer templates do not change the affected template. They just render it inactive at the time it would have been used.

You can use killer templates to disable any type of template except another killer template.

File Type Templates

The CE looks for templates during system initialization and the first time the CE needs a template after someone has called the `kDETcmdUnloadTemplates` callback routine (page 5-208). The CE always looks for templates in files of type 'detf', `'dsam'`, `'msam'`, and `'csam'`; and in files of type `'fext'` that have creator type `'adbk'`. (See “File and Resource Types Used by the Catalogs Extension” on page 5-73 for more information about these file types.) File type templates specify additional file types in which the CE looks for templates. The new files can also include file type templates. All of the new files containing templates must reside in the System Extensions folder.

How Aspect and Information Page Templates Work

The most important function of AOCE templates is to provide users with new information pages through which they can view and modify information contained within the AOCE catalog system. To accomplish this, the Catalogs Extension starts with a record or attribute and, following the instructions in aspect templates, creates aspects of the record or attribute. Each aspect can contain information derived from a single attribute, or—in the case of aspects derived from records—from several attributes. Because each aspect is independent from other aspects of the same record or attribute, you can create them without concern for what other developers might do. An aspect contains one or more values, each of a specific type—a text string or a number, for example. These values are referred to as *properties*. The CE then follows the instructions in an information page template to use the properties to fill in one or more fields (referred to as *views*) in the information page that is displayed to the user. Each information page is associated with only one aspect, from whose properties values for all of its views are taken.

When the user changes a value in an information page and then closes the page, the CE uses the same process in reverse to revise the attribute value in the record. If the record does not already contain this attribute value, the CE creates a new one from the values in the information page, following the instructions in the aspect template.

Figure 5-7 shows the process of translating between a record and an aspect. The aspect template uses lookup tables and code resources to parse the attribute data into properties. The aspect template can also contain property resources that it uses to create uneditable properties; in Figure 5-7 the icon property is created in this way.

Figure 5-7 Creating an aspect from a record

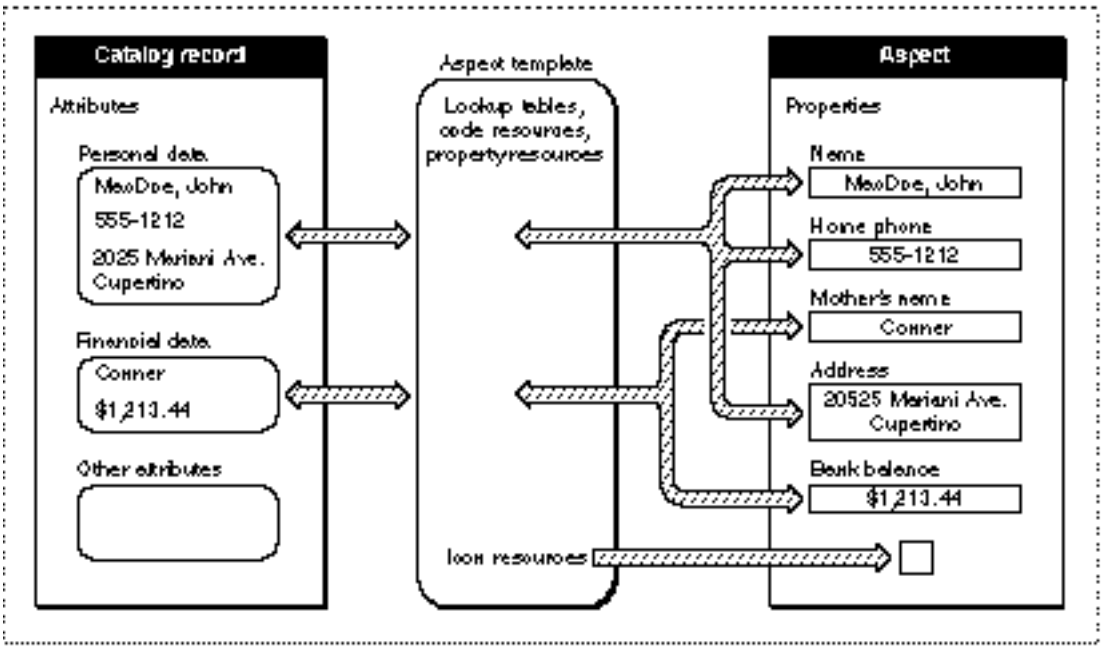
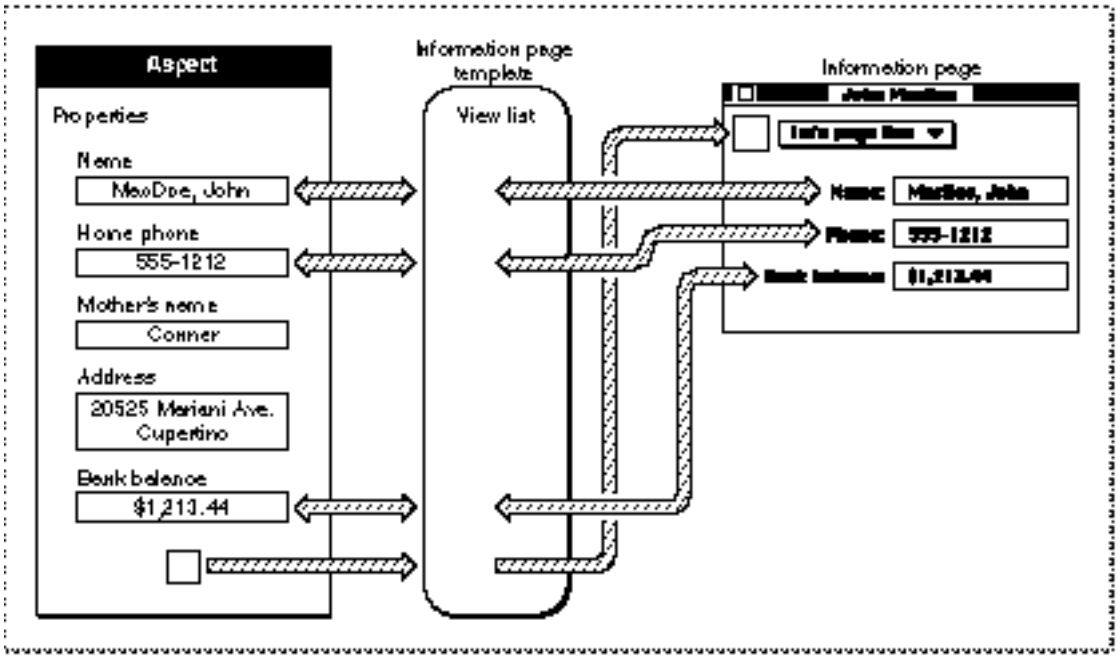


Figure 5-8 shows how the information page template translates properties in an aspect into views (fields) in an information page. The information page template includes one or more view lists that describe the layout and type of each field in the information page and that assign a property to each view. Note that not all the properties in the aspect must be represented in a single information page.

Figure 5-8 Creating an information page from an aspect

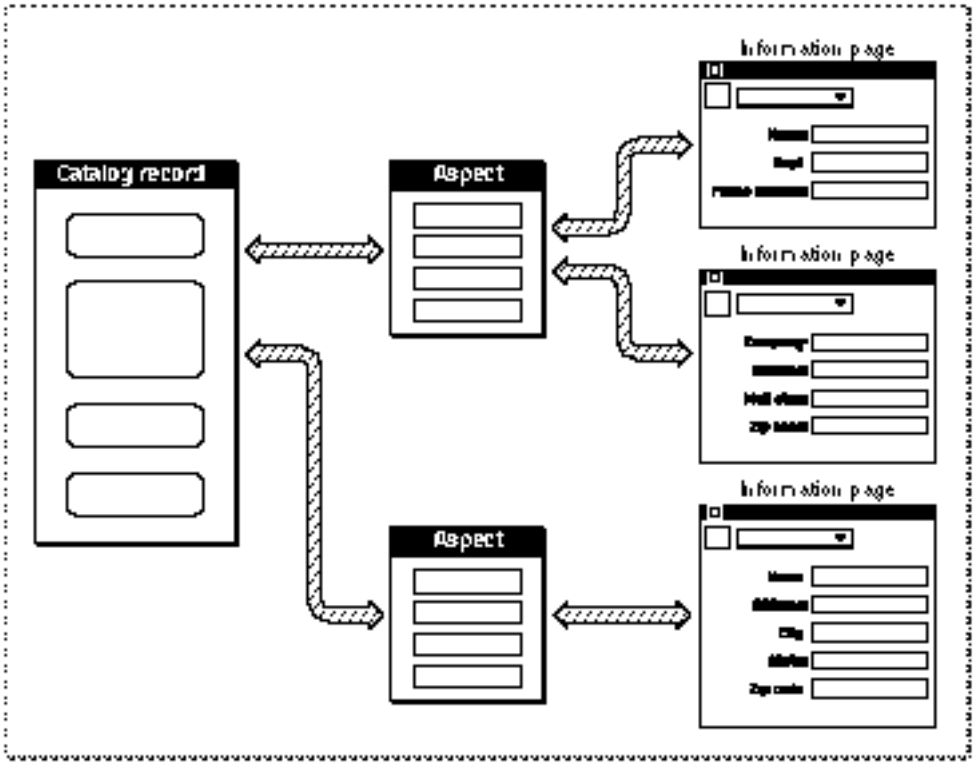


Any number of aspect templates can extract data from the same record or attribute, and the aspects they create can exist simultaneously without conflict. Because the properties in a given aspect are identified only by number, the CE uses the combination of the aspect's name and the property number to identify a property uniquely. Therefore, it is important that the name of an aspect be unique. When you create an aspect template, you should give it a name that includes a description of its purpose plus your application signature or other unique identifying string. The CE uses aspect names solely for internal identification; the user never sees them.

An information page takes the properties for all of its views from a single aspect. Each view specification in the information page template tells which property number is to be used. The information page template itself includes the name of the aspect within which these properties are found.

Figure 5-9 shows a record with multiple aspects and information pages. Note that one aspect can be used by more than one information page.

Figure 5-9 Multiple aspects and information pages



When a record or attribute appears in a sublist—in a dNode window in the case of records, or in an information page in the case of attributes—the CE takes the properties needed to fill in the data for an item in the list from a special aspect known as the **main aspect**. Whereas any aspect can contain information about the *contents* of a record or attribute, only a main aspect contains information about the record or attribute itself: its name, kind, category, and icon. In the case of records, this information usually consists of unchangeable properties stored permanently in the main aspect template. In the case of attributes, however, the main aspect template often retrieves the information from the attribute. Thus, the information changes when someone edits the attribute value.

There is a separate main aspect for each item in a sublist (note, however, that more than one of these main aspects may be derived from the same aspect template). Figure 5-10 shows how main aspects for records are used to fill in the contents of a sublist in a dNode window.

Figure 5-10 Main aspects for records

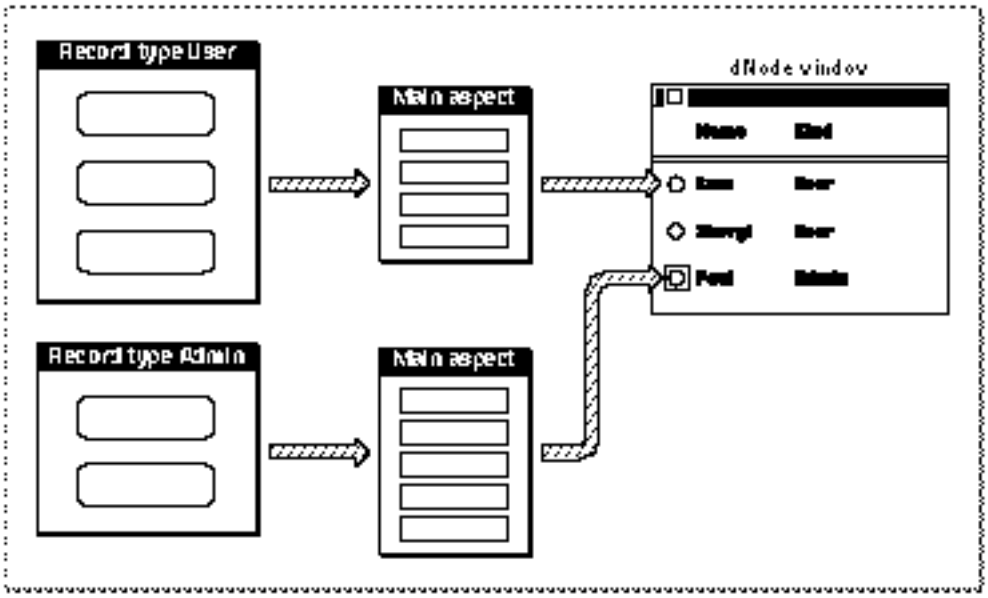
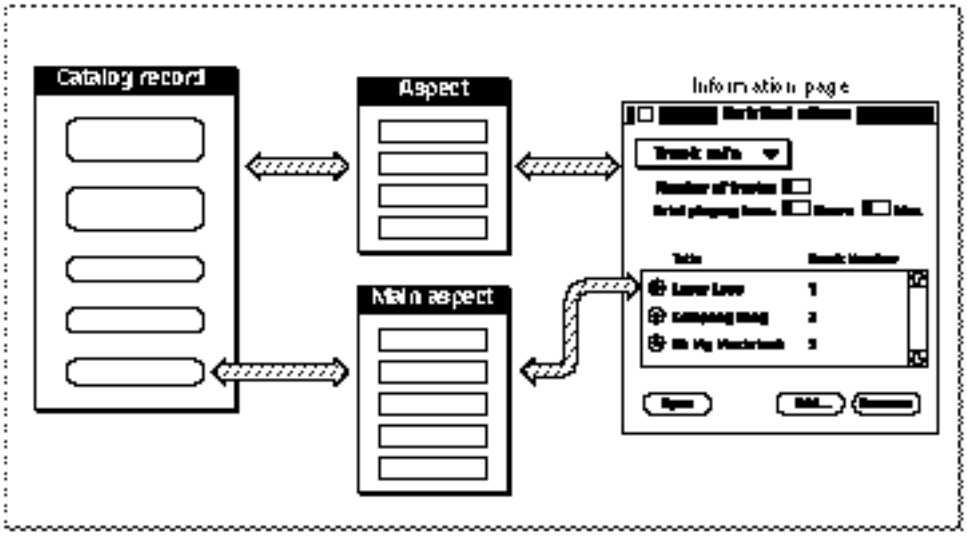


Figure 5-11 shows how aspects and main aspects are used to fill in the contents of an information page that contains a sublist. All of the properties for the views in the main part of the information page come from a single aspect, the **main view aspect**. This aspect also specifies whether there is a sublist in the information page and which attribute types are to be included in the sublist. Each attribute in the sublist has its own main aspect, which provides the information shown in the sublist for that attribute.

Figure 5-11 shows two of the aspects used to fill in an information page: the main view aspect and a main aspect for an item in the sublist. There is one aspect template for each attribute type in the sublist, and a separate aspect template for the main view aspect. Note that, whereas the properties in the aspect for the main part of the information page can come from any number of attributes in the record, a main aspect (which describes a single line in a sublist) derives its properties from a single attribute value.

Figure 5-11 Main aspects for attributes



There must be a separate main aspect template for each type of record displayed in a dNode window and for each type of attribute displayed in a sublist. The CE can use a main aspect template to create main aspects for any number of items of the same type. Figure 5-12 shows three records, two of type User and one of type Admin. The two records of type User are processed by a single main aspect template to create a main aspect for each record; the record of type Admin is processed by a separate main aspect template to create its main aspect.

Figure 5-12 Main aspect templates for records

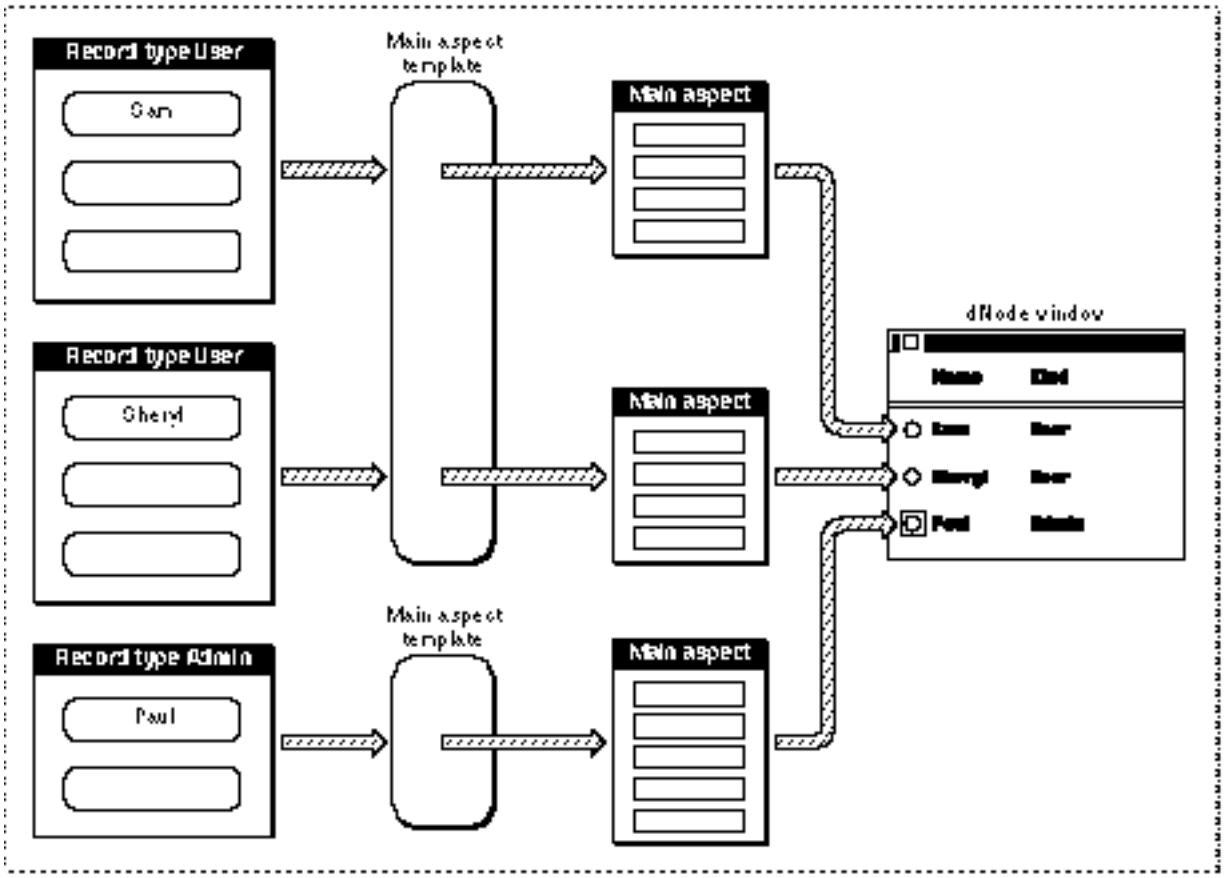
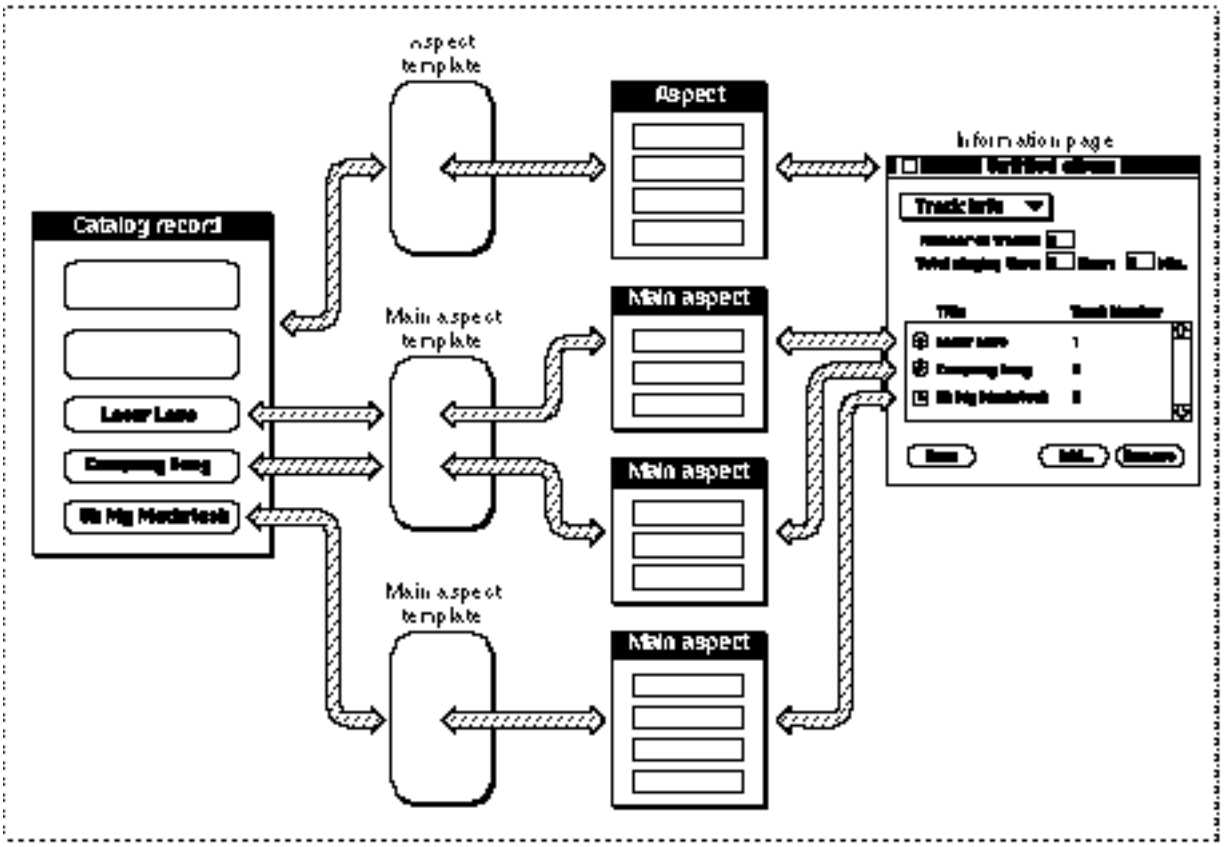


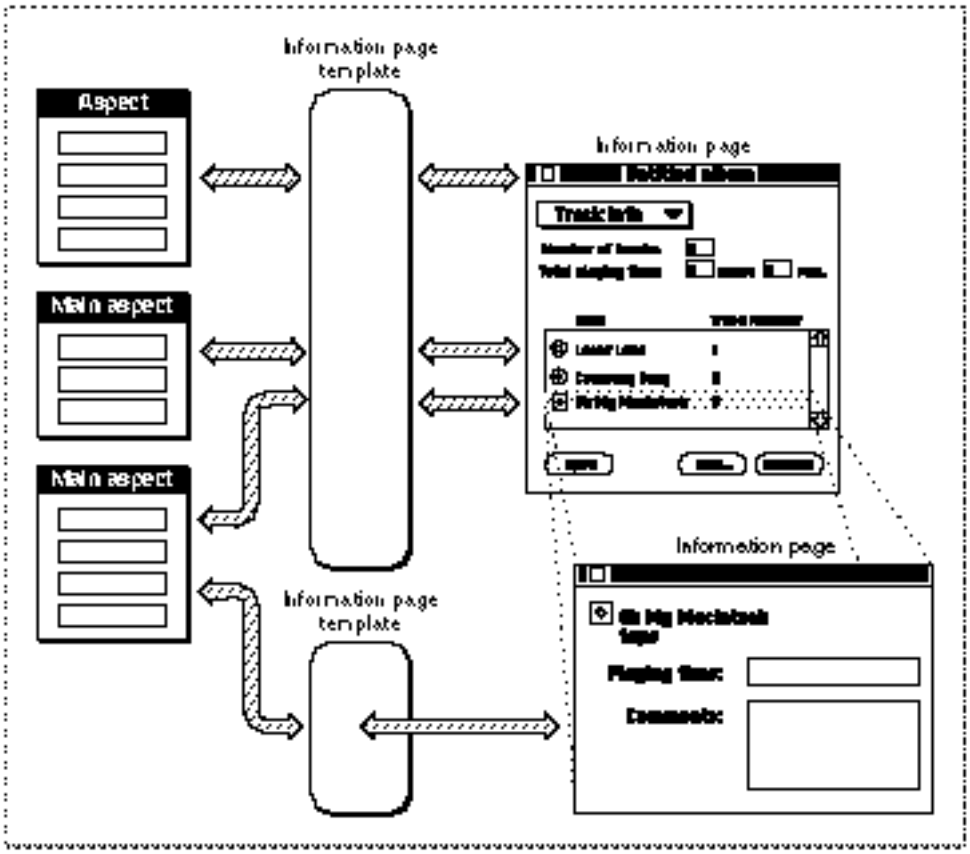
Figure 5-13 shows how an information page with a sublist is created from one aspect template and one or more main aspect templates. The aspect template creates an aspect for the main part of the information page. Each attribute type has a separate main aspect template; several attributes of the same type might be processed by the same main aspect template. Each attribute in the sublist has its own main aspect.

Figure 5-13 Main aspect templates for attributes



A main aspect can contain properties used by other information pages as well as the information needed for a sublist. As shown in Figure 5-14, a typical use for this feature is for a main aspect to contain all the properties for the information page that appears when the user double-clicks an attribute in a sublist. Note that all of the views for a single information page are described in a single information page template. Even the position of the sublist and the layout of each line in the sublist are described in this information page template. As shown in the figure, the information page that appears when the user opens an attribute in the sublist requires its own information page template.

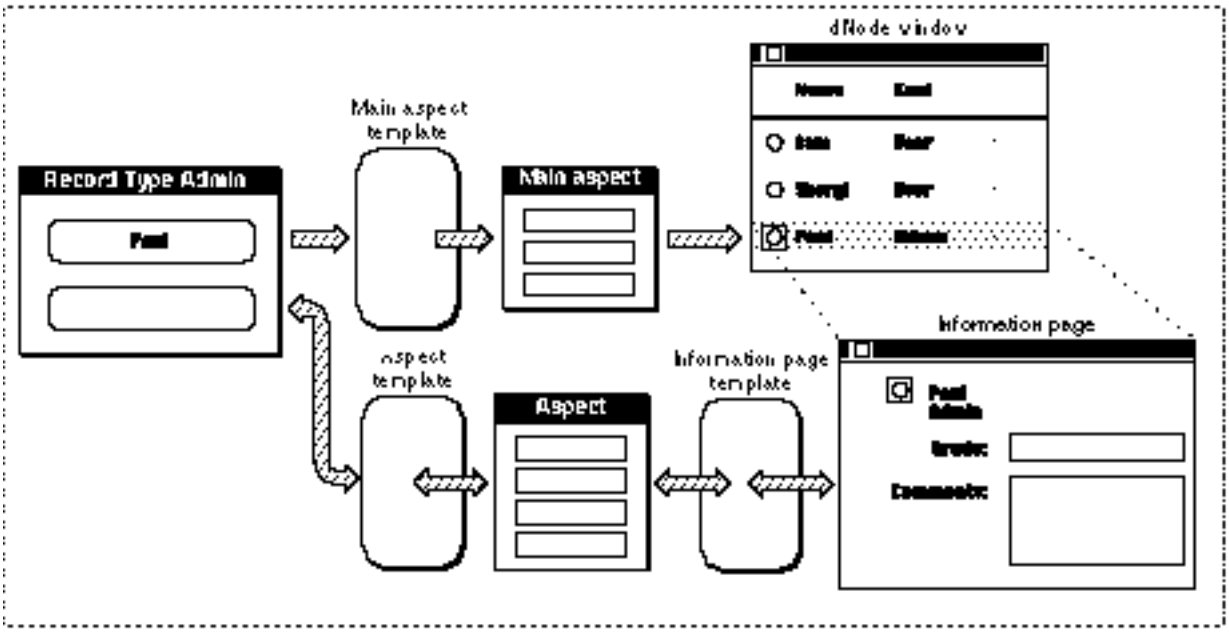
Figure 5-14 Providing an information page for an attribute in a sublist



In contrast to the situation shown in Figure 5-13 and Figure 5-14, a record shown in a dNode window list typically has at least two aspect templates associated with it: a main aspect template used to display information about the record in the dNode window plus one or more aspect templates used to provide properties for the information pages that are displayed when the user opens that record. Figure 5-15 illustrates this situation. Note that you must provide both an aspect template and an information page template to display the contents of the record, but you do not provide an information page template for the dNode window.

Keeping the main aspect template and other aspect templates for a record separate allows the Catalogs Extension to load into memory only the aspects that are needed at a given time and makes it easier for developers and users to create new information pages for an existing record type.

Figure 5-15 Providing an information page for a record in a dNode window list



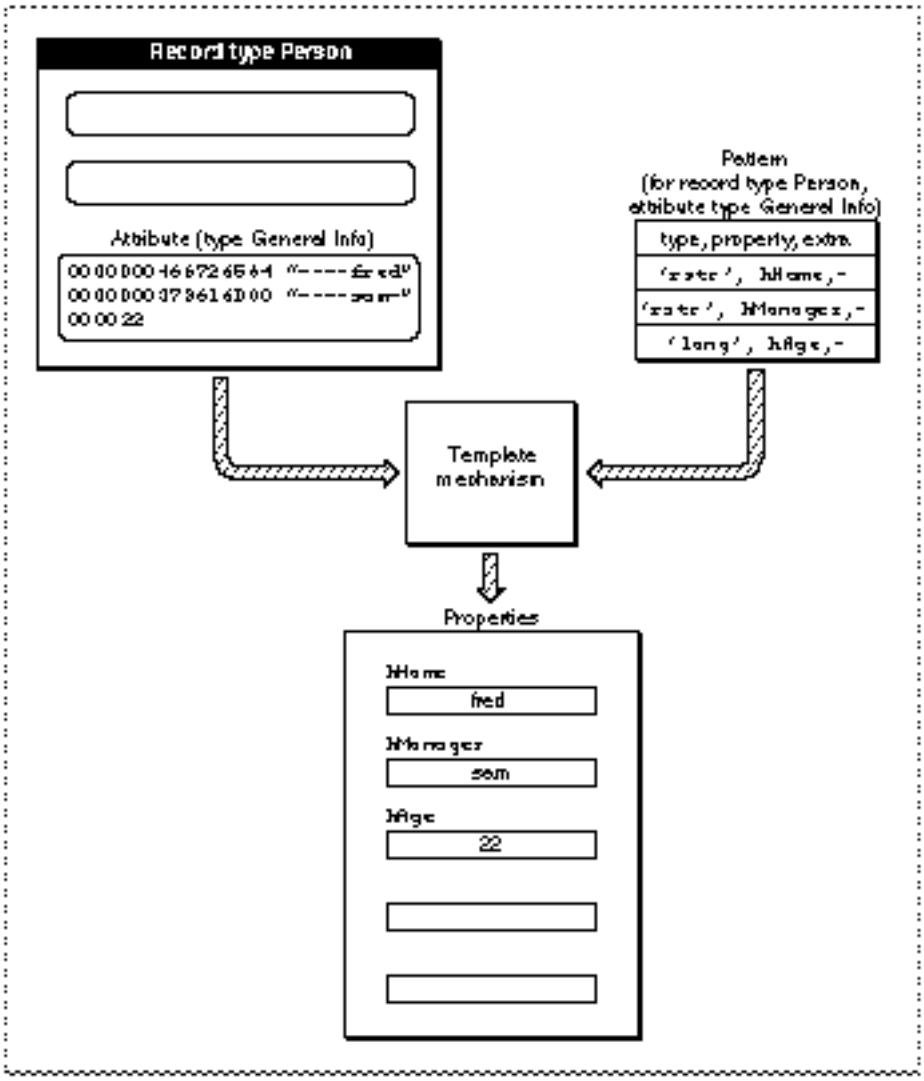
A main aspect template for a record type specifies how new records of that type are to be added to the containing dNode. Similarly, a main aspect template for an attribute type specifies how new attributes of that type are to be added to the containing record.

Main aspect templates are described in "Aspect Template Signature Resource," which starts on page 5-88.

The process of filling in views in information pages from properties in aspects is fairly straightforward. The information page template includes a property number for each view that requires data from the aspect. The information page template includes instructions for how to interpret the contents of the property—as a number, a text string, and so forth—how to display it, and whether to allow the user to edit it. Information page templates may display the same type of property in different ways depending on the circumstances. For example, a number property is used in both checkboxes and pop-up menus. In checkboxes, this property indicates whether the checkbox is selected or not. In pop-up menus, it indicates which of the entries in the menu is currently selected.

The process of filling in properties from records and records from properties is more complex. The aspect template provides two mechanisms: lookup tables and code resources. Lookup tables can translate a large variety of data structures without requiring you to write any code. Code resources cover all data formats, including those not handled by lookup tables. In addition, code resources can perform actions based on the new data being written to the catalog system. For example, adding a new user record might trigger an update of a personal gateway's internal user list.

Figure 5-16 Pattern-based attribute parsing



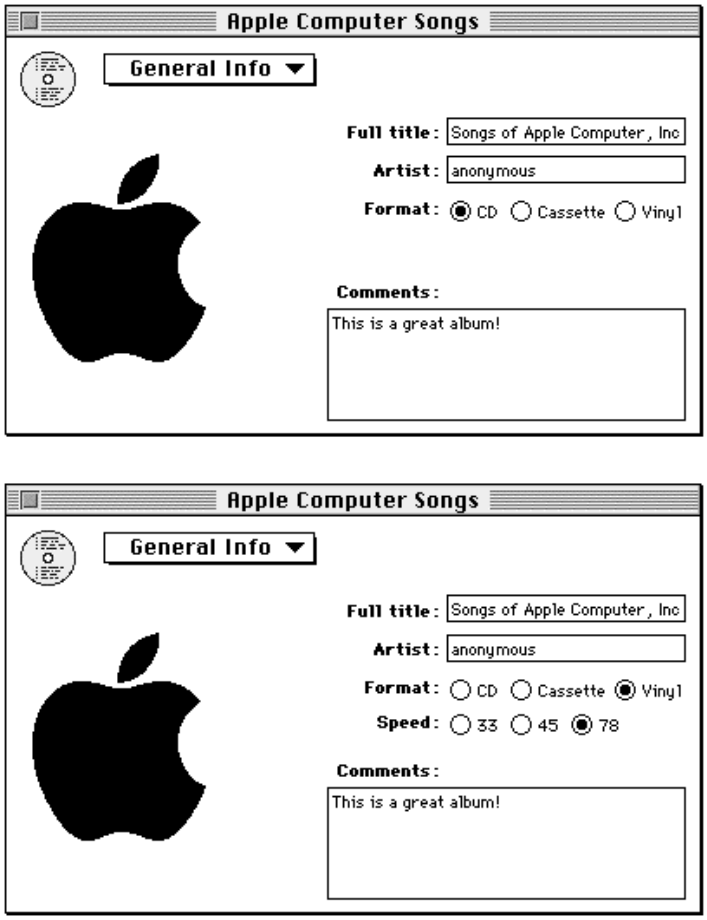
Lookup Tables

A **lookup table** contains a pattern that describes the contents of an attribute and specifies into which property to store each part of the attribute value. In many cases, the pattern is extremely simple. For instance, an attribute value might consist only of a single string of type `RString` or only of a single binary number. Other attribute value formats may be more complex, combining multiple items in a single attribute value, or requiring conditional evaluation of the contents. Figure 5-16 illustrates the basic process of creating properties from attribute values.

A lookup table also works in reverse, revising the contents of attribute values when the user enters new data in views in the information page. If an attribute value does not

already exist, the lookup table creates a new attribute value from the data and puts it into the record.

Figure 5-17 Conditional view



Conditional Views

The information page template includes one or more view lists; each view list describes one or more views that can be displayed on the information page. Each view list is associated with two property values. The views described by that view list are displayed only if those property values are equal (or if either property equals `kDETNoProperty`). Therefore, you can control whether a particular view is displayed on the information page by changing the values of properties in the aspect associated with the information page. Views that are made to appear and disappear in this fashion are called **conditional views**. Figure 5-17 illustrates the use of a conditional view. In this case, the radio buttons that specify the playing speed of the album (33, 45, or 78 RPM) appear only when the

user selects Vinyl as the format of the album. For more information on conditional views, see Listing 5-14 on page 5-122 and "Implementing Conditional Views" beginning on page 5-131.

Code Resources

Aspect templates can include code resources that allow developers to extend the capabilities of the templates.

The Catalogs Extension calls your code resource when certain events occur that affect the aspect with which the code resource is associated. Such events include user actions, such as the user clicking a button in an information page or dropping a file on a catalog object, and administrative events, such as initialization or a query as to whether a control should be drawn as enabled. The code resource may call the CE to perform a variety of services, such as returning information, converting one data type to another, or updating an information page.

The routine selectors and parameters that the CE passes to your code resource are described in "Functions You Can Provide as Part of Your Code Resource" beginning on page 5-148. The CE-provided routines that your code resource can call are described in "CE-Provided Functions That Your Code Resource Can Call" beginning on page 5-196.

How the Catalogs Extension Saves New Values

When the user closes an information page or makes another information page the active one, the Catalogs Extension checks all of the visible views in the information page the user just closed or left. If the user has changed any of the properties associated with those views, the Catalogs Extension saves the new values.

For each property, the CE first calls the code resource for the aspect from which the property came with the `kDETcmdValidateSave` routine selector (page 5-168). If your code resource does not return an error for any of the changed properties, then for each of these properties, the CE finds all of the lookup-table patterns that include that property. The CE processes those lookup-table patterns to write attribute values. Any of the lookup-table patterns can contain custom elements that you define; in that case, the CE calls your code resource to process those elements.

For a property to be saved, it must both be in a visible view and be marked as changed, or it must be in a lookup-table pattern with another property for which those two conditions are met.

If the user makes a change to a sublist value, the CE saves the change as soon as the user leaves the item and clicks somewhere else on the screen. The CE uses the lookup-table pattern in the main aspect for items of the type changed to process the change. The CE does not call your `kDETcmdValidateSave` routine for changes in sublists.

Property Value Synchronization

The Catalogs Extension checks a catalog system flag periodically to see if the data in the catalog system has changed. If it has, the Catalogs Extension processes the lookup tables of all the aspects for open information pages, recalculating all the properties derived from the catalog system. The CE then updates the aspects and open information pages accordingly. At the time the CE checks for changes, it calls the aspects' code resources with the `kDETcmdShouldSync` routine selector (page 5-185). If you have derived any properties from data outside the catalog system or from records or attributes other than the one to which your aspect applies and you have reason to believe their values have changed, your code resource should tell the CE to update all the properties, which it will then do whether data in the catalog system has changed or not.

When the CE updates all the property values in an aspect—either because data in the catalog system has changed or because your code resource told it to—the CE calls your code resource with the `kDETcmdDoSync` routine selector (page 5-186). If your code resource has supplied any of the property values, you should update your sublist items and your other properties.

When the CE synchronizes a sublist, it first marks every item in the list as "unseen." The CE then reads in all the attribute values mentioned in the lookup tables and calls the code resource's `kDETcmdDoSync` routine. The code resource should update any sublist items that it supplied. For each attribute the CE processes that the lookup table lists as for use in the sublist, the CE checks the type and creation ID of the item to see if it is already in the sublist. If the item is in the sublist, the CE updates it and marks it as "seen." If it's not there, the CE creates a new item, adds it to the sublist, and marks it as "seen." After processing all such items, The CE removes from the sublist any items that are still marked "unseen."

Drags and Drops

The user can drag HFS and catalog objects—such as files, information cards, records, and attributes—and drop them on records, attributes, or sublists. In each case, the Catalogs Extension determines the most appropriate action based on the type of object dragged, the type of object on which the item was dropped, and instructions in the aspect templates of the dragged and destination objects (see note at end of this section).

For example, if the user drags an information card and drops it on a record in a catalog, the CE checks every aspect template available that applies to that record for resources that provide drop instructions. The CE then determines what to do (perhaps to add an alias to the information card to the sublist of the record) and calls the code resource (if any) in each aspect for the record. The code resource can take some other action, carry out the action recommended by the CE, or take no action and return control to the CE. See the descriptions of the `kDETcmdDropQuery` (page 5-172) and `kDETcmdDropMeQuery` (page 5-170) routines for more information on how code resources handle drags and drops.

If the user drags more than one item onto a catalog object, the CE collects all of the operations and executes them in batches—for instance, the CE might copy half the items and use the other half to invoke custom operations in a destination code resource.

In addition, there may be more than one aspect in the destination that can accept a drop. For example, a Group record includes an aspect that can add a user to the group and another aspect that can mail an information card to a group. Each aspect includes a string that the CE can present to the user to confirm the action.

The aspect template signature resource includes a drop-check Boolean value and a drop-operation order number. If there is only one aspect that can handle the drop and you specify `dropCheckAlways` as the Boolean value, the CE displays a dialog box to let the user confirm the action. You must provide the prompt string for the dialog box in an aspect template resource. If you specify `dropCheckConflicts` as the Boolean value, the CE handles the drop without checking with the user. If there is more than one aspect that can handle the drop, the CE displays a confirmation dialog box for the option offered by the aspect that has the lowest drop-operation order number.

The resources that you must provide in an aspect template to support drags and drops are described in “Supporting Drags and Drops” beginning on page 5-98.

How the Catalogs Extension decides which drop operation to perform

The process the Catalogs Extension goes through to decide which drop operation is appropriate is fairly complex. First, the CE finds every aspect that might accept the drop (that is, every aspect of the destination object that has drag-in resources or a code resource). For each one, the CE figures out what operation the aspect wants to perform by looking at where the aspect is located (for example, whether to move an object or copy it depends on whether the destination is on the same volume as the original location), the access masks (can the CE delete the original, for example?), the drag-in and drag-out resources in the aspects of the source and destination containers, and the code resource (if any).

The CE calls the code resource in the aspect of the object being dropped with the `kDETcmdDropMeQuery` routine selector and then calls the code resource of the destination aspect with the `kDETcmdDropQuery` routine selector. These routines can specify that a different action be performed in response to the drop. In both bases, if the code resource does not handle the request, the CE calls the code resource of the object’s container (if the object is an attribute, its container is a record).

At this point, the CE has a list of possible operations—one for each possible destination. If the user has dragged several objects, the CE repeats this process until it has such a list for each item being dropped. Then the CE groups together all the items that share the same set of possible operations. For each group for which there’s a choice of possible operations, the CE selects the operation with the lowest drop-operation order number and displays a dialog box asking the user whether to perform the operation.

The operation can be a move, a copy (also referred to as a drag), the creation of an alias, or the sending of a property command to a code resource. If the operation is a property command specified by the destination's code resource (in response to the `kDETcmdDropQuery` request), then the CE sends the property command to the destination's code resource. If the operation is a property command specified by the dragged object's code resource (in response to the `kDETcmdDropMeQuery` request), then the CE sends a property command to that code resource. If the operation is a move, copy, or creation of an alias, then the CE carries out the operation itself, displaying status windows as appropriate. u

Writing AOCE Templates

This section provides some simple examples of source code for AOCE templates. The templates shown here create a new record type that stores information about a user's collection of recording albums. A user can place these templates in the System Folder to add a new record type and information pages to his or her personal catalog.

A set of AOCE templates includes a large number of resources of several different types. To understand this section you must be familiar with the definitions and concepts provided in the preceding sections of this chapter. In addition, the resource types used in this section are all described fully in "AOCE Templates Reference" beginning on page 5-73 and "Code Resources Reference" beginning on page 5-142; cross references to the reference material are provided wherever practical. You will probably have to refer to the reference material frequently while reading this section. Additionally, the AOCE templates provide many features not illustrated by these examples; to learn about all these features you will have to read the reference sections in detail.

Note
All of the resource examples in this chapter are written in the syntax of the Rez resource compiler. All other code is written for the MPW C compiler. u

Defining a New Record Type or Attribute Type

When you define a new record type or attribute type, you must provide a main aspect for that record or attribute type. The main aspect includes a signature resource, a resource that specifies the record or attribute type to which the main aspect applies, and several other resources (for instance, icon resources, the text of help balloons, the text of the New item in the Catalogs menu for records or the text of the Add item in the new-attribute-item dialog box, and the name given to newly created records or the initial value for new attributes). For a full list of the required and optional resources used only by main aspect templates, see Table 5-4 on page 5-89. Additionally, main aspect templates can contain any of the resources found in other aspect templates. For a full list of resources that can be used by aspect templates, see Table 5-1 on page 5-78.

Listing 5-1 shows a main aspect template for a new record type. Because Listing 5-1 is for a main aspect template for a record, it includes only resources that are specific to the main aspect. Separating the main aspect template from other aspect templates for a record has certain advantages. This segregation of resources into main aspect templates and other aspect templates allows the Catalogs Extension to load into memory only the aspects that are needed at a given time and makes it easier for developers and users to create new information pages for an existing record type.

Note

In order to ensure uniqueness of attribute and record types, this and other code listings in this chapter use WAVE, the application signature of the fictitious application SurfWriter, as the first part of all attribute and record type names. u

Listing 5-1 Main aspect template

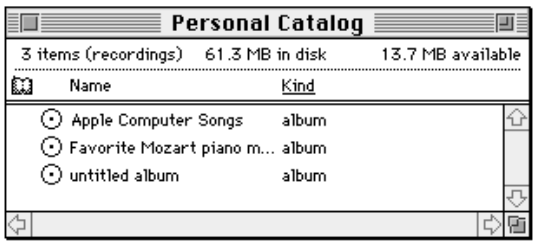
```
// File: AlbumMainAspect.r

#include "Types.r"
#include "OCETemplates.h"
#include "OCE.r"

#define kAlbumMainAspect   kDETFirstID

// Aspect template signature resource

resource 'deta' (kAlbumMainAspect, purgeable) {
   0,                   // drop-operation order (not used in this aspect)
   dropCheckAlways,     // drop-check flag (not used in this aspect)
   isMainAspect         // is the main aspect
};

// Template name

resource 'rstr' (kAlbumMainAspect + kDETTemplateName, purgeable) {
   "WAVE Album Main Aspect"   // start name with application signature
};

// Record type to which this template applies

resource 'rstr' (kAlbumMainAspect + kDETRecordType, purgeable) {
   "WAVE Album"   // start with application signature
};
```

```
// Categories to which this record type belongs

resource 'rst#' (kAlbumMainAspect + kDETAspectCategory,purgeable)
   {{
   "Recordings"
   }};

// String to be displayed in the Catalogs menu

resource 'rstr' (kAlbumMainAspect + kDETAspectNewMenuName, purgeable) {
   "New Album"
};

// Name given to new record of this type

resource 'rstr' (kAlbumMainAspect + kDETAspectNewEntryName, purgeable) {
   "untitled album"
};

// Record kind as shown in a sublist

resource 'rstr' (kAlbumMainAspect + kDETAspectKind, purgeable) {
   "album"
};

// Text for help balloons

resource 'rstr' (kAlbumMainAspect + kDETAspectWhatIs, purgeable) {
   "Album\n\nA description of an album. Open this icon to display information
    about the album."
};

resource 'rstr' (kAlbumMainAspect + kDETAspectAliasWhatIs, purgeable) {
   "Album alias\n\nAn alias to a description of an album. Open this alias to
    display information about the album."
};

// Icons

include "AlbumIcons" 'ICN#'(0) as
      'ICN#'(kAlbumMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'icl4'(0) as
      'icl4'(kAlbumMainAspect + kDETAspectMainBitmap, purgeable);
```

```
include "AlbumIcons" 'icl8'(0) as
      'icl8'(kAlbumMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'ics#'(0) as
      'ics#'(kAlbumMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'ics4'(0) as
      'ics4'(kAlbumMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'ics8'(0) as
      'ics8'(kAlbumMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'SICN'(0) as
      'SICN'(kAlbumMainAspect + kDETAspectMainBitmap, purgeable);
```

The main aspect in Listing 5-1 makes it possible for the user to create a new record of type Album by choosing New Album from the Catalogs menu. A new record of this type has the name “untitled album” until the user renames it. Records of this type are displayed in the catalog window when the user chooses Recordings from the View menu. Figure 5-18 shows a personal catalog window displaying records of type Album. The icons displayed in this window are provided by the icon resources in the main aspect.

Figure 5-18 Catalog window displaying the record type defined by Listing 5-1



Defining the Contents of the New Record Type or Attribute Type

When you define a new record type or attribute type, you must define its contents and provide a mapping between attributes and properties. In the case of a new attribute, you would normally include this information in the main aspect template. In the case of a record, however, you usually provide a separate aspect template to support each information page.

Listing 5-2 shows an aspect template for the Album record type defined in Listing 5-1. This aspect template defines several properties, provides a lookup table mapping attributes to properties, and provides default values and help-balloon strings for each property type. The lookup table maps a single attribute (`"WAVE Album General Info"`) into four properties (`prArtist`, `prTitle`, `prComments`, and `prFormat`) and a second attribute (“WAVE Album Cover”) into another property (`prCover`). Note that this mapping also works in reverse: The first time the user provides new values for the properties and closes the information page, the Catalogs Extension creates the attributes

and places them in the record. Lookup tables are described in “The Lookup-Table Resource” beginning on page 5-105.

Listing 5-2 Defining properties for a record

```
/*
   File:    AlbumMainAspect.r
*/

#include "Types.r"
#include "OCETemplates.h"
#include "OCE.r"

#define kAlbum1stInfoPageAspect  kDETSecondID

// Aspect template signature resource

resource 'deta' (kAlbum1stInfoPageAspect, purgeable) {
   0,                   // drop-operation order (not used in this aspect)
   dropCheckAlways,     // drop-check flag (not used in this aspect)
   notMainAspect        // not the main aspect
};

// Template name

resource 'rstr' (kAlbum1stInfoPageAspect + kDETTemplateName, purgeable) {
   "WAVE Album First Info Page Aspect" //start with application signature
};

// Record type to which this template applies

resource 'rstr' (kAlbum1stInfoPageAspect + kDETRecordType, purgeable) {
   "WAVE Album"                        //start with application signature
};

// Properties

#define prTitle      kDETFirstDevProperty
#define prArtist     kDETFirstDevProperty + 1
#define prComments   kDETFirstDevProperty + 2
#define prFormat     kDETFirstDevProperty + 3
#define prCover      kDETFirstDevProperty + 4
```

```
// Lookup table - maps attributes to properties

resource 'dett' (kAlbum1stInfoPageAspect + kDETAspectLookup, purgeable) {
   {
   {"WAVE Album General Info"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
         'rstr', prTitle, 0;
         'rstr', prArtist, 0;
         'rstr', prComments, 0;
         'word', prFormat, 0;
      };
   {"WAVE Album Cover"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      { 'rest', prCover , 0 };
   }
};

// Default property values

resource 'rstr' (kAlbum1stInfoPageAspect + prTitle) {
   "<Put the album’s full title here.>"
};

resource 'rstr' (kAlbum1stInfoPageAspect + prArtist) {
   "<Put the album’s recording artist or group here.>"
};

resource 'rstr' (kAlbum1stInfoPageAspect + prComments) {
   "<Put comments here. Did you like it? What's the best track?>"
};

resource 'detn' (kAlbum1stInfoPageAspect + prFormat) {
   1
};

include "AlbumIcons" 'detb'(0) as
   'detb'(kAlbum1stInfoPageAspect + prCover, purgeable);


// Text for help balloons for the properties

resource 'rst#' (kAlbum1stInfoPageAspect + kDETAspectBalloons,purgeable) {
```

```
   {
   "The full title.", "The full title. Uneditable because the record is
    locked or access is restricted.",
   "The artist or group.", "The artist or group. Uneditable because the
    record is locked or access is restricted.",
   "Comments.", Comments. Uneditable because the record is locked or
    access is restricted.",
   "Format.", "Format. Uneditable because the record is locked or
    or access is restricted."
   "Album's cover.", Album's cover. Uneditable because the record is locked
    or access is restricted."
   }
};
```

To display the properties defined in Listing 5-2, you must provide an information page.

Laying Out an Information Page

Once you have defined a new record type or attribute type, or even if you just want to display the contents of an existing record type or attribute type in a new way, you have to provide one or more information page templates that tell the Catalogs Extension how to display the information in the record or attribute.

Listing 5-3 provides an icon and title for the information page and lays out the way in which the properties are displayed. The view list specifies the location and type of each field used to display a property value. View lists are described in "View Lists" beginning on page 5-123.

Listing 5-3 A simple information page

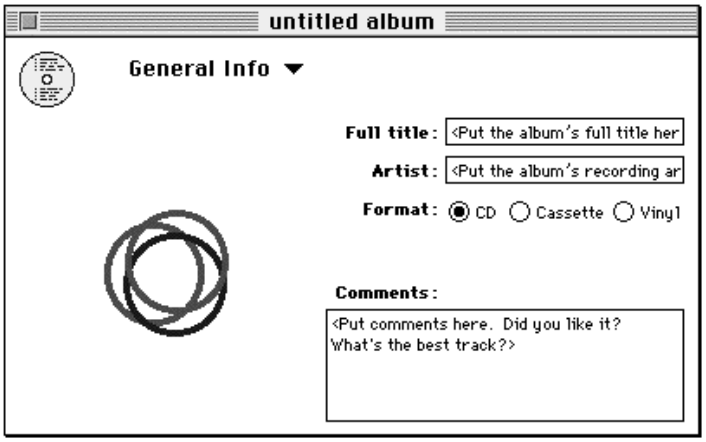
```
#define kAlbumInfoPage  kDETThirdID


resource 'deti' (kAlbumInfoPage, purgeable) {
   1000,                            // sort order
   {0, 0, 0, 0},                    // rectangle to put sublist in
   selectFirstText,                 // select the first text
                                    //  field when info-page opens


   {                                // the header view list
      kDETNoProperty, kDETNoProperty, kAlbumInfoPage;
   },
   {                                // no subview view lists
```

```
   }
};

resource 'rstr' (kAlbumInfoPage + kDETTemplateName, purgeable) {
   "WAVE Album 1st Info Page"          // start with application signature
};

resource 'rstr' (kAlbumInfoPage + kDETInfoPageName, purgeable) {
   "General Info"
};

// Associate this information page with records of this type
//  and with the aspect

resource 'rstr' (kAlbumInfoPage + kDETRecordType, purgeable) {
   "WAVE Album"
};

resource 'rstr' (kAlbumInfoPage + kDETInfoPageMainViewAspect, purgeable) {
   "WAVE Album First Info Page Aspect"
};

// View list

#define kCoverTop             (kDETSubpageIconBottom + 8)
#define kCoverLeft            (kDETSubpageIconLeft - 2)
#define kCoverBottom          (kCoverTop + 175)
#define kCoverRight           (kCoverLeft + 175)

#define k1stColumnLeft        (kCoverRight + 4)
#define k1stColumnRight       (k1stColumnLeft + 65)
#define k2ndColumnLeft        (k1stColumnRight + 4)
#define k2ndColumnRight       (kDETRecordInfoWindWidth - 8)

#define kTitleTop             (kCoverTop)
#define kTitleBottom          (kTitleTop + kDETAppFontLineHeight + 4)
#define kArtistTop            (kTitleBottom + 6)
#define kArtistBottom         (kArtistTop + kDETAppFontLineHeight + 4)
#define kFormatTop            (kArtistBottom + 6)
#define kFormatBottom         (kFormatTop + kDETAppFontLineHeight + 4)
#define kNumFormats           (3)
#define kCDRadioLeft          (k2ndColumnLeft)
#define kCDRadioRight         (kCDRadioLeft + 35)
```

```
#define kCassetteRadioLeft    (kCDRadioRight)
#define kCassetteRadioRight   (kCassetteRadioLeft + 60)
#define kVinylRadioLeft       (kCassetteRadioRight)
#define kVinylRadioRight      (k2ndColumnRight)
#define kCommentsTop          (kFormatBottom + 32)
#define kCommentsLabelBottom  (kCommentsTop + kDETAppFontLineHeight + 4)
#define kCommentsBottom       (kCoverBottom)


resource 'detv' (kAlbumInfoPage, purgeable) {
   {
   kDETSubpageIconRect, kDETNoFlags, kDETAspectMainBitmap,
   Bitmap { kDETLargeIcon };


   {kTitleTop, k1stColumnLeft, kTitleBottom, k1stColumnRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETRight, kDETBold, "Full title:" };

   {kTitleTop - 2, k2ndColumnLeft, kTitleBottom - 2, k2ndColumnRight},
   kDETEnabled, prTitle,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
    kDETNormal };


   {kArtistTop, k1stColumnLeft, kArtistBottom, k1stColumnRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETRight, kDETBold, "Artist:" };

   {kArtistTop - 2, k2ndColumnLeft, kArtistBottom - 2, k2ndColumnRight},
   kDETEnabled, prArtist,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
    kDETNormal };


   {kFormatTop, k1stColumnLeft, kFormatBottom, k1stColumnRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETRight, kDETBold, "Format:" };
```

```
    {kFormatTop, kCDRadioLeft, kFormatBottom, kCDRadioRight},
    kDETEnabled, prFormat,
    RadioButton { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
     kDETNormal, "CD", prFormat, 1 };

    {kFormatTop, kCassetteRadioLeft, kFormatBottom, kCassetteRadioRight},
     kDETEnabled, prFormat,
    RadioButton { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
     kDETNormal, "Cassette", prFormat, 2 };

    {kFormatTop, kVinylRadioLeft, kFormatBottom, kVinylRadioRight},
     kDETEnabled, prFormat,
    RadioButton { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
     kDETNormal, "Vinyl", prFormat, 3 };

    {kCommentsTop, k1stColumnLeft, kCommentsLabelBottom, k1stColumnRight},
     kDETNoFlags, kDETNoProperty,
    StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
     kDETRight, kDETBold, "Comments:" };

    {kCommentsLabelBottom, k1stColumnLeft, kCommentsBottom - 2,
     k2ndColumnRight},
    kDETEnabled + kDETMultiLine, prComments,
    EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
     kDETNormal };

    { kCoverTop, kCoverLeft, kCoverBottom, kCoverRight },
     kDETNoFlags, prCover,
    EditPicture { 8 };
    }
};
```

Figure 5-19 Simple information page

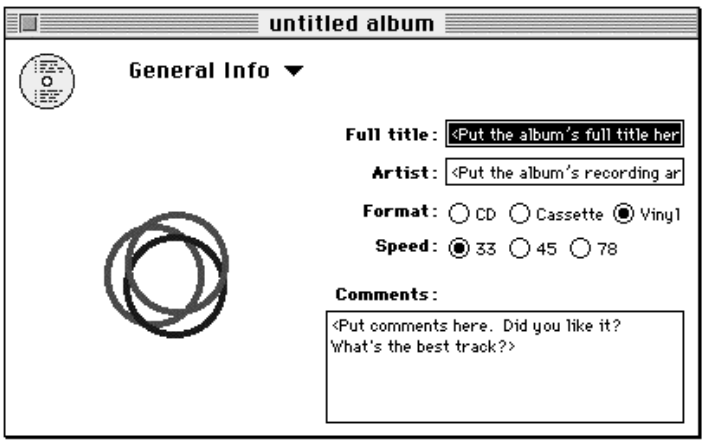


Listing 5-3 together with Listing 5-2 on page 5-34 describe the information page shown in Figure 5-19. The user can type information into the editable text fields and place a figure in the editable picture field.

Adding a Conditional View

A conditional view is one that appears in an information page only if certain conditions are met. For example, the Album information page shown in the preceding example could display radio buttons that specify the speed of the album, but only if the user selects the Vinyl radio button for album format (Figure 5-20).

Figure 5-20 Simple information page with a conditional view



To implement the conditional view shown in Figure 5-20, make the following additions to the templates:

- Add the property `prVinylSpeed` to the list of properties.

```
#define prTitle      kDETFirstDevProperty
#define prArtist     kDETFirstDevProperty + 1
#define prComments   kDETFirstDevProperty + 2
#define prFormat     kDETFirstDevProperty + 3
#define prCover      kDETFirstDevProperty + 4
#define prVinylSpeed kDETFirstDevProperty + 5
```

- Add the `prVinylSpeed` property to the lookup table.

```
resource 'dett' (kAlbum1stInfoPageAspect + kDETAspectLookup,
purgeable) {
   {
   {"WAVE Album General Info"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias,
       isNotRecordRef,
      {
         'rstr', prTitle, 0;
         'rstr', prArtist, 0;
         'rstr', prComments, 0;
         'word', prFormat, 0;
         'word', prVinylSpeed, 0;
      };
   {"Album Cover"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias,
       isNotRecordRef,
      { 'rest', prCover , 0 };
   }
};
```

- Add a default property value for the `prVinylSpeed` property.

```
resource 'detn' (kAlbumMainAspect + prVinylSpeed) {
   1
};
```

- Add help balloons.

```
resource 'rst#' (kAlbum1stInfoPageAspect + kDETAspectBalloons,
                  purgeable) {
   {
   "The full title.", "The full title. Uneditable because the
    record is locked or access is restricted.",
```

```
   "The artist or group.", "The artist or group. Uneditable
    because the record is locked or access is restricted.",
   "Comments.", Comments. Uneditable because the record is locked
    or access is restricted.",
   "Format.", "Format. Uneditable because the record is locked or
    access is restricted."
   "Album's cover.", Album's cover. Uneditable because the record
    is locked or access is restricted."
   "Record speed", Record speed. Uneditable because the record is
    locked or access is restricted."
   }
};
```

- Add a line to the information page signature resource for a second view list. Each view list has a corresponding line in the information page signature resource; each line has two property numbers and a resource ID for the view list resource. The view is displayed only if the values of the two properties are equal. In this case, the second line requires that the property `prFormat` must equal 3; that is, the value of the property (`kDETFirstConstantProperty + 3`) is the constant 3. Information page signature resources are defined and described in "Information Page Template Signature Resource" on page 5-121.

```
resource 'deti' (kAlbumInfoPage, purgeable) {
   1000,
   {0, 0, 0, 0},
   selectFirstText,
   {
      kDETNoProperty, kDETNoProperty, kAlbumInfoPage;
      prFormat, kDETFirstConstantProperty + 3, kAlbumInfoPage + 1;
   },
   {
   }
};
```

- Add the definitions and view list for the conditional view. Notice that the conditional view resource (`'detv'`) includes the identification number for the conditional view, `kAlbumInfoPage + 1`.

```
#define kConditionalTop       (kFormatBottom + 4)
#define kConditionalBottom    (kConditionalTop +
                               kDETAppFontLineHeight + 4)
#define k33RadioLeft          (k2ndColumnLeft)
#define k33RadioRight         (kCDRadioLeft + 35)
#define k45RadioLeft          (k33RadioRight)
```

```
#define k45RadioRight         (k45RadioLeft + 35)
#define k78RadioLeft          (k45RadioRight)
#define k78RadioRight         (k45RadioRight + 35)

resource 'detv' (kAlbumInfoPage + 1, purgeable) {
   {
   {kConditionalTop, k1stColumnLeft, kConditionalBottom,
    k1stColumnRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont,
      kDETApplicationFontSize, kDETRight, kDETBold, "Speed:" };

   {kConditionalTop, k33RadioLeft, kConditionalBottom,
    k33RadioRight},
   kDETEnabled, prVinylSpeed,
   RadioButton { kDETApplicationFont, kDETApplicationFontSize,
      kDETLeft, kDETNormal, "33", prVinylSpeed, 1 };

   {kConditionalTop, k45RadioLeft, kConditionalBottom,
    k45RadioRight}, kDETEnabled, prVinylSpeed,
   RadioButton { kDETApplicationFont, kDETApplicationFontSize,
      kDETLeft, kDETNormal, "45", prVinylSpeed, 2 };

   {kConditionalTop, k78RadioLeft, kConditionalBottom,
    k78RadioRight}, kDETEnabled, prVinylSpeed,
   RadioButton { kDETApplicationFont, kDETApplicationFontSize,
      kDETLeft, kDETNormal, "78", prVinylSpeed, 3 };
   }
};
```

For another example of a conditional view, see “Implementing Conditional Views” beginning on page 5-131.

Adding an Information Page With a Sublist

Listing 5-4 shows the aspect and information page templates for a second information page for the Album record. This information page (shown in Figure 5-21) provides details about the tracks on the album, including a list of all the tracks. The list of tracks is implemented as an information page sublist. Each item in the sublist is an attribute value; each attribute value includes the title and track number of a track on the album. For more information on sublists, see “Sublists” on page 5-136.

Figure 5-21 Information page with a sublist

Note that this template supports the dropping of an attribute into the sublist as a way to add an item to the sublist. The aspect template signature resource, the `kDETAspectDragInString` resource, and the `kDETAspectAttrDragIn` resource all support drops. See “Supporting Drags and Drops” beginning on page 5-98 for more information about resources that support dragging and dropping objects on templates.

Listing 5-4 An information page with a sublist

```
#include "Types.r"
#include "OCETemplates.h"
#include "OCE.r"


// This is an aspect template with this base resource ID.

#define kAlbum2ndInfoPageAspect  kDETFourthID

// Aspect template signature resource

resource 'deta' (kAlbum2ndInfoPageAspect, purgeable) {
   0,                   // drop-operation order
   dropCheckAlways,     // drop-check flag
   notMainAspect        // not the main aspect
};

// Template name

resource 'rstr' (kAlbum2ndInfoPageAspect + kDETTemplateName, purgeable) {
```

```
   "WAVE Album Second Info Page Aspect"
};



// Associate this aspect template with records of type Album.

resource 'rstr' (kAlbum2ndInfoPageAspect + kDETRecordType, purgeable) {
   "WAVE Album"
};


// Icons

include "AlbumIcons" 'ICN#'(0) as
    'ICN#'(kAlbum2ndInfoPageAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'icl4'(0) as
    'icl4'(kAlbum2ndInfoPageAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'icl8'(0) as
    'icl8'(kAlbum2ndInfoPageAspect + kDETAspectMainBitmap, purgeable);

// Aspect properties - shared between aspect and info page(s)

#define prTrackNumber         kDETFirstDevProperty
#define prNumTracks           kDETFirstDevProperty + 1
#define prPlayingTimeHours    kDETFirstDevProperty + 2
#define prPlayingTimeMinutes  kDETFirstDevProperty + 3
#define prPlayingTimeSeconds  kDETFirstDevProperty + 4


// Lookup table

// This lookup table defines the format of attribute type
//  WAVE Album Track Info. This attribute type is not displayed in a
//  sublist and so does not require a main aspect.
// Attribute values of type WAVE Track are displayed in the sublist.
//  The format of attribute type WAVE Track is defined in the main aspect
//  shown in Listing 5-5 on page 5-52.

resource 'dett' (kAlbum2ndInfoPageAspect + kDETAspectLookup, purgeable) {
   {
   {"WAVE Album Track Info"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
```

```
      {
         'word', prNumTracks, 0;
         'long', prPlayingTimeHours, 0;
         'long', prPlayingTimeMinutes, 0;
         'long', prPlayingTimeSeconds, 0;
      };

   {"WAVE Track"}, typeBinary,
      notForInput, notForOutput, useInSublist, isNotAlias, isNotRecordRef,
      {};
   }
};

// Drag and drop information (see
//  “Supporting Drags and Drops” beginning on page 5-98)

// Prompt string for drag-in dialog box
resource 'rstr' (kAlbum2ndInfoPageAspect + kDETAspectDragInString,
                  purgeable) {
   "Do you want to add %3%“^3”%the selected items% to the track address
list of *0x/the/* ^1 “^2”?"
};

// Attributes can be dragged from any kind of record (""); attributes of
//  type WAVE Track can be dragged into this record; and the new copy of the
//  attribute will be of type WAVE Track.
resource 'rst#' (kAlbum2ndInfoPageAspect + kDETAspectAttrDragIn, purgeable) {
   {
   "", "WAVE Track", "WAVE Track"
   }
};


// Sublist sorting information

// Property names in this resource appear in the View menu, and property
//  numbers tell the CE what to sort by. Positive property number is
//  alphanumeric sort; negative number is numeric sort.

resource 'detm' (kAlbum2ndInfoPageAspect + kDETAspectViewMenu, purgeable) {
   kAlbum2ndInfoPageAspect + kDETAspectViewMenu,
   {
   kDETAspectName,"by Title";
```

```
   -prTrackNumber,"by Track Number";
   }
};

// Properties in this resource are sorted in reverse order.

resource 'detp' (kAlbum2ndInfoPageAspect + kDETAspectReverseSort,
purgeable) {
   {
   prTrackNumber
   }
};

// Text for help balloons for the properties

resource 'rst#' (kAlbum2ndInfoPageAspect + kDETAspectBalloons,purgeable) {
   {
   "The number of tracks on the album.", The number of tracks on the album.
    Uneditable because the record is locked or access is restricted.",
   "The number of hours of music on the album.", "The number of hours of
    music on the album. Uneditable because the record is locked or access
     is restricted.",
   "The number of minutes of music on the album.", "The number of minutes of
    music on the album. Uneditable because the record is locked or access
     is restricted.",
   "The number of seconds of music on the album.", "The number of seconds of
    music on the album. Uneditable because the record is locked or access
     is restricted.",
   }
};
// ---------------------------------------------------------------------
//
// Album information page



#define kAlbum2ndInfoPage     kDETFifthID

#define kTitleTop             (85)
#define kTitleBottom          (kTitleTop + 12)
#define kSublistTop           (kTitleBottom + 2)
#define kSublistBottom        (kDETRecordInfoWindHeight - 40)
#define kSublistLeft          (12)
```

```
#define kSublistRight         (kDETRecordInfoWindWidth - 12)

// Information page template signature resource

resource 'deti' (kAlbum2ndInfoPage, purgeable) {
   2000,
   {kSublistTop, kSublistLeft, kSublistBottom, kSublistRight},
   selectFirstText,
// View list for main view is identified by the following line.
   {
      kDETNoProperty, kDETNoProperty, kAlbum2ndInfoPage;
   },
// View list for sublist is identified by this line.
   {
      kDETNoProperty, kDETNoProperty, kAlbum2ndInfoPage + 1;
   }
};

resource 'rstr' (kAlbum2ndInfoPage + kDETTemplateName, purgeable) {
   "WAVE Album 2nd Info Page"
};

resource 'rstr' (kAlbum2ndInfoPage + kDETInfoPageName, purgeable) {
   "Track Info"
};

// Associate this information page with records of type WAVE Album
//  and with this aspect template.

resource 'rstr' (kAlbum2ndInfoPage + kDETRecordType, purgeable) {
   "WAVE Album"
};

resource 'rstr' (kAlbum2ndInfoPage + kDETInfoPageMainViewAspect, purgeable) {
   "WAVE Album Second Info Page Aspect"
};


// View list - what you see in this information page

#define kMyFirstColumnLeft    (55)
#define kMyFirstColumnRight   (kMyFirstColumnLeft + 120)
#define kEditTextWidth        (23)
```

```
#define kSpaceBeforeEditDesc  (25)
#define kNumEditColumns       (3)
#define kMyEditColumnWidth    (70)
#define k1stEditColumnLeft    (kMyFirstColumnRight + 2)
#define k2ndEditColumnLeft    (k1stEditColumnLeft + kMyEditColumnWidth)
#define k3rdEditColumnLeft    (k2ndEditColumnLeft + kMyEditColumnWidth)
#define k4thEditColumnLeft    (k3rdEditColumnLeft + kMyEditColumnWidth)

#define kNumTracksTop         (40)
#define kNumTracksBottom      (kNumTracksTop + kDETAppFontLineHeight + 4)
#define kPlayingTimeTop       (kNumTracksBottom + 4)
#define kPlayingTimeBottom    (kPlayingTimeTop + kDETAppFontLineHeight + 4)
#define k2ndColumnRightInset  (kDETRecordInfoWindWidth - 10)

#define kButtonTop            (kSublistBottom + 15)
#define kButtonBottom         (kButtonTop + 16)
#define kOpenLeft             62
#define kOpenRight            112
#define kAddLeft              208
#define kAddRight             258
#define kRemoveLeft           270
#define kRemoveRight          320

#define kIconLeft             2
#define kNameLeft             22
#define kTrackNumberLeft      162
#define kPrefLeft             285
#define kPrefRight            305

#define kIconEntryTop         -7
#define kIconEntryBottom      9
#define kEntryTop             -5
#define kEntryBottom          9

resource 'detv' (kAlbum2ndInfoPage, purgeable) {
   {
   kDETSubpageIconRect, kDETNoFlags, kDETAspectMainBitmap,
   Bitmap { kDETLargeIcon };

   {kNumTracksTop, kMyFirstColumnLeft, kNumTracksBottom,
    kMyFirstColumnRight},
      kDETNoFlags, kDETNoProperty,
      StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
```

```
    kDETRight, kDETBold, "Number of tracks:" };

{kNumTracksTop - 2, k1stEditColumnLeft, kNumTracksBottom - 2,
 k1stEditColumnLeft + kEditTextWidth},
   kDETEnabled + kDETNumericOnly, prNumTracks,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
    kDETNormal };

{kPlayingTimeTop, kMyFirstColumnLeft, kPlayingTimeBottom,
 kMyFirstColumnRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETRight, kDETBold, "Total playing time:" };

{kPlayingTimeTop - 2, k1stEditColumnLeft, kPlayingTimeBottom - 2,
 k1stEditColumnLeft + kEditTextWidth},
   kDETEnabled + kDETNumericOnly, prPlayingTimeHours,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
    kDETNormal };

{kPlayingTimeTop, k1stEditColumnLeft + kSpaceBeforeEditDesc,
 kPlayingTimeBottom, k2ndEditColumnLeft},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETLeft, kDETNormal, "Hours" };

{kPlayingTimeTop - 2, k2ndEditColumnLeft, kPlayingTimeBottom - 2,
 k2ndEditColumnLeft + kEditTextWidth},
   kDETEnabled + kDETNumericOnly, prPlayingTimeMinutes,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
    kDETNormal };

{kPlayingTimeTop, k2ndEditColumnLeft + kSpaceBeforeEditDesc,
 kPlayingTimeBottom, k3rdEditColumnLeft},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETLeft, kDETNormal, "Minutes" };

{kPlayingTimeTop - 2, k3rdEditColumnLeft, kPlayingTimeBottom - 2,
 k3rdEditColumnLeft + kEditTextWidth},
   kDETEnabled + kDETNumericOnly, prPlayingTimeSeconds,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
    kDETNormal };
```

```
   {kPlayingTimeTop, k3rdEditColumnLeft + kSpaceBeforeEditDesc,
    kPlayingTimeBottom, k4thEditColumnLeft},
      kDETNoFlags, kDETNoProperty,
      StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
       kDETLeft, kDETNormal, "Seconds" };

   {kSublistTop - 1, kSublistLeft - 1, kSublistBottom + 1,
    kSublistRight + 1},
      kDETNoFlags, kDETNoProperty,
      Box { kDETUnused };

   {kTitleTop, kSublistLeft + kNameLeft, kTitleBottom,
    kSublistLeft + kTrackNumberLeft - 2},
      kDETNoFlags, kDETAspectName,
      StaticCommandTextFromView { kDETDefaultFont, kDETDefaultFontSize,
       kDETLeft, kDETUnderline, "Title", kDETChangeViewCommand,  - 1};

   {kTitleTop, kSublistLeft + kTrackNumberLeft, kTitleBottom,
    kSublistLeft + kPrefLeft - 2},
      kDETNoFlags, prTrackNumber,
      StaticCommandTextFromView { kDETDefaultFont, kDETDefaultFontSize,
       kDETLeft, kDETNormal, "Track Number", kDETChangeViewCommand, - 2 };

   {kButtonTop, kOpenLeft, kButtonBottom, kOpenRight},
      kDETNoFlags, kDETOpenSelectedItems,
      Button { kDETApplicationFont, 10, kDETCenter, kDETNormal, "Open",
       kDETOpenSelectedItems };

   {kButtonTop, kAddLeft, kButtonBottom, kAddRight},
      kDETNoFlags, kDETAddNewItem,
      Button { kDETApplicationFont, 10, kDETCenter, kDETNormal, "Add…",
       kDETAddNewItem };

   {kButtonTop, kRemoveLeft, kButtonBottom, kRemoveRight},
      kDETNoFlags, kDETRemoveSelectedItems,
      Button { kDETApplicationFont, 10, kDETCenter, kDETNormal, "Remove",
       kDETRemoveSelectedItems };
   }
};

// View list for sublist
```

```
resource 'detv' (kAlbum2ndInfoPage + 1, purgeable) {
   {
   {kIconEntryTop, kIconLeft, kIconEntryBottom, kNameLeft-4},
      kDETHilightIfSelected, kDETAspectMainBitmap,
      Bitmap { kDETMiniIcon };

   {kEntryTop, kNameLeft, kEntryBottom, kTrackNumberLeft - 2},
      kDETHilightIfSelected + kDETDynamicSize, kDETAspectName,
      EditText { kDETDefaultFont, kDETDefaultFontSize, kDETLeft,
       kDETNormal };

   {kEntryTop, kTrackNumberLeft, kEntryBottom, kPrefLeft - 2},
      kDETHilightIfSelected + kDETDynamicSize, prTrackNumber,
      EditText { kDETDefaultFont, kDETDefaultFontSize, kDETLeft,
       kDETNormal };
   }
};
```

Writing a Main Aspect and Information Page for an Attribute

The information page in Listing 5-4 on page 5-44 allows a user to add a new attribute of type Track. To make this possible, you have to provide a main aspect for attributes of that type. To let the user see the contents of the attribute, you need to provide an information page (see Figure 5-5 on page 5-9). Listing 5-5 shows the main aspect template and information page template for attributes of type Track. Because this is an attribute, the main aspect template contains all the properties needed by the information page in addition to the resources required for a main aspect template.

Listing 5-5 Attribute main aspect and information page

```
#include "Types.r"
#include "OCETemplates.h"
#include "OCE.r"
#include "Track.h"

#define kDETSixthID     (1000 + 5 * kDETIDSep)
#define kTrackAspect    (kDETSixthID + kDETIDSep)
#define kTrackInfoPage  (kDETSixthID + (2 * kDETIDSep))


// The aspect template

resource 'deta' (kTrackAspect, purgeable) {
```

```
   0,                // drop-operation order
   dropCheckAlways,  // drop-check flag
   isMainAspect      // is the main aspect
};

resource 'rstr' (kTrackAspect + kDETTemplateName, purgeable) {
   "WAVE Track Aspect"
};

resource 'rstr' (kTrackAspect + kDETAttributeType, purgeable) {
   "WAVE Track"
};

resource 'rstr' (kTrackAspect + kDETAspectKind, purgeable) {
   "Track"
};

resource 'rstr' (kTrackAspect + kDETAspectWhatIs, purgeable) {
   "Track\n\nA track on an album."
};

resource 'rst#' (kTrackAspect + kDETAspectCategory, purgeable)
   {{
   "Recordings"
   }};

resource 'rstr' (kTrackAspect + kDETAspectNewMenuName, purgeable) {
   "New Track"
};

#define prTrackNumber      kDETFirstDevProperty
#define prTrackMinutes     (kDETFirstDevProperty + 1)
#define prTrackSeconds     (kDETFirstDevProperty + 2)
#define prTrackComposer    (kDETFirstDevProperty + 3)
#define prTrackComments    (kDETFirstDevProperty + 4)

// Default values for a newly created attribute

data 'detb' (kTrackAspect + kDETAspectNewValue, purgeable) {
   $"626E 7279"                          // tag (bnry)
   $"0000 0001"                          // prTrackNumber (1)
   $"0000 0000"                          // prTrackMinutes (1)
   $"0000 0000"                          // prTrackSeconds (1)
```

```
   $"0000 0007 3C74 6974 6C65 3E"         // kDETAspectName (<title>)
   $"0000 000A 3C63 6F6D 706F 7365 723E"  // composer (<composer>)
   $"0000 000A 3C63 6F6D 6D65 6E74 733E"  // comments (<comments>)
};

// Lookup table

resource 'dett' (kTrackAspect + kDETAspectLookup, purgeable) {
   {
   {"WAVE Track"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
      'long', prTrackNumber, 0;
      'long', prTrackMinutes, 0;
      'long', prTrackSeconds, 0;
      'rstr', kDETAspectName, 0;
      'rstr', prTrackComposer, 0;
      'rstr', prTrackComments, 0
      };
   }
};

// Icons
include "TrackIcons" 'ICN#'(0) as 'ICN#'(kTrackAspect + kDETAspectMainBitmap,
       purgeable);
include "TrackIcons" 'icl4'(0) as 'icl4'(kTrackAspect + kDETAspectMainBitmap,
       purgeable);
include "TrackIcons" 'icl8'(0) as 'icl8'(kTrackAspect + kDETAspectMainBitmap,
       purgeable);
include "TrackIcons" 'ics#'(0) as 'ics#'(kTrackAspect + kDETAspectMainBitmap,
       purgeable);
include "TrackIcons" 'ics4'(0) as 'ics4'(kTrackAspect + kDETAspectMainBitmap,
       purgeable);
include "TrackIcons" 'ics8'(0) as 'ics8'(kTrackAspect + kDETAspectMainBitmap,
       purgeable);
include "TrackIcons" 'SICN'(0) as 'SICN'(kTrackAspect + kDETAspectMainBitmap,
       purgeable);

// ---------------------------------------------------------------------------

// Information page

#define kTrackNumberTop          (50)
```

```
#define kTrackNumberBottom       (kTrackNumberTop + kDETAppFontLineHeight
                                  + 4)
#define kTrackPlayingTimeTop     (kTrackNumberBottom + 4)
#define kTrackPlayingTimeBottom  (kTrackPlayingTimeTop +
                                  kDETAppFontLineHeight + 4)
#define kTrackComposerTop        (kTrackPlayingTimeBottom + 4)
#define kTrackComposerBottom     (kTrackComposerTop +
                                  kDETAppFontLineHeight + 4 +
                                  kDETAppFontLineHeight + 4 +
                                  kDETAppFontLineHeight + 4)
#define kTrackCommentsTop        (kTrackComposerBottom + 4)
#define kTrackCommentsBottom     (kTrackCommentsTop +
                                  kDETAppFontLineHeight + 4 +
                                  kDETAppFontLineHeight + 4 +
                                  kDETAppFontLineHeight + 4 +
                                  kDETAppFontLineHeight + 4 +
                                  kDETAppFontLineHeight + 4)

#define kTrackEditTextWidth      (23)
#define kTrackSpaceBeforeEditDesc(25)
#define kTrack1stColumnLeft      (4)
#define kTrack1stColumnRight     (kDETAttributeInfoWindWidth / 2 - 20)
#define kTrack2ndColumnLeft      (kTrack1stColumnRight + 4)
#define kTrack2ndColumnRight     (kDETAttributeInfoWindWidth - 8)
#define kTrackSecondsColumnLeft  (kTrack2ndColumnLeft +
                                  kTrackSpaceBeforeEditDesc + 40)

resource 'deti' (kTrackInfoPage, purgeable) {
   1000,
   {0, 0, 0, 0},
   selectFirstText,
   {
   kDETNoProperty, kDETNoProperty, kTrackInfoPage;
   },
   {
   }
};

resource 'rstr' (kTrackInfoPage + kDETTemplateName, purgeable) {
   "WAVE Track Info Page"
};

resource 'rstr' (kTrackInfoPage + kDETAttributeType, purgeable) {
```

```
   "WAVE Track"
};

resource 'rstr' (kTrackInfoPage + kDETInfoPageName, purgeable) {
   "Track Info"
};

resource 'rstr' (kTrackInfoPage + kDETInfoPageMainViewAspect, purgeable) {
   "WAVE Track Aspect"
};

// View list

resource 'detv' (kTrackInfoPage, purgeable) {
   {
   kDETSubpageIconRect, kDETNoFlags, kDETAspectMainBitmap,
   Bitmap { kDETLargeIcon };

   {kTrackNumberTop, kTrack1stColumnLeft, kTrackNumberBottom,
       kTrack1stColumnRight},
      kDETNoFlags, kDETNoProperty,
      StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
       kDETRight, kDETBold, "Track Number:" };

   {kTrackNumberTop - 2, kTrack2ndColumnLeft, kTrackNumberBottom - 2,
       kTrack2ndColumnLeft + kTrackEditTextWidth},
      kDETEnabled + kDETNumericOnly, prTrackNumber,
      EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
       kDETNormal };

   {kTrackPlayingTimeTop, kTrack1stColumnLeft, kTrackPlayingTimeBottom,
       kTrack1stColumnRight},
      kDETNoFlags, kDETNoProperty,
      StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
       kDETRight, kDETBold, "Playing Time:" };

   {kTrackPlayingTimeTop - 2, kTrack2ndColumnLeft,
       kTrackPlayingTimeBottom - 2,
       kTrack2ndColumnLeft + kTrackEditTextWidth},
      kDETEnabled + kDETNumericOnly, prTrackMinutes,
      EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
       kDETNormal };
```

```
{kTrackPlayingTimeTop, kTrack2ndColumnLeft + kTrackSpaceBeforeEditDesc,
    kTrackPlayingTimeBottom, kTrackSecondsColumnLeft},
   kDETNoFlags, kDETNoProperty,
StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETLeft, kDETNormal, "Mins" };

{kTrackPlayingTimeTop - 2, kTrackSecondsColumnLeft,
    kTrackPlayingTimeBottom - 2, kTrackSecondsColumnLeft +
    kTrackEditTextWidth},
   kDETEnabled + kDETNumericOnly, prTrackSeconds,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
    kDETNormal };

{kTrackPlayingTimeTop, kTrackSecondsColumnLeft +
    kTrackSpaceBeforeEditDesc, kTrackPlayingTimeBottom,
    kTrack2ndColumnRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETLeft, kDETNormal,"Secs" };

{kTrackComposerTop, kTrack1stColumnLeft, kTrackComposerBottom,
    kTrack1stColumnRight},
   kDETNoFlags, kDETNoProperty,
StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETRight, kDETBold,"Composer:" };

{kTrackComposerTop - 2, kTrack2ndColumnLeft, kTrackComposerBottom - 2,
    kTrack2ndColumnRight},
   kDETEnabled + kDETMultiLine, prTrackComposer,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
    kDETNormal };

{kTrackCommentsTop, kTrack1stColumnLeft, kTrackCommentsBottom,
    kTrack1stColumnRight},
   kDETNoFlags, kDETNoProperty,
StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETRight, kDETBold,"Comments:" };

{kTrackCommentsTop - 2, kTrack2ndColumnLeft, kTrackCommentsBottom - 2,
    kTrack2ndColumnRight},
   kDETEnabled + kDETMultiLine, prTrackComments,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
```

```
        kDETNormal };
    }
};
```

Creating a Custom Information Page Window

The aspect and information page templates in Listing 5-6 define a new AOCE record type and the information page that displays the record's contents. The user can use this record type to store information about collections of recording albums. The information page lists the albums in the collection. Because an AOCE record cannot contain another record, the AOCE record type Album Collection actually contains aliases to records of type Album.

The information page consists of a sublist listing the albums in the collection. The information page window is a custom size, defined by the `'detw'` resource with a resource ID of `kCollectionAspect + kDETAspectInfoPageCustomWindow` (see page 5-97 for a description of this resource). This resource specifies the flag `discludePopup`, so the Catalogs Extension does not include a pop-up menu in the window. The template does not add a custom pop-up menu either. Therefore, no one can add any more information pages to this information page window, because the user would have no way of selecting which page to look at. For this reason, a single aspect template is used for both the main aspect and the information page aspect for this new record type.

Notice also that the aspect template for the Album Collection record type includes a view list. Ordinarily, you can put view lists only in information page templates, not in aspect templates. However, because this aspect template defines a custom information page window, you can include a view list with a resource ID of `kCollectionAspect + kDETAspectInfoPageCustomWindow`. The views defined by this view list appear in every information page associated with this main aspect. (In Listing 5-6, there's only one information page, so the view list could be placed in either the aspect or information page template.)

Listing 5-6 Templates for a custom information page

```
#include "Types.r"
#include "OCETemplates.h"
#include "OCE.r"

#define kCollectionAspect     (kDETFifthID + (3 * kDETIDSep))
#define kCollectionInfoPage   (kDETFifthID + (4 * kDETIDSep))

#define kGeneva 3
#define kSystemFont 0

// Page layout defines
```

```
#define iconColumnWidth       16
#define nameColumnWidth       132
#define kindColumnWidth       86
#define spaceBetweenColumns   2

#define iconColumnLeft        0
#define iconColumnRight       (iconColumnLeft + iconColumnWidth)
#define nameColumnLeft        (iconColumnRight + spaceBetweenColumns)
#define nameColumnRight       (nameColumnLeft + nameColumnWidth)
#define kindColumnLeft        nameColumnRight
#define kindColumnRight       (kindColumnLeft + kindColumnWidth)

#define sublistIconTop        (-7)
#define sublistIconBottom     (9)
#define sublistTextTop        (-6)
#define sublistTextBottom     (8)
#define sublistTitleTop       (35)
#define sublistTitleBottom    (sublistTitleTop + 12)

#define windowHeight          280
#define windowWidth           (kindColumnLeft + kindColumnWidth + 15 + 16)

#define sublistTopBound       (sublistTitleBottom + 2)
#define sublistBottomBound    (windowHeight - 12)
#define sublistLeftBound      12
#define sublistRightBound     (windowWidth - 12)

#define kPageBitmapLeft       (11)
#define kPageBitmapRight      (kPageBitmapLeft + 16)
#define kPageBitmapTop        7
#define kPageBitmapBottom     23



// Aspect template; serves as both main aspect and information page aspect.
//
resource 'deta' (kCollectionAspect, purgeable) {
   0,                   // drop-operation order
   dropCheckConflicts,  // drop-check flag
   isMainAspect         // is the main aspect
};
```

```
resource 'rstr' (kCollectionAspect + kDETTemplateName, purgeable) {
   "WAVE Album Collection Aspect"
};

resource 'rstr' (kCollectionAspect + kDETRecordType, purgeable) {
   "WAVE Album Collection"
};

resource 'rstr' (kCollectionAspect + kDETAspectKind, purgeable) {
   "album collection"
};

resource 'rstr' (kCollectionAspect + kDETAspectWhatIs, purgeable) {
   "Album Collection\n\nA collection of albums.Open this icon to display
    information about the collection."
};

resource 'rstr' (kCollectionAspect + kDETAspectAliasKind, purgeable) {
   "album collection alias"
};

resource 'rstr' (kCollectionAspect + kDETAspectAliasWhatIs, purgeable) {
   "Album Collection alias\n\This is an alias to a collection of albums.
    Open this alias to display information about the collection."
};

// Record category; this record type is assigned to the same category as the
//  Album record type.

resource 'rst#' (kCollectionAspect + kDETAspectCategory,purgeable)
   {{
   "Recordings"
   }};

// Define a custom information page window.

resource 'detw' (kCollectionAspect + kDETAspectInfoPageCustomWindow,
                  purgeable) {
   { 0, 0, windowHeight, windowWidth },
   discludePopup
};

// View list for views to appear in all information pages for this
```

```
//  main aspect

resource 'detv' (kCollectionAspect + kDETAspectInfoPageCustomWindow,
                  purgeable)
   {
   {
   {6, kPageBitmapRight + 8, 25, kPageBitmapRight + 8 + 166},
    kDETNoFlags, kDETInfoPageNumber,
      Menu {kSystemFont, 12, kDETLeft, kDETNormal, "", kDETInfoPageNumber,
            kDETInfoPageNumber };
   };
   };

resource 'rstr' (kCollectionAspect + kDETAspectNewMenuName, purgeable) {
   "New Album Collection"
};

resource 'rstr' (kCollectionAspect + kDETAspectNewEntryName, purgeable) {
   "untitled album collection"
};

include "AlbumCollectionIcons" 'ICN#'(0) as 'ICN#'(kCollectionAspect +
kDETAspectMainBitmap, purgeable);
include "AlbumCollectionIcons" 'icl4'(0) as 'icl4'(kCollectionAspect +
kDETAspectMainBitmap, purgeable);
include "AlbumCollectionIcons" 'icl8'(0) as 'icl8'(kCollectionAspect +
kDETAspectMainBitmap, purgeable);
include "AlbumCollectionIcons" 'ics#'(0) as 'ics#'(kCollectionAspect +
kDETAspectMainBitmap, purgeable);
include "AlbumCollectionIcons" 'ics4'(0) as 'ics4'(kCollectionAspect +
kDETAspectMainBitmap, purgeable);
include "AlbumCollectionIcons" 'ics8'(0) as 'ics8'(kCollectionAspect +
kDETAspectMainBitmap, purgeable);
include "AlbumCollectionIcons" 'SICN'(0) as 'SICN'(kCollectionAspect +
kDETAspectMainBitmap, purgeable);

// Supporting drops

resource 'rstr' (kCollectionAspect + kDETAspectDragInString, purgeable) {
   "Do you want to add %3%“^3”%the selected items% to *0x/the/* ^1 “^2”?"
};

resource 'rst#' (kCollectionAspect + kDETAspectRecordCatDragIn,purgeable)
```

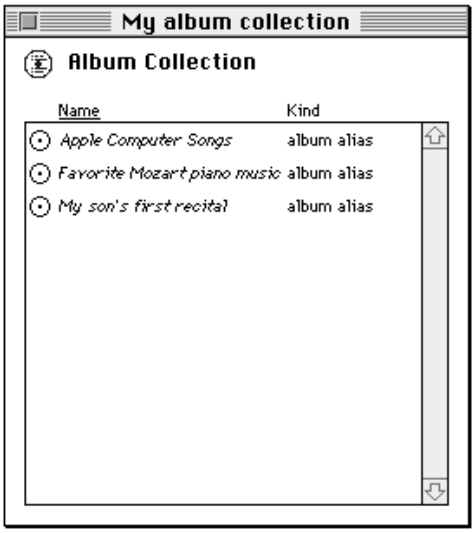
```
   {{
   "Recordings", kMemberAttrTypeBody
   }};

resource 'rst#' (kCollectionAspect + kDETAspectAttrDragIn,purgeable)
   {{
   "", kMemberAttrTypeBody, kMemberAttrTypeBody
   }};

resource 'dett' (kCollectionAspect + kDETAspectLookup, purgeable)
   {{
   {kMemberAttrTypeBody}, typePackedDSSpec,
      notForInput, notForOutput, useInSublist, isAlias, isNotRecordRef,
      {};
   }};

resource 'detm' (kCollectionAspect + kDETAspectViewMenu, purgeable)
   {
   kCollectionAspect + kDETAspectViewMenu,
   {
   kDETPrName, "by Name";
   kDETPrKind, "by Kind";
   }
   };


//-------------------------------------------------------------------------

// The information page template

#define k2ndColumnRightInset  (kDETRecordInfoWindWidth-kDETSubpageRightInset)


resource 'deti' (kCollectionInfoPage, purgeable) {
   1000,
   {sublistTopBound, sublistLeftBound, sublistBottomBound,
    sublistRightBound},
   noSelectFirstText,
   {
   kDETNoProperty, kDETNoProperty, kCollectionInfoPage;
   },
   {
   kDETNoProperty, kDETNoProperty, kCollectionInfoPage + 1;
```

```
   }};

resource 'rstr' (kCollectionInfoPage + kDETTemplateName, purgeable) {
   "WAVE Album Collection Info Page"
};

resource 'rstr' (kCollectionInfoPage + kDETRecordType, purgeable) {
   "WAVE Album Collection"
};

resource 'rstr' (kCollectionInfoPage + kDETInfoPageName, purgeable) {
   "Album Collection"
};

resource 'rstr' (kCollectionInfoPage + kDETInfoPageMainViewAspect, purgeable)
{
   "WAVE Album Collection Aspect"
};

resource 'detv' (kCollectionInfoPage, purgeable)
   {
   {
   {kPageBitmapTop, kPageBitmapLeft, kPageBitmapBottom, kPageBitmapRight},
      kDETNoFlags, kDETAspectMainBitmap,
      Bitmap { kDETSmallIcon };

   {sublistTopBound - 1, sublistLeftBound - 1, sublistBottomBound + 1,
      sublistRightBound + 1}, kDETNoFlags, kDETNoProperty,
      Box { kDETUnused };

   {sublistTitleTop, sublistLeftBound + nameColumnLeft, sublistTitleBottom,
    sublistLeftBound + nameColumnRight},
      kDETNoFlags, kDETPrName,
      StaticCommandTextFromView { kGeneva, 9, kDETLeft, kDETUnderline,
       "Name", kDETChangeViewCommand, - 1 };

   {sublistTitleTop, sublistLeftBound + kindColumnLeft, sublistTitleBottom,
    sublistLeftBound + kindColumnRight},
      kDETNoFlags, kDETPrKind,
      StaticCommandTextFromView { kGeneva, 9, kDETLeft, kDETNormal, "Kind",
       kDETChangeViewCommand, - 2 };
   };
   };
```

```
resource 'detv' (kCollectionInfoPage + 1, purgeable)
   {
   {
   {sublistIconTop, iconColumnLeft, sublistIconBottom, iconColumnRight},
      kDETHilightIfSelected, kDETAspectMainBitmap,
      Bitmap { kDETMiniIcon };

   {sublistTextTop, nameColumnLeft, sublistTextBottom, nameColumnRight},
      kDETHilightIfSelected + kDETDynamicSize, kDETPrName,
      StaticText { kGeneva, 9, kDETLeft, kDETIconStyle };

   {sublistTextTop, kindColumnLeft, sublistTextBottom, kindColumnRight},
      kDETNoFlags, kDETPrKind,
      StaticText { kGeneva, 9, kDETLeft, kDETNormal };
   }
   };
```

Figure 5-22 shows an example of the information page defined by Listing 5-6. Notice that this information page contains no Add or Remove buttons. The only way for a user to add a record alias to a record of type Album Collection is to drag an Album record into the sublist. The only way to remove one is to drag it from the sublist into the Trash. This design works well for the Album Collection record type because letting the user create a new, empty attribute for this record would make little sense. When the user double-clicks an album in the sublist, the Catalogs Extension opens the information page for the album, not for an attribute in the Album record.

Figure 5-22 Custom information page



Writing Template Code Resources

The set of templates you've seen so far creates information pages that let a user store information about an album, about each track on an album, and about a collection of albums. Because the user can enter the duration of each track in the Track Info attribute information page (Figure 5-5 on page 5-9), you can provide a code resource that automatically adds up the number of tracks and the total playing time so that the user does not have to enter that information into editable text boxes. The resulting information page (Figure 5-23) is identical to an earlier information page (Figure 5-21 on page 5-44) except that the number of tracks and total playing time are no longer editable text.

Figure 5-23 Information page using a code resource

The aspect and information page templates that create the information page in Figure 5-23 are identical to those in Listing 5-4 on page 5-44, with the following exceptions:

- The aspect template includes the code resource. To include the code shown in Listing 5-8 on page 5-68 (assuming this code has been compiled and saved as the resource `Album2Code` of type `'detc'` with a resource ID of 0), add the following line to the aspect template:

```
include "Album2Code" 'detc'(0) as
    'detc'(kAlbum2ndInfoPageAspect + kDETAspectCode, purgeable);
```

- The lookup table does not contain the attribute Album Track Info or elements for the properties `prNumTracks`, `prPlayingTimeHours`, `prPlayingTimeMinutes`, or `prPlayingTimeSeconds`. These properties are all handled by the code resource.
- Instead of the edit-text views in the view lists for the "Number of tracks" and "Playing time" fields, the view list contains static text fields that get the values to display from the code resource. Listing 5-7 shows the view lists.

Listing 5-7 View lists that get values from a code resource

```
resource 'detv' (kAlbum2ndInfoPage + 1, purgeable) {
   {
   {kNumTracksTop, kMyFirstColumnLeft, kNumTracksBottom,
    kMyFirstColumnRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont,
    kDETApplicationFontSize, kDETRight, kDETBold, "Number of
    tracks:" };

   {kNumTracksTop, k1stEditColumnLeft, kNumTracksBottom,
    k1stEditColumnLeft + kEditTextWidth},
   kDETNoFlags, prNumTracks,
   StaticText { kDETApplicationFont, kDETApplicationFontSize,
    kDETLeft, kDETNormal };
   }
};

resource 'detv' (kAlbum2ndInfoPage + 2, purgeable) {
   {
   {kPlayingTimeTop, kMyFirstColumnLeft, kPlayingTimeBottom,
    kMyFirstColumnRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont,
    kDETApplicationFontSize, kDETRight, kDETBold,
    "Total playing time:" };
```

```
   {kPlayingTimeTop, k1stEditColumnLeft, kPlayingTimeBottom,
    k1stEditColumnLeft + kEditTextWidth},
   kDETNoFlags, prPlayingTimeHours,
   StaticText { kDETApplicationFont, kDETApplicationFontSize,
    kDETLeft, kDETNormal };

   {kPlayingTimeTop, k1stEditColumnLeft + kSpaceBeforeEditDesc,
    kPlayingTimeBottom, k2ndEditColumnLeft},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont,
    kDETApplicationFontSize, kDETLeft, kDETNormal, "Hours" };

   {kPlayingTimeTop, k2ndEditColumnLeft, kPlayingTimeBottom,
    k2ndEditColumnLeft + kEditTextWidth},
   kDETNoFlags, prPlayingTimeMinutes,
   StaticText { kDETApplicationFont, kDETApplicationFontSize,
    kDETLeft, kDETNormal };

   {kPlayingTimeTop, k2ndEditColumnLeft + kSpaceBeforeEditDesc,
    kPlayingTimeBottom, k3rdEditColumnLeft},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont,
    kDETApplicationFontSize, kDETLeft, kDETNormal, "Minutes" };

   {kPlayingTimeTop, k3rdEditColumnLeft, kPlayingTimeBottom,
    k3rdEditColumnLeft + kEditTextWidth},
   kDETNoFlags, prPlayingTimeSeconds,
   StaticText { kDETApplicationFont, kDETApplicationFontSize,
    kDETLeft, kDETNormal };

   {kPlayingTimeTop, k3rdEditColumnLeft + kSpaceBeforeEditDesc,
    kPlayingTimeBottom, k4thEditColumnLeft},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont,
    kDETApplicationFontSize, kDETLeft, kDETNormal, "Seconds" };
   }
};
```

All of the routines you can provide in code resources for aspect templates are described in “Functions You Can Provide as Part of Your Code Resource” beginning on page 5-148. The Catalogs Extension can call the code resource for the aspect of the information page the user is currently using, or it can target another code resource. The code resource in Listing 5-8 handles only calls from the CE that are not targeted or for which the target is

kDETSelf. Targeting of code resource routines is described in "Target Specifier" on page 5-142.

At initialization, Listing 5-8 sets the call-for mask so that the CE calls this code resource only for idle events and view-change events. Thus, the CE calls this code resource periodically to let it process idle-time tasks. The CE also calls this code resource whenever the user opens the information page or displays a conditional view. The call-for mask is described in "Call-For Mask" on page 5-149.

Listing 5-8 calls routines provided by the CE—referred to in this chapter as *callback routines*—to get and set the values of properties and to obtain the number of items in the sublist. The CallBackDET macro that you can use to call these routines is described on "Calling CE-Provided Functions" on page 5-197. All of the available callback routines are described in "CE-Provided Functions That Your Code Resource Can Call" beginning on page 5-196.

Listing 5-8 calculates the playing time by adding up the playing times of all the individual tracks. To calculate this total, Listing 5-8 calls the kDETcmdGetPropertyNumber callback routine repeatedly, targeting each call to the attribute representing a specific track. See the description of the kDETSublistItem target selector in "Target Specifier" on page 5-142 to gain a better understanding of this technique.

IMPORTANT

When you design your code resource, you must follow certain rules to avoid corrupting or crashing the Finder. See "Rules for Writing Code Resources" on page 5-142 for details. s

Listing 5-8 Template code resource

```
/* Forward declaration of function defined later */

static OSErr DoIdle(DETCallBlockPtr callBlockPtr);

/* Dispatcher for routines in this code resource that the CE can call */

pascal OSErr MyAlbumCode(DETCallBlockPtr callBlockPtr)
   {
   OSErr err = kDETDidNotHandle;

   if ((callBlockPtr->protoCall.reqFunction < kDETcmdTargetedCall) ||
    (callBlockPtr->protoCall.target.selector == kDETSelf))
      {
      switch (callBlockPtr->protoCall.reqFunction)
         {
         case kDETcmdInit:
            callBlockPtr->init.newCallFors = kDETCallForIdle +
```

```
            kDETCallForViewChanges;
            break;

         case kDETcmdIdle:
         case kDETcmdViewListChanged:
            err = DoIdle(callBlockPtr);
            break;
         }
      }

   return err;
   }

/* This routine calls the CE-provided routine kDETcmdGetPropertyNumber to
    obtain the value of a property as a number. */

static OSErr DoGetPropertyNumber(DETCallBlockPtr callBlockPtr,
                                 DETTargetSelector selector,
                                 long itemNumber,
                                 short property,
                                 long *value)
   {
   OSErr err;
   DETCallBackBlock cbb;

   cbb.getPropertyNumber.reqFunction = kDETcmdGetPropertyNumber;
   cbb.getPropertyNumber.property = property;

   cbb.getPropertyNumber.target.selector = selector;
   cbb.getPropertyNumber.target.aspectName = nil;
   cbb.getPropertyNumber.target.itemNumber = itemNumber;

   err = CallBackDET(callBlockPtr, &cbb);

   *value = cbb.getPropertyNumber.propertyValue;

   return err;
   }

/* This routine calls the CE-provided routine kDETcmdSublistCount to
    obtain the number of items in a sublist. */
```

```
static OSErr DoGetNumSublistItems(DETCallBlockPtr callBlockPtr, long *num)
   {
   OSErr err;
   DETCallBackBlock cbb;

   cbb.sublistCount.reqFunction = kDETcmdSublistCount;
   cbb.sublistCount.target.selector = kDETSelf;

   err = CallBackDET(callBlockPtr, &cbb);

   *num = cbb.sublistCount.count;

   return err;
   }


/* This routine calls the CE-provided routine kDETcmdSetPropertyNumber to
    set the value of a number property. */

static OSErr DoSetPropertyNumber(DETCallBlockPtr callBlockPtr,
                                 short property,
                                 long newValue)
   {
   OSErr err;
   DETCallBackBlock cbb;

   cbb.setPropertyNumber.reqFunction = kDETcmdSetPropertyNumber;
   cbb.setPropertyNumber.property = property;
   cbb.setPropertyNumber.target.selector = kDETSelf;
   cbb.setPropertyNumber.newValue = newValue;

   err = CallBackDET(callBlockPtr, &cbb);

   return err;
   }

/* Here is the main routine for this code resource. This routine counts the
    number of tracks and adds up the total time for the album. */

static OSErr DoIdle(DETCallBlockPtr callBlockPtr)
   {
   OSErr err;
   long oldNumber, actualNumber;
```

```
/* Get the current value of the property prNumTracks. */
err = DoGetPropertyNumber(callBlockPtr, kDETSelf, 0, prNumTracks,
 &oldNumber);

/* Get the number of items in the sublist. Because each sublist item
    represents one track, set the value of prNumTracks equal to the
    number of sublist items. */
if (err == noErr)
   {
   err = DoGetNumSublistItems(callBlockPtr, &actualNumber);
   }

if ((err == noErr) && (oldNumber != actualNumber))
   {
   err = DoSetPropertyNumber(callBlockPtr, prNumTracks, actualNumber);
   }

if (err == noErr)
   {
   long index;
   long oldSeconds, actualSeconds = 0;
   long seconds;
   long minutes;
   long hours;

   /* Calculate the playing time by adding up the playing times of all
       the tracks, calling each track in the sublist in turn. */
   for (index = 1; (err == noErr) && (index <= actualNumber); ++index)
      {
      err = DoGetPropertyNumber(callBlockPtr, kDETSublistItem, index,
       prTrackSeconds, &seconds);

      if (err == noErr)
         {
         err = DoGetPropertyNumber(callBlockPtr, kDETSublistItem, index,
          prTrackMinutes, &minutes);
         }

      if (err == noErr)
         {
         actualSeconds += (minutes * 60 + seconds);
         }
```

```
   }

/* Get the old total playing time. */
if (err == noErr)
   {
   err = DoGetPropertyNumber(callBlockPtr, kDETSelf, 0,
    prPlayingTimeHours, &hours);
   }

if (err == noErr)
   {
   err = DoGetPropertyNumber(callBlockPtr, kDETSelf, 0,
    prPlayingTimeMinutes, &minutes);
   }

if (err == noErr)
   {
   err = DoGetPropertyNumber(callBlockPtr, kDETSelf, 0,
    prPlayingTimeSeconds, &seconds);
   }

/* Now compare the two playing times. If they're different, set
    the properties to equal the new one. */
if (err == noErr)
   {
   oldSeconds = 3600 * hours + 60 * minutes + seconds;

   if (oldSeconds != actualSeconds)
      {
      hours = actualSeconds / 3600;
      err = DoSetPropertyNumber(callBlockPtr, prPlayingTimeHours,
       hours);

      if (err == noErr)
         {
         actualSeconds -= (hours * 3600);
         minutes = actualSeconds / 60;
         err = DoSetPropertyNumber(callBlockPtr, prPlayingTimeMinutes,
          minutes);
         }

      if (err == noErr)
         {
```

```
            actualSeconds -= (minutes * 60);
            err = DoSetPropertyNumber(callBlockPtr, prPlayingTimeSeconds,
             actualSeconds);
            }
         }
      }
   }

return err;
}
```

AOCE Templates Reference

This section describes the file types, template types, and resource types you can use to extend the capabilities of the Catalogs Extension. For complete lists of the resource types required to create each type of AOCE template, see Table 5-1 on page 5-78 for aspect templates, Table 5-10 on page 5-120 for information page templates, Table 5-11 on page 5-138 for forwarder templates, Table 5-12 on page 5-140 for killer templates, and Table 5-13 on page 5-141 for file type templates.

Code resources are described in detail in "Code Resources Reference" beginning on page 5-142.

File and Resource Types Used by the Catalogs Extension

AOCE templates exist as resources in the resource forks of files in the Macintosh file system. A single file can contain multiple templates. The Catalogs Extension looks for files containing templates in the System Extensions folder inside the System Folder. It looks in all files in its list of appropriate file types. Initially, it uses the following file types:

File type	Description
`'detf'`	CE template file. This file type exists solely to hold templates.
`'dsam'`	A CSAM. If you create new templates to support a CSAM, you can make installation easy for users by including the templates in the CSAM file.
`'msam'`	An MSAM. If you create new templates to support an MSAM, you can make installation easy for users by including the templates in the MSAM file.
`'csam'`	A combined CSAM and MSAM. If you create new templates to support a combined CSAM and MSAM, you can make installation easy for users by including the templates in the SAM file.
`'fext'`	Finder extensions. The CE searches only for those Finder extension files that have creator `'adbk'`, which is the CE. This file supplies the templates that come with the CE and that are installed when the CE is installed.

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 "combined service access modules." Therefore a file of type `'dsam'` implements a CSAM and a file of type `'csam'` implements both a CSAM and an MSAM.

See the chapter "Service Access Module Setup" in *Inside Macintosh: AOCE Service Access Modules* for descriptions of the templates you must write to support CSAMs and MSAMs. u

File type templates can specify additional file types for the CE to search for template resources. You can use this feature to include AOCE templates with extension files of other types.

Within the file, templates consist of sets of associated resources. Each template in the file includes a signature resource, which specifies the type of the template and the base ID of the associated resources. All signature resources include a version number to tell the CE the format of the template. Because these version numbers are automatically included as a part of the Rez template resource definition, you do not need to include them explicitly in your Rez file. Some signature resources also contain additional template-related information.

AOCE templates use the following resource types for template signature resources:

Resource type	Kind of template
`'deta'`	Aspect template
`'deti'`	Information page template
`'detf'`	Forwarder template
`'detk'`	Killer template
`'detx'`	File type template

AOCE templates also use the following types of resources:

Resource type	Description
`'detb'`	Binary data
`'detc'`	Code resource
`'detm'`	Menu entries
`'detn'`	Number (long)
`'detp'`	Reverse-sort properties list
`'dett'`	Lookup table
`'detv'`	View list
`'detw'`	Custom information page window
`'rstr'`	String (`RString`)
`'rst#'`	`RString` array

In addition, icons are composed of a suite of resource types as described in the chapter "Finder Interface" in *Inside Macintosh, Macintosh Toolbox Essentials*.

IMPORTANT

It is very important not to overlap resource IDs from two different templates within the same file. One way to help ensure that they don't is to separate template base IDs by 250. s

Template Names

Every template includes a name resource, described in this section.

Each template must have a name so that it can be referred to by other templates. To avoid ambiguity in such cross-references, you should make your template names unique. To ensure uniqueness, you should start the template name with a four-character application signature registered with Macintosh Developer Technical Support.

kDETTemplateName

The template name resource has a resource ID with an offset of `kDETTemplateName` from the template's base (signature) resource ID.

```
resource 'rstr' (rMyBaseID + kDETTemplateName, purgeable) {
   "WAVE This is the name of my template"
};
```

The `'rstr'` resource is defined as follows:

```
type 'rstr' {
   rstring;       /* an RString */
};
```

To ensure the uniqueness of the template name, start it with a four-character application signature registered with Macintosh Developer Technical Support.

Specifying Record and Attribute Types for Templates

Each aspect and information page template applies only to specific types of records or attributes. To specify the types of records and attributes with which it is used, each template must include one or both of the record-type and attribute-type resources.

If your template applies to records, include a `kDETRecordType` resource but no `kDETAttributeType` resource. If your template applies to attributes in records of any type, include a `kDETAttributeType` resource but no `kDETRecordType` resource. Such

a template also supports stand-alone attributes. If your template applies to attributes in a specific type of record, include both a `kDETAttributeType` resource and a `kDETRecordType` resource. The following table summarizes these rules:

kDETRecordType	kDETAttributeType	Template applies to
Not present	Not present	Nothing (invalid)
`"my rectype"`	Not present	Records of type `"my rectype"`
Not present	`"my attrtype"`	Attributes of type `"my attrtype"` in records of any type or as stand-alone attributes
`"my rectype"`	`"my attrtype"`	Attributes of type `"my attrtype"` only in records of type `"my rectype"`

Note
Specifying both a `kDETAttributeType` resource and a `kDETRecordType` resource does not prevent a user from dragging an attribute from a sublist and dropping it on the desktop or on another object. To control which attribute types can be dragged from a sublist, use a `kDETAspectDragOut` resource (page 5-102). u

Note
A stand-alone attribute has a record type formed by concatenating the value of the constant `kAttributeValueRecTypeBody` (aoce Attribute Value), the attribute tag value, and the attribute type of the original attribute (without the attribute type's length or character set). The `RString` structure that holds the record type has a character set that is the same as the character set of the original attribute type. u

kDETRecordType

The record-type resource specifies the record type to which an aspect or information page template applies. The record-type resource has a resource ID with an offset of `kDETRecordType` from the template's base resource ID.

```
resource 'rstr' (rMyBaseResourceID + kDETRecordType, purgeable) {
"WAVE record type"
};
```

To ensure the uniqueness of the record type, start it with a four-character application signature registered with Macintosh Developer Technical Support.

kDETAttributeType

The attribute-type resource specifies the attribute type to which an aspect or information page template applies. The attribute-type resource has a resource ID with an offset of `kDETAttributeType` from the template's base resource ID.

```
resource 'rstr' (rMyBaseResourceID + kDETAttributeType,
                  purgeable) {
"WAVE attribute type"
};
```

To ensure the uniqueness of the attribute type, start it with a 4-character application signature registered with Macintosh Developer Technical Support.

kDETAttributeValueTag

The attribute-tag resource specifies the attribute value tag of the attributes to which an aspect or information page template applies. The attribute-tag resource has a resource ID with an offset of `kDETAttributeValueTag` from the template's base resource ID.

```
resource 'detn' (rMyTemplate + kDETAttributeValueTag, purgeable) {
   'bnry'
};
```

The `'detn'` resource type is defined as follows:

```
type 'detn' {
   longInt;
};
```

You can specify this resource only if the template also contains an attribute-type resource. If this resource is present, then the template applies only to attributes that have the specified tag. Because you can define your own attribute value tags, you can use this resource in any way you wish. If this resource is not present, then the template applies to attributes with any tag value.

The attribute-tag resource is useful to improve the efficiency of the display of addresses in a user address information page. Address attributes in user records should have attribute tags set to the value of the address subtype (for example, `'alan'` for LAN-based mail, `'aphn'` for Direct Dialup mail). Address templates should include the attribute-tag resource to specify the tag they work with. This resource allows the Catalogs Extension to determine very quickly the proper template to use with a given address.

Components of Aspect Templates

The primary purpose of an aspect template is to supply information about a record or attribute in an AOCE catalog. Most of this information is in the form of properties used by either an information page field or a sublist in an information page or dNode. Each information page template includes the name of an aspect used to supply properties for its fields and to generate its sublist (if any). In addition, an aspect template may provide information about how to create new items (records or attributes) of the type described by the template, the icon to use in a sublist, and the string to put in the "Kind" column of a sublist.

An aspect template can contain the resources listed in Table 5-1.

Table 5-1 Resources in aspect templates

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
`'deta'`	`0`	Identifies template as aspect and provides a base resource ID. Required for all aspect templates.
`'rstr'`	`kDETTemplateName`	Name of template. Required for all aspect templates.
`'rstr'`	`kDETRecordType`	Type of record to which the template applies. Either this resource, the `kDETAttributeType` resource, or both must be included.
`'rstr'`	`kDETAttributeType`	Type of attribute to which the template applies. Either this resource, the `kDETRecordType` resource, or both must be included.
`'detn'`	`kDETAttributeValueTag`	Attribute tag of attributes to which the template applies. You can provide this resource if you have also provided the `kDETAttributeType` resource. If you don't provide this resource, the template applies to attributes with any tag value.
icon suite	`kDETAspectMainBitmap`	Suite of icons. Neither the code resource nor the user can change these values. A set of icon resources at this offset is required for main aspect templates. (You may also include in any aspect template one or more icon suites with other resource IDs to provide icons for display in the information page.)

Table 5-1 Resources in aspect templates (continued)

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
'rstr'	kDETAspectKind	The kind of record or attribute as shown in a sublist. Neither the code resource nor the user can change this value. You must include this resource in main aspect templates; it is not needed in other aspect templates.
'rst#'	kDETAspectCategory	The names of categories to which the record or attribute belongs. These names are used internally by the CE. They are also displayed to the user if no template includes a corresponding kDETAspectExternalCategory resource. You must include this resource in main aspect templates for records and stand-alone attributes; it is not needed in other aspect templates.
'rst#'	kDETAspectExternalCategory	The names of categories to which the record or attribute belongs. These names are displayed to the user; they must correspond 1:1 to those in the kDETAspectCategory resource. If no template includes this resource, the CE displays the names in the kDETAspectCategory resource to the user. You can include this resource in main aspect templates; it is not needed in other aspect templates.
'detn'	kDETAspectGender	The gender of the kind to display in a sublist for objects of this type. For use with languages that require this information. You can include this resource in main aspect templates when necessary; it is not needed in other aspect templates.
'rstr'	kDETAspectWhatIs	Help-balloon string for objects of the type described by this aspect when they appear in a sublist. You must include this resource in main aspect templates; it is not needed in other aspect templates.
'rstr'	kDETAspectAliasKind	The kind of record or attribute to display in a sublist for aliases to the type of object described by this aspect. You must include this resource in main aspect templates; it is not needed in other aspect templates.

continued

Table 5-1 Resources in aspect templates (continued)

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
'detn'	kDETAspectAliasGender	The gender of the kind to display in a sublist for an alias to an object of this type. For use with languages that require this information. You can include this resource in main aspect templates when necessary; it is not needed in other aspect templates.
'rstr'	kDETAspectAliasWhatIs	Help-balloon string for aliases to objects of the type described by this aspect when the aliases appear in a sublist. You must include this resource in main aspect templates; it is not needed in other aspect templates.
'rstr'	kDETAspectNewMenuName	Text for the New item in the Catalogs menu for records, or for the Add button for the new-attribute-item dialog box. Include this resource only in a main aspect and only if the user is allowed to add a new record or attribute of this type. If you do not include this resource, the user cannot use the Catalogs menu or Add button to add a new object of this type.
'rstr'	kDETAspectNewEntryName	Name given to newly created records of the type described by this aspect. Include this resource only in a main aspect for a record and only if the user is allowed to add a new record of this type. If you do not include this resource, the user cannot use the Catalogs menu to add a new record of this type.
'rstr'	kDETAspectName	Name displayed in the sublist for newly created attributes of the type described by this aspect. Include only in a main aspect for an attribute, only if the user is allowed to add a new attribute of this type, and only if the kDETAspectNewValue resource does not provide a name for the new attribute.
'detb'	kDETAspectNewValue	The concatenation of the 4-byte attribute tag and the attribute value used as an initial value for newly created attributes of the type described by this aspect. Include this resource only in a main aspect for an attribute and only if the user is allowed to add a new attribute of this type. If you do not include this resource, the user cannot use the Add button to add a new attribute of this type.

Table 5-1 Resources in aspect templates (continued)

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
'detn'	kDETAspectSublistOpenOnNew	If you include this resource set to a nonzero number, the CE automatically opens newly created attributes or records of this type. This resource sets the value of the property that has property number kDETAspectSublistOpenOnNew. Your code resource can specify a different resource, overriding the resource in the aspect template (see "Dynamic Creation of Resources" beginning on page 5-154). You can also use the kDETcmdSetPropertyNumber callback routine (page 5-227) to change this property value at any time. You can include this resource only in main aspect templates. This resource is optional.
'detw'	kDETAspectInfoPageCustomWindow	The width, height, and placement of the set of custom information pages that appears if the user opens the catalog object to which this aspect applies. This resource also specifies whether a page-selection pop-up menu should be included in the window. You can include this resource only in main aspect templates. Include only if you do not want to use the default information page window.
'detv'	kDETAspectInfoPageCustomWindow	A view list that describes items to be displayed on every information page that appears if the user opens the catalog object to which this aspect applies. You can include this resource only in main aspect templates. Optional.
'rst#'	kDETAspectRecordDragIn	Types of records the user is allowed to drag and drop on the catalog object to which this aspect applies, paired with the attribute types used to store aliases to these records. When the user drops such a record, the CE creates an alias to the record and stores it in an attribute of the type you specify (unless your code resource takes some other action). Do not include this resource if you do not want to allow the user to drop records on this catalog object.

continued

Table 5-1 Resources in aspect templates (continued)

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
'rst#'	kDETAspectRecordCatDragIn	Categories of records that can be dropped on the catalog object to which this aspect applies, paired with attribute types used to store aliases to these records. When the user drops such a record, the CE creates an alias to the record and stores it in an attribute of the type you specify (unless your code resource takes some other action). Do not include this resource if you do not want to allow the user to drop records on this catalog object.
'rst#'	kDETAspectAttrDragIn	Record and attribute types of attributes the user is allowed to drag and drop on the catalog object to which this aspect applies. When the user drops such an attribute, the CE creates a copy of the attribute (unless your code resource takes some other action). Together with the record type and attribute type of each attribute the user can drop, the resource lists the attribute type the CE should assign to the new copy of the attribute. Do not include this resource if you do not want to allow the user to drop attributes on this catalog object.
'rstr'	kDETAspectDragInString	Prompt string that the CE displays when the user drags and drops an object on the catalog object to which this aspect applies. You do not have to include this resource if your aspect template does not support drops.
'rstr'	kDETAspectDragInVerb	Label for the OK button in the dialog box that the CE might display when the user drags and drops an object on the catalog object to which this aspect applies. You do not have to include this resource if your aspect template does not support drops.
'rstr'	kDETAspectDragInSummary	Short phrase that describes the action of dropping an object on the catalog object to which this aspect applies. The CE can use this phrase in a selection list if more than one aspect can receive the drop.

Table 5-1 Resources in aspect templates (continued)

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
'rst#'	kDETAspectDragOut	Types of attributes that the user is allowed to drag out of the aspect's sublist. If you do not provide this resource, the user can drag any attribute types from the sublist. To prevent the user from dragging any attributes from the sublist, include this resource but do not specify any attribute types.
'detn', 'rstr', 'detb'	Any property number in the range 0–249	If the aspect template has not used a lookup table or code resource to construct a property with this property number, the CE looks for a resource with an offset equal to this property number and uses the value in that resource as the value of the property. Your code resource can change the value of these resources before the CE loads them (see "Dynamic Creation of Resources" beginning on page 5-154). You can also use callback routines to change property values and types from a code resource at any time (see "Setting Value, Type, and Other Features of Properties" beginning on page 5-223).
'detm'	kDETAspectViewMenu	A list of property name and property number pairs that specifies the properties by which the user can sort items in a sublist. The property names you list in this resource appear in the View menu, and the property numbers tell the CE what to sort by. If the property number is positive, the sort is alphanumeric; if you specify the negative of the property number, the sort is numeric. You should provide this resource for any aspect template that includes a sublist.
'detp'	kDETAspectReverseSort	A list of properties that you want sorted in reverse order; that is, descending alphanumeric or ascending numeric order. You can list any property that you have already listed in a kDETAspectViewMenu resource.

continued

Table 5-1 Resources in aspect templates (continued)

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
'rst#'	kDETAspectBalloons	Help-balloon strings for views in an information page. You should provide strings for any properties you define, starting with property number kDETFirstDevProperty. For each such property, you include a string to be used if the property is editable and one that appears if the property is not editable. The strings appear in help balloons when Balloon Help is on and the user places the cursor on a view that has a help-balloon property number corresponding to an entry in this resource. If your aspect contains any such properties, you should include this resource.
'dett'	kDETAspectLookup	A lookup table that parses attributes into properties. Lookup tables are described in "The Lookup-Table Resource" beginning on page 5-105. You can include a lookup table in any aspect template.
'detc'	kDETAspectCode	A code resource. Code resources for aspect templates are described in "Code Resources Reference" beginning on page 5-142. You can include a code resource in any aspect template.

Properties

Properties are individual, self-contained pieces of information, such as a number or a string. Properties come from several sources: attribute values, resources in the template, constants, and data provided by the Catalogs Extension. Each property is identified by a property number. The property number uniquely identifies that property within an aspect. Note, however, that other aspects can have entirely unrelated properties with the same number; thus, it is important that each aspect have a unique name.

Each property also has a type. Table 5-2 lists the property types currently defined by Apple Computer, Inc.

Table 5-2 Property types

Property type	Use
`kDETPrTypeNumber`	A number, stored internally as a 32-bit unsigned long word
`kDETPrTypeString`	A string, stored as an `RString` structure
`kDETPrTypeBinary`	A binary block, stored as an uninterpreted sequence of bytes of any size

The CE converts between these types as necessary for purposes of storage or display. For example, a number stored in a 32-bit field in an attribute would be kept in a number-type property when read in to the aspect. However, the number would have to be displayed to the user as a character string. The CE does this conversion automatically.

You can define your own property types as needed. Apple Computer, Inc., reserves all property-type values less than or equal to 0. Therefore, if you define your own property type, give it a positive property-type value. Whenever the CE needs to convert to or from one of your private property types, it calls your code resource. The CE performs such conversions only between string and numeric types.

For example, you might want to store the date and time as a 32-bit long integer. Your code resource could convert between that format and a string. The date and time could then be stored in 32-bit format within the attribute value but displayed to the user using normal edit-text views in an information page.

Some properties are supplied as resources in the aspect template file, some properties are constructed from the contents of attribute values, and some are derived from the catalog system in other ways.

Properties are divided into three main categories:

- Local properties: Properties with property numbers from 0 through 249 are taken from the current (local) aspect. First, the CE checks properties constructed by parsing attribute values. (“The Lookup-Table Resource” beginning on page 5-105 describes how to use lookup tables to convert attribute values to properties.) If the CE finds a constructed property with the appropriate number, it uses that property. If not, then the CE looks for a template resource of type `'detn'` (number), `'rstr'` (type `RString`), or `'detb'` (binary) with a resource ID equal to the template base ID plus the property number. You should use property numbers starting with the value `kDETFirstDevProperty`. For example, property `rMyProperty` of type `RString` might be defined as follows:

```
#define rMyProperty kDETFirstDevProperty + 10

resource 'rstr' (rMyAspectResourceID + rMyProperty, purgeable)
{
"My fixed property value."
};
```

- Constants: information page templates can use property numbers in the range 250–499 as a shortcut to defining numeric constants in the range 0–249. To specify the constant *n* ($0 \le n \le 249$), use property number *n* + 250.
- Metaproperties: These properties are generated by the CE itself or are retrieved from an AOCE catalog**,** but they are not attribute values. Examples of these are the name or type of a record, or the value of a record or attribute access mask. Table 5-3 lists the kinds of metaproperties that are available. (Note that a "metaproperty name" is a symbolic name for a reserved property number.)

Several of the metaproperty descriptions in Table 5-3 mention property commands. A **property command** is any command handled by your code resource's `kDETcmdPropertyCommand` routine (page 5-159). The CE calls your code resource with the `kDETcmdPropertyCommand` routine selector whenever 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, as described in Table 5-14 starting on page 5-161. Each property command includes a property number, usually the number of the property that corresponds to the information page control that originated the command.

Table 5-3 Metaproperties

Metaproperty name	Use
`kDETPrName`	The name of an attribute, record, or alias. If you specify this property in a view list, for example, the CE displays the name of the record, the name of the alias (which it gets from the record pointed to by the alias), or the name of the attribute (which it gets from the `kDETAspectName` property). You cannot store a name in this metaproperty. To store the name of an attribute, use the `kDETAspectName` property in a lookup table or resource.
`kDETPrKind`	The kind of a record or alias.
`kDETPastFirstLookup`	The value of this property is 0 until the CE has completed its first catalog lookup, after which it's 1.
`kDETInfoPageNumber`	The number of the information page currently being displayed. This number is 0 if no information page is open. You can have the CE display a different information page by issuing a property command (such as a command sent by a pop-up menu) with this property number and with the new page number in the `parameter` field. You must use this metaproperty to implement your own page-selection pop-up menu in place of the default pop-up menu.
`kDETAspectTemplateNumber`	The template number of the targeted aspect's template. This value can be used with the `kDETAspectTemplate` target selector (see "Target Specifier" on page 5-142).

Table 5-3 Metaproperties (continued)

Metaproperty name	Use
`kDETInfoPageTemplateNumber`	The template number of the template for the currently open information page (if any). This value can be used with the `kDETInfoPageTemplate` target selector (see “Target Specifier” on page 5-142).
`kDETOpenSelectedItems`	A property number for use with the property command associated with an Open button in an information page with a sublist. When you issue a property command with this property number, the CE opens the sublist items the user has selected.
`kDETAddNewItem`	A property number for use with the property command associated with an Add button in an information page with a sublist. When you issue a property command with this property number, the CE displays a dialog box that lets the user add an attribute to the sublist.
`kDETRemoveSelectedItems`	A property number for use with the property command associated with a Remove button in an information page with a sublist. When you issue a property command with this property number, the CE removes the sublist items the user has selected.
`kDETAspectSublistOpenOnNew`	If you set this property to a nonzero number, the CE automatically opens newly created sublist entries. You can also set a default value for this property by including a `'detn'` resource at this offset in the aspect template.
`kDETDNodeAccessMask`	The dNode access mask.
`kDETRecordAccessMask`	The record access mask.
`kDETAttributeAccessMask`	The attribute access mask.
`kDETPrimaryMaskByBit`	The full set of 16 bits of the primary access mask (the access mask for the object you are in); the following properties provide values for several of the individual bits.
`kDETPrimarySeeMask`	The “see” access mask bit (1 for “can see”)
`kDETPrimaryAddMask`	The “add” access mask bit (1 for “can add”)
`kDETPrimaryDeleteMask`	The “delete” access mask bit (1 for “can delete”)
`kDETPrimaryChangeMask`	The “change” access mask bit (1 for “can change”)
`kDETPrimaryRenameMask`	The “rename” access mask bit (1 for “can rename”)
`kDETPrimaryChangePrivsMask`	The “change privileges” access mask bit (1 for “can change privileges”)

NOTE Access masks are described in the chapter “Catalog Manager” in this book.

Aspect Template Signature Resource

As with all templates, an aspect template has a signature resource that identifies the type of template and that provides a base resource ID relative to which all other resource IDs are numbered.

'deta' Resource

The signature resource for an aspect template is of type `'deta'`.

```
type 'deta' {
   longInt = kDETAspectVersion;                /* template format version */
   longInt;                                    /* drop-operation order */
   boolean dropCheckConflicts, dropCheckAlways; /* confirm drops? */
   boolean notMainAspect, isMainAspect;        /* main aspect? */
   align word;                                 /* reserved */
};
```

The Catalogs Extension uses the drop-operation order in case a catalog object is associated with more than one aspect that can handle drop operations. The operation specified by the aspect with the lowest drop-operation order is offered to the user. If two templates have the same drop-operation order number, the CE picks one arbitrarily. If your template does not support drop operations, the CE ignores this field.

If you specify `dropCheckAlways` as the drop-check Boolean value, the CE always displays a dialog box asking the user to confirm a drop operation before performing it. If you specify `dropCheckConflicts` as this Boolean value, the CE displays the dialog box only if there is more than one aspect that supports drops for this catalog object. You must provide a prompt string and button label for the dialog box in additional resources in the aspect template. If your template does not support drop operations, the CE ignores this field.

See “Drags and Drops” on page 5-28 for more information about drag-and-drop operations. See “Supporting Drags and Drops” beginning on page 5-98 for descriptions of all the resources you must provide to support these operations.

You use the second Boolean field of the aspect signature resource to specify whether this template is for a main aspect. The next section describes additional resources you must provide if this is a main aspect template.

Main Aspect Template Resources

Main aspects provide the information the Catalogs Extension needs to display the record or attribute value in a sublist—such as a record in a dNode window or an attribute value in a record window. Figure 5-21 on page 5-44 shows an information page with a sublist. Main aspect templates provide the information the CE needs to create a main aspect. You

can also use a main aspect template to specify the size, location, and standard contents of custom information pages that the CE displays when the user opens an object to which the main aspect template applies.

Table 5-4 lists the resources used specifically by main aspect templates. In addition to the resources in Table 5-4, main aspect templates can include any of the resources shown in Table 5-1 on page 5-78. The resources required by main aspect templates are described in this section. The resources required for all aspect templates are described in the preceding sections. All aspect-template resource descriptions are summarized in Table 5-1 on page 5-78.

Note that main aspects can support information pages in addition to supporting a sublist. For example, a single main aspect can provide the information needed to list a specific attribute value in a sublist, plus all the properties needed by the information page displayed when the user double-clicks that line on the sublist.

Table 5-4 Resources used by main aspect templates

Resource type	Offset of resource ID from signature resource ID	Comments
`'deta'`	`0`	Required for all aspect templates.
`'rstr'`	`kDETTemplateName`	Required for all aspect templates.
`'rstr'`	`kDETRecordType`	Either this resource, the `kDETAttributeType` resource, or both must be included.
`'rstr'`	`kDETAttributeType`	Either this resource, the `kDETRecordType` resource, or both must be included.
icon suite	`kDETAspectMainBitmap`	Required for all main aspect templates.
`'rstr'`	`kDETAspectKind`	Required for all main aspect templates.
`'rst#'`	`kDETAspectCategory`	Required for records and stand-alone attributes.
`'rst#'`	`kDETAspectExternalCategory`	Optional.
`'detn'`	`kDETAspectGender`	Optional.
`'rstr'`	`kDETAspectWhatIs`	Required for all main aspect templates.
`'rstr'`	`kDETAspectAliasKind`	Required for all main aspect templates.
`'detn'`	`kDETAspectAliasGender`	Optional.
`'rstr'`	`kDETAspectAliasWhatIs`	Required for all main aspect templates.
`'rstr'`	`kDETAspectNewMenuName`	Include if the user is allowed to add a new record or attribute of this type.

continued

Table 5-4 Resources used by main aspect templates (continued)

Resource type	Offset of resource ID from signature resource ID	Comments
'rstr'	kDETAspectNewEntryName	Include in a template for a record if the user is allowed to add a new record of this type.
'rstr'	kDETAspectName	Include in a template for an attribute if the user is allowed to add a new attribute of this type, and if the kDETAspectNewValue resource does not provide a name for the new attribute.
'detb'	kDETAspectNewValue	Include in a template for an attribute if the user is allowed to add a new attribute of this type.
'detn'	kDETAspectSublistOpenOnNew	Optional.
'detw'	kDETAspectInfoPageCustomWindow	Include only if you do not want to use the default information page window.
'detv'	kDETAspectInfoPageCustomWindow	Optional.

kDETAspectMainBitmap

A main aspect 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 AlbumIcons, that all of your icon resources had resource IDs of 0, and that your resource base ID was kMainAspect. In that case, you could use the following code to include the icon suite in your main aspect template:

```
include "AlbumIcons" 'ICN#'(0) as
                  'ICN#'(kMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'icl4'(0) as
                  'icl4'(kMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'icl8'(0) as
                  'icl8'(kMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'ics#'(0) as
                  'ics#'(kMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'ics4'(0) as
                  'ics4'(kMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'ics8'(0) as
                  'ics8'(kMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'SICN'(0) as
                  'SICN'(kMainAspect + kDETAspectMainBitmap, purgeable);
```

The icon suite must be included in the main aspect template or specified by your `kDETcmdDynamicResource` code resource routine (page 5-156). Once the icon suite has been specified, it cannot be changed from a code resource or by the user.

kDETAspectKind

Specify the kind of the record or attribute as it is to be displayed in a sublist with an `RString` resource with an offset of `kDETAspectKind` from the template's base resource ID.

```
resource 'rstr' (kMainAspect + kDETAspectKind, purgeable)
{
   "My Kind"
};
```

This resource must be included in the main aspect template or specified by your `kDETcmdDynamicResource` code resource routine (page 5-156). Once the record or attribute kind has been specified, it cannot be changed from a code resource or by the user.

kDETAspectCategory

Specify the categories to which the record or attribute belongs with an `RString`-array resource with an offset of `kDETAspectCategory` from the template's base resource ID.

```
resource 'rst#' (kMainAspect + kDETAspectCategory, purgeable)
{
   { "Category 1", "Category 2" }
};
```

The `'rst#'` resource type is defined as follows:

```
type 'rst#' {
   integer = (endOfData - startOfData) / 8;
startOfData:
   integer = $$CountOf(RStrArray);           /* Array size */
   array RStrArray {
      rstring;
      align word;
   };
endOfData:
};
```

Every record must be assigned to one or more categories. The Catalogs Extension uses record categories to group records in catalog windows. You also specify record categories if you include a `kDETAspectRecordCatDragIn` resource in your template (see page 5-99). If no template includes a `kDETAspectExternalCategory` resource, the CE uses the category name you provide in the `kDETAspectCategory` resource for display in the View menu.

You must include a `kDETAspectCategory` resource in an attribute's main aspect template if the template supports a stand-alone attribute—that is, if you do not include a `kDETRecordType` resource in the main aspect template.

Note that an item in a sublist can be of only one kind, but it can be in as many categories as is appropriate. Thus, the kind resource is a single string of type `RString`, but the category resource is an array of `RString` strings.

kDETAspectExternalCategory

Specify category names to be displayed to the user with an `RString`-array resource with an offset of `kDETAspectExternalCategory` from the template's base resource ID.

```
resource 'rst#' (kMainAspect + kDETAspectExternalCategory, purgeable)
{
   { "Category 1", "Category 2" }
};
```

This resource must contain one category name for each name in the `kDETAspectCategory` resource, in the same sequence. If you do not include this resource in the main aspect template, the Catalogs Extension uses the external category names provided for these categories in any other template available. If no template provides this resource, the CE uses the names in the `kDETAspectCategory` resource. (If more than one template provides a `kDETAspectExternalCategory` resource, the CE picks one of them and ignores the others.)

You should use category names in your local language for the `kDETAspectCategory` resource when you are creating the template. Then, when someone localizes the template to another language, the person doing the localizing can add a `kDETAspectExternalCategory` template to change the names displayed to the user.

kDETAspectGender

You can specify the gender of the record kind of the record to which this main aspect applies with a resource of type `'detn'` with an offset of `kDETAspectGender` from the template's base resource ID.

```
resource 'detn' (kMainAspect + kDETAspectGender, purgeable) {
   1
};
```

You can use this resource to match the article of the record kind to the gender of the record kind when you are displaying the kind to the user. The significance of the value in this resource depends on the language with which it is being used; you can place any value you wish in this resource, and you are responsible for interpreting it. It is recommended that you follow the guidelines in *Guide to Macintosh Software Localization*.

kDETAspectWhatIs

Each main aspect template must provide a help-balloon string. The Finder displays this string when the user enables Balloon Help online assistance and moves the cursor over an object (a record or attribute) of the type 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' (kMainAspect + kDETAspectWhatIs, purgeable) {
   "Flue handle record\n\nA record containing information about a flue
handle."
};
```

kDETAspectAliasKind

To specify the kind of an alias to a record or attribute as it is to be displayed in a sublist, use an `RString` resource with an offset of `kDETAspectAliasKind` from the template's base resource ID.

```
resource 'rstr' (kMainAspect + kDETAspectAliasKind, purgeable)
{
   "My kind alias"
};
```

kDETAspectAliasGender

To specify the gender of the kind of the alias to a record, use a resource of type `'detn'` with an offset of `kDETAspectAliasGender` from the template's base resource ID.

```
resource 'detn' (kMainAspect + kDETAspectAliasGender, purgeable) {
   1
};
```

The significance of the value in this resource depends on the language with which it is being used; you can place any value you wish in this resource, and you are responsible for interpreting it.

kDETAspectAliasWhatIs

Each main aspect template must provide a help-balloon string for aliases. The Finder displays this string when the user enables Balloon Help online assistance and moves the cursor over an alias to an object (a record or attribute) of the type to which this main aspect applies. The help-balloon string is in an `RString` resource with an offset `kDETAspectAliasWhatIs` from the template's base resource ID.

```
resource 'rstr' (kMainAspect + kDETAspectAliasWhatIs, purgeable) {
   "Alias to a flue handle record\n\nAn alias to a record containing
information about a flue handle."
};
```

kDETAspectNewMenuName

A main aspect template for a record type specifies how new records of that type are to be added to the containing dNode. Similarly, a main aspect template for an attribute type specifies how new attributes of that type are to be added to the containing record. To allow the user to add a new record or attribute, you must supply an `RString` resource containing the text of the New item for the Catalogs menu or the Add item for the new-attribute dialog box. The menu name resource must have an ID with the offset `kDETAspectNewMenuName` from the template's base resource ID.

```
resource 'rstr' (rMyAspectResourceID + kDETAspectNewMenuName, purgeable)
{
   "The text for the New menu item goes here"
};
```

IMPORTANT

To allow the user to add a new attribute, you must provide a New button in your information page, and the property command associated with that button must use the property number `kDETAddNewItem`. See Table 5-3 on page 5-86 for a description of this and other special property numbers. s

kDETAspectNewEntryName

A main aspect template for a record type must also specify the name the Catalogs Extension should give to newly created records of that type. To provide this name, use an `RString` resource with an offset of `kDETAspectNewEntryName` from the template's base resource ID.

```
resource 'rstr' (rMyAspectResourceID + kDETAspectNewEntryName, purgeable)
{
   "Name of new record goes here"
};
```

kDETAspectName

A main aspect template for an attribute type can specify a default name for newly created attribute values of that type. To provide this name, use an `RString` resource with an offset of `kDETAspectName` from the template's base resource ID.

```
resource 'rstr' (rMyAspectResourceID + kDETAspectName, purgeable)
{
   "Name of new attribute goes here"
};
```

For main aspect templates for records, the Catalogs Extension automatically sets the `kDETAspectName` property to be the name of the record. This property provides the name the CE displays in the "Name" column in dNode windows.

If you wish to specify a name for an attribute value, you can provide a value for the `kDETAspectName` property in the main aspect template for the attribute. You can display attribute value names in sublists (use the metaproperty `kDETPrName` for this purpose); the CE uses attribute value names for stand-alone attributes.

You can store the name of an attribute in the `kDETAspectName` property at any time. You can use the lookup table, a code resource, a `kDETAspectName` resource, or any combination of these methods to provide a value for the `kDETAspectName` property.

You should limit sublist items to one line. Multiline sublist items are not guaranteed to work correctly.

Note
For records to be displayed in the Key Chain, your setup main aspect template must provide a `kDETAspectName` resource to explicitly set this property. See the chapter "Service Access Module Setup" in *Inside Macintosh: Service Access Modules* for complete information about setup templates. u

kDETAspectNewValue

If the new item is an attribute value, the main aspect template must contain a binary block resource (type `'detb'`) containing the concatenation of the attribute tag and the new value. Give this resource an ID with an offset of `kDETAspectNewValue` from the template's base resource ID.

```
data 'detb' (kTrackAspect + kDETAspectNewValue, purgeable) {
   $"626E 7279"                          // tag (bnry)
   $"0000 0001"                          // prTrackNumber (1)
   $"0000 0000"                          // prTrackMinutes (1)
   $"0000 0000"                          // prTrackSeconds (1)
   $"0000 0007 3C74 6974 6C65 3E"        // kDETAspectName (<title>)
   $"0000 000A 3C63 6F6D 706F 7365 723E" // composer (<composer>)
   $"0000 000A 3C63 6F6D 6D65 6E74 733E" // comments (<comments>)
```

The Catalogs Extension needs the attribute tag, the attribute value, and the attribute type to add an attribute to a record in an AOCE catalog. When the user clicks the Add button in your information page to create a new attribute value, the CE gets the attribute type from the aspect template's attribute-type resource (see "Specifying Record and Attribute Types for Templates" on page 5-75) and the tag and initial attribute value from the `kDETAspectNewValue` resource. Once the CE has added the attribute to the catalog record, the CE uses the main aspect for attributes of that type to display that attribute in a sublist.

kDETAspectSublistOpenOnNew

A main aspect template can specify whether the Catalogs Extension should automatically open an information page for a newly created attribute or record in a sublist. If you set the value of property `kDETAspectSublistOpenOnNew` to a nonzero number, the CE automatically opens newly created attributes or records of the type associated with the main aspect. You can provide a `'detn'` resource with an offset of `kDETAspectSublistOpenOnNew` to provide a default value for this property.

```
resource 'detn' (kMainAspect + kDETAspectSublistOpenOnNew, purgeable) {
   1
};
```

Your code resource can change the value of this resource before the CE loads it (see "Dynamic Creation of Resources" beginning on page 5-154). You can also use the `kDETcmdSetPropertyNumber` callback routine (page 5-227) to change this property value from a code resource at any time. This resource is optional.

kDETAspectInfoPageCustomWindow

If you want to specify a nonstandard size or location for the information page window that appears when the user opens the catalog object to which this main aspect applies, you can provide a `'detw'` resource with an offset of `kDETAspectInfoPageCustomWindow` from the template's base resource ID. This resource also specifies whether a page-selection pop-up menu should be included in the window.

```
resource 'detw' (kMainAspect + kDETAspectInfoPageCustomWindow, purgeable) {
   {kCustomPageTop, kCustomPageLeft, kCustomPageBottom, kCustomPageRight},
   discludePopup
};
```

The `'detw'` resource type is defined as follows:

```
type 'detw' {
   rect;                                /* info-page window in  */
                                        /*  global coordinates */
   boolean discludePopup, includePopup;/* include a page-selection pop-up? */
   align word;                          /* Future expansion */
};
```

Note that you can specify no pop-up menu only if there is only one information page for the object or if you are also providing a `'detv'` resource in this main aspect template that provides a pop-up menu of information page names.

If there is more than one information page for the object and you include this resource in the main aspect template, then all of the information pages have the size you specify in this resource.

IMPORTANT

If the information page window you specify is too large to be displayed on the user's screen, the Finder makes the window smaller, truncating the bottom and right side of the information page. To be sure an information page can fit on a Macintosh computer screen of any size, make it no larger than 322 pixels high by 512 pixels wide. s

There are two special values you can use for the top left corner of a custom information page window:

- specify (0, 0) to place the upper-left corner of the window slightly below and to the right of the upper-left corner of the parent window
- specify (–1, –1) to center the window on the screen

If you want to specify a view list for views that are to be displayed on all the information pages that appear when the user opens the catalog object to which this main aspect applies, you can provide a `'detv'` resource with an offset of `kDETAspectInfoPageCustomWindow` from the template's base resource ID.

If, as part of this view list, you include a pop-up menu (view type `Menu`) with a property number of `kDETInfoPageNumber`, the Catalogs Extension displays a pop-up menu with a list of information page names at the location you specify. Therefore, by combining a `'detw'` resource with the `discludePopup` flag set and a `'detv'` resource with a pop-up menu for selecting information pages, you can create a custom set of information pages with the pop-up menu at any location you wish. The following view list displays an icon and a pop-up menu for selecting information pages.

```
resource 'detv' (kMainAspect + kDETAspectInfoPageCustomWindow, purgeable) {
   {
   {kBitmapTop, kBitmapLeft, kBitmapBottom, kBitmapRight},
   kDETNoFlags, kDETAspectMainBitmap,
   Bitmap { kDETLargeIcon };

   {kMenuTop, kMenuLeft, kMenuBottom, kMenuRight},
   kDETPopupDynamicSize, kDETInfoPageNumber,
   Menu { kDETApplicationFont, kDETApplicationFontSize,
                  kDETLeft, kDETNormal, "Page", kDETInfoPageNumber,
                  rMenuResourceID };
   }
};
```

View lists are described in "View Lists" beginning on page 5-123.

Supporting Drags and Drops

If your aspect supports drags and drops, your template should include the resources listed in this section, a code resource that handles drags and drops, or both. Drags and drops are described in "Drags and Drops" on page 5-28. For more information on how code resources handle drags and drops, see the descriptions of the `kDETcmdDropQuery` (page 5-172) and `kDETcmdDropMeQuery` (page 5-170) routines.

kDETAspectRecordDragIn

When a user drags a record and drops it on another record, the most common result is that the Catalogs Extension creates an alias to the record and stores it in an attribute in the target record. When the user drops a record on a record for which you provided an aspect template, the CE looks for a resource with an ID offset of `kDETAspectRecordDragIn` from the base resource ID. This resource lists the types of records that can be dragged into the aspect, paired with the attribute type in which the CE should store an alias to the record that was dragged. Use the `kDETAspectRecordCatDragIn` resource (described next) to specify categories of records that can be dragged and dropped on the catalog object.

```
resource 'rst#' (kMyAspectTemplate + kDETAspectRecordDragIn, purgeable)
{
    "Record type 1", "Alias attribute type 1",
    "Record type 2", "Alias attribute type 2"
};
```

Do not include a `kDETAspectRecordDragIn` resource if you do not want to allow the user to drop records on this catalog object.

Note
In addition to checking for drag-in resources, the CE calls your code resource (if any) to check whether the code resource can handle the drop. A code resource can take an action different from that specified in the drag-in resources, can handle a drop in the absence of drag-in resources, and can handle drags and drops for source and destination objects other than records (as long as either the source or the destination is an AOCE catalog object). u

See the discussion of drags and drops in "Drags and Drops" on page 5-28 and the descriptions of the `kDETcmdDropQuery` (page 5-172) and `kDETcmdDropMeQuery` (page 5-170) routines for more information on this process.

IMPORTANT
If your code resource does not take a different action and you do want the CE to store an alias to the record in an attribute value, you must be sure that you have listed the attribute type in your aspect's lookup table with the `useInSublist` and `isAlias` flags set. Lookup tables are described in "The Lookup-Table Resource" beginning on page 5-105. s

kDETAspectRecordCatDragIn

Instead of or in addition to specifying individual record types of records that can be dragged and dropped on the record to which the aspect applies, you can specify categories of records that can be dropped. When the user drops such a record, the

Catalogs Extension creates an alias to the record and stores it in an attribute (unless your code resource takes some other action). For each category of record the user can drop, the resource also lists the attribute type of the new attribute containing the alias to the record. Do not include this resource if you do not want to allow the user to drop records on this catalog object.

```
resource 'rst#' (kMyAspectTemplate + kDETAspectRecordCatDragIn, purgeable)
{
    "Category 1", "Alias attribute type 1",
    "Category 2", "Alias attribute type 2"
};
```

See the preceding discussion of the `kDETAspectRecordDragIn` resource for more information on drags and drops.

kDETAspectAttrDragIn

When a user drags an attribute and drops it on a record, the most common result is that the Catalogs Extension creates a copy of the attribute and stores it in the target record. When the user drops an attribute on a record for which you provided an aspect template, the CE looks for a resource with an ID offset of `kDETAspectAttrDragIn` from the base resource ID. This resource lists the types of records that can be dragged from, the types of attributes that can be dragged into the target record, and the attribute type the CE should assign to the new copy of the attribute. To indicate any type, you can use the empty string "" for the type of record that can be dragged from.

```
resource 'rst#' (kMyAspectTemplate + kDETAspectAttrDragIn, purgeable)
{
    "Record type 1", "Attribute type 1", "New attribute type 1",
    "Record type 2", "Attribute type 2", "New attribute type 2"
};
```

Do not include this resource if you do not want to allow the user to drop attribute values on this catalog object.

Note
In addition to checking for drag-in resources, the CE calls your code resource (if any) to check whether the code resource can handle the drop. A code resource can take an action different from that specified in the drag-in resources, can handle a drop in the absence of drag-in resources, and can handle drags and drops for source objects other than attributes or records and destination objects other than records (as long as either the source or the destination is an AOCE catalog object). u

See the discussion of drags and drops in "Drags and Drops" on page 5-28 and the descriptions of the `kDETcmdDropQuery` (page 5-172) and `kDETcmdDropMeQuery` (page 5-170) routines for more information on this process.

kDETAspectDragInString

If your aspect supports drops, you must provide a prompt string for the Catalogs Extension to display when the user drags and drops an object on the catalog object to which the aspect applies. You provide this string in an `RString` resource with an ID offset of `kDETAspectDragInString` from the base resource ID.

```
resource 'rstr' (rMyAspectTemplate + kDETAspectDragInString, purgeable)
{
   "Do you want to send %3%"^3"%the selected items% to *0x/the/* ^1 "^2"?"
};
```

The string `%3%"^3"%the selected items%` helps the CE either insert the name of the item being dragged (in `^3` if it's a single selection) or insert the substring "`the selected items`" (if it's a multiple-item selection).

The token `^1` is the destination's kind as displayed in the sublist (`kDETAspectKind`).

The token `^2` is the destination's name.

When localizing this resource, replace the string `*0x/the/*` with an article consistent with the destination's kind (token `^1`).

You do not have to include this resource if your aspect template does not support drops.

kDETAspectDragInVerb

If your aspect supports drops, you must provide a label for the OK button in the dialog box that the Catalogs Extension can display when the user drags and drops an object on the catalog object to which the aspect applies. You provide this label in an `RString` resource with an ID offset of `kDETAspectDragInVerb` from the base resource ID.

```
resource 'rstr' (rMyAspectTemplate + kDETAspectDragInVerb, purgeable)
{
   "Send"
};
```

If you are unsure as to what label to use for this feature, use OK.

You do not have to include this resource if your aspect template does not support drops.

Drag-in verbs are not implemented in the initial release of the AOCE software, but you should include this resource to support future enhancements to the software.

kDETAspectDragInSummary

If your aspect supports drops, you should provide a short phrase that describes the action of dropping an object on the catalog object to which this aspect applies. The Catalogs Extension can use this phrase in a selection list if more than one aspect can receive the drop.

```
resource 'rstr' (rMyAspectTemplate + kDETAspectDragInSummary, purgeable)
{
   "Send item"
};
```

You do not have to include this resource if your aspect template does not support drops.

Selection lists are not implemented in the initial release of the AOCE software, but you should include this resource to support future enhancements to the software.

kDETAspectDragOut

If your aspect template's lookup table includes any attribute types for which you have set the `useInSublist` flag, the user might try to drag an attribute of that type from the sublist to drop it on another object or on the desktop. You can include a resource of type `'rst#'` that specifies which attribute types the user is allowed to drag from the sublist. You must give this resource an ID offset of `kDETAspectDragOut` from the base resource ID.

```
resource 'rst#' (rMyAspectTemplate + kDETAspectDragOut, purgeable)
{
   {
   "First attribute type",
   "Second attribute type"
   }
};
```

If you do not provide this resource, the user can drag any attribute types from the sublist. To prevent the user from dragging any attributes from the sublist, include this resource but do not specify any attribute types.

Other Aspect Template Resources

Any aspect template can include the resources listed in this section. In addition to the resources described here, see Table 5-1 on page 5-78 for a summary of all of the resources that you can include in an aspect template.

Any property number in the range 0–249

If the information page template has not used a lookup table or code resource to construct a property for any property number in the range 0–249, the Catalogs Extension looks for a resource whose ID has an offset from the base resource ID equal to that property number and uses the value in that resource as the value of the property. You can use resources of types `'detn'`, `'rstr'`, or `'detb'` for this purpose.

You must be careful to ensure that none of your property numbers conflict with the resource ID numbers defined in the AOCE header files. To do so, use the value `kDETFirstDevProperty` for your first property number and increment each additional property number by 1.

kDETAspectViewMenu

The user can sort a sublist in an information page by any of a number of different properties, just as users can sort lists in the Finder by name, kind, date, and so forth. The user can use the View menu to select the property to use for sorting. If your aspect template supports a sublist (that is, if the lookup table includes any attribute types for which you have set the `useInSublist` flag), you should provide a resource of type `'detm'` to specify the properties that can be used for sorting. You must give this resource an ID offset of `kDETAspectViewMenu` from the base resource ID.

```
resource 'detm' (rMyAspectTemplate + kDETAspectViewMenu, purgeable)
{
   rMyAspectResourceID + kDETAspectViewMenu,
   {
   kPrName, "By Name";
   -kPrAge, "By Age";
   }
};
```

The 'detm' resource type is defined as follows:

```
type 'detm' as 'fmnu';
```

For each property by which the sublist may be sorted, you must provide the property number followed by the text that appears in the View menu.

Normally, the Catalogs Extension sorts items in ascending alphanumeric order. To have the items sorted numerically rather than alphanumerically (that is, taking the value of the property as a number rather than a text string), use the negative of the property number. Numeric sorting is descending by default (matching the Finder's normal procedure of sorting sizes and dates in descending order).

For a way to let users sort items in a sublist without using the View menu, see the description of the StaticCommandTextFromView view type on page 5-128.

kDETAspectReverseSort

If you want to sort any properties of items in a sublist in reverse order—that is, descending alphanumeric or ascending numeric order— you must list its property number in a resource of type 'detp'. You must give this resource an ID offset of kDETAspectReverseSort from the base resource ID. The resource can list any property that you have already listed in a kDETAspectViewMenu resource.

```
resource 'detp' (rMyAspectTemplate + kDETAspectReverseSort, purgeable)
{
   { kPrAge }
};
```

The 'detp' resource type is defined as follows:

```
type 'detp' {
   integer = $$CountOf(SortArray);  /* Number of items */
   array SortArray {
      integer;                      /* Property number */
   };
};
```

kDETAspectBalloons

Your aspect template should provide help-balloon strings for any properties that can appear in an information page. The Finder displays a help-balloon string when the user turns on Balloon Help assistance and positions the mouse over a view. For each property you define, you should provide two strings: one to be presented if the property is editable and one to be presented if the property is not editable. The first pair of strings in the resource is used for property number `kDETFirstDevProperty`; the second pair of strings is for property number `kDETFirstDevProperty` + 1; and so forth.

Use a resource of type `'rst#'` to specify the help-balloon strings. You must give this resource an ID offset of `kDETAspectBalloons` from the base resource ID.

```
resource 'rst#' (rMyAspectResourceID + kDETAspectBalloons, purgeable)
{
   {
   "The foobar’s age.",
   "The foobar’s age. Uneditable because the foob is locked or access is
    restricted.",
   "The foobar’s size.",
   "The foobar’s size. Uneditable because the foob is locked or access is
    restricted."
   }
}
```

The Lookup-Table Resource

You can use a lookup-table resource in an aspect template to parse attribute values into properties and properties into attribute values. An aspect-template lookup table contains an entry for each type of attribute value to be translated into and from properties. Attribute values to be translated into properties come from two sources:

- Attribute values in the record or attribute to which the aspect applies. For each attribute type for which the lookup table contains a pattern, the lookup table automatically processes all of the attribute values with that attribute type in this record or attribute.
- Attribute values sent to the lookup table by the code resource. You can use the code resource callback routine `kDETcmdBreakAttribute` (page 5-224) to send to the lookup table an attribute value from anywhere within or outside of an AOCE catalog.

Each lookup-table entry includes a list of attribute types, an attribute tag, a flags field, and a pattern that specifies the mapping between attribute values and properties. The attribute types and tag specify the types of attribute values to be processed. Use a tag value of 0 to process all tag types. The flags further qualify how and when the table entry should be used. The lookup-table flags are shown in Table 5-5 on page 5-109.

In many cases, the translation pattern consists of a single item—indicating that the attribute value maps to a single property. However, it is possible to have much more complex patterns, including variable-length and repeating elements. Figure 5-24 on page 5-107 illustrates the format of a lookup table. There is one entry for each type of data block to be parsed (that is, each attribute value with a specific combination of attribute type and attribute value tag), and each entry contains a list of pattern elements. Each pattern element contains three parts: a format, which drives the parsing process; a property number, telling where to store the result (which may be `kDETNoProperty` if no result should be stored); and an "extra" parameter, which is used by some of the pattern types to specify a second property number. Basic lookup-table pattern elements are shown in Table 5-6 on page 5-111.

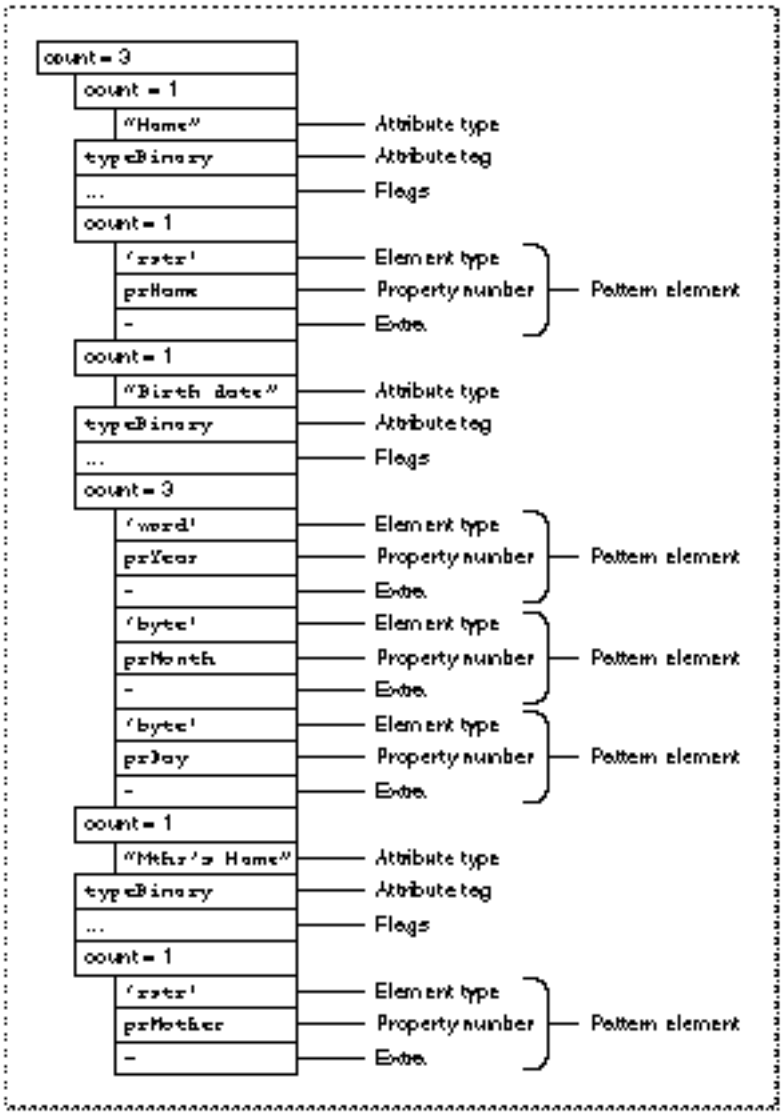
Note that a specific attribute might contain more than one attribute value with the same tag, and you might want to parse these values differently. The lookup-table format includes conditional elements and pattern elements that let you identify each kind of attribute value and process each one appropriately. Lookup-table elements for repeating patterns are shown in Table 5-9 on page 5-115, and elements for conditional patterns are shown in Table 5-7 on page 5-112.

The Catalogs Extension sends any pattern element whose type begins with an uppercase letter to the code resource for processing; see the description of the `kDETcmdPatternIn` routine on page 5-182 for details.

When the CE uses lookup-table patterns to create or update attribute values, a pattern entry can combine any number of property values into a single attribute value. When the user closes an information page, the CE checks whether any properties have changed. If it finds one that has, the CE calls the aspect's code resource (if there is one) with the `kDETcmdValidateSave` routine selector (page 5-168). If the code resource does not want the new property value saved, it returns an error. If the code resource returns `noErr` or `kDETDidNotHandle`, the CE processes all the entries in the lookup table that have the `useForOutput` flag set and that include the property that has changed.

You can provide separate lookup-table entries for input (that is, converting attribute values into property values) and output (converting property values into attribute values), or you can specify that a single entry be used for both purposes.

Figure 5-24 Lookup-table format



kDETAspectLookup

You specify a lookup table with a resource of type `'dett'` with an offset of `kDETAspectLookup` from the template's base resource ID.

```
resource 'dett' (rMyAspectResourceID + kDETAspectLookup, purgeable) {
   {
   {"Name"}, typeBinary,
   useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
      'rstr', prName, 0;         // name
      };
   {"Birthdate"}, typeBinary,
   useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
      'word', prYear, 0;         // year of birth
      'byte', prMonth, 0;        // month of birth
      'byte', prDay, 0;          // day of birth
      };
   {"Mthr's Name"}, typeBinary,
   useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
      'rstr', prMother, 0;       // mother's name
      };
   }
};
```

The format for a lookup-table resource is as follows:

```
type 'dett' {
   integer = $$CountOf(AttributeArray);   /* attribute array size */
   array AttributeArray {
      integer = $$CountOf(TypeArray);     /* attribute type array size */
      array TypeArray {
         RString[32];                     /* attribute type */
      };
      longInt;                            /* attribute tag */

      /* Flags */
      boolean notForInput, useForInput;   /* use this pattern for
                                              input processing? */
      boolean notForOutput, useForOutput; /* use for output processing? */
      boolean notInSublist, useInSublist; /* include attr type in sublist? */
      boolean isNotAlias, isAlias;        /* mark attr value as an alias? */
```

```
      boolean isNotRecordRef, isRecordRef;/* reserved; use isNotRecordRef */

      align word;                         /* reserved */
      integer = $$CountOf(PatternArray);  /* pattern array size */
      array PatternArray {
         longint;                         /* pattern element type */
         integer;                         /* property number */
         integer;                         /* extra parameter */
      };
   };
};
```

IMPORTANT

The pattern element types in lookup tables are case sensitive. Thus, the following two patterns are *not* equivalent:

```
'word', prYear, 0;
'byte', prMonth, 0;
};
{
'Word', prYear, 0;
'Byte', prMonth, 0;
};
```

The Catalogs Extension would interpret the pattern element types `'word'` and `'byte'` as standard types, but would send `'Word'` and `'Byte'` to the code resource for processing. s

The flags field in each entry in the lookup table contains several bits that help select and control the translation process. Table 5-5 shows the currently defined flags (the rest are reserved for future use, and should be set to 0).

Table 5-5 Lookup-table flags

Flag	Meaning
`useForInput`	Use this table entry for translating attribute values to properties.
`useForOutput`	Use this table entry for translating properties to attribute values.
`useInSublist`	Include this attribute value in the sublist. This flag is used in aspect templates of records only. You must set this flag for each attribute type that you want included in the sublist of any information page that has a sublist and that uses this aspect. You should limit sublist items to one line. Multiline sublist items are not guaranteed to work correctly.
`isAlias`	The resulting entry in the sublist is an alias. This flag applies only to attributes in a sublist that hold aliases. If the `useInSublist` flag is not set, the CE ignores this flag.
`isRecordRef`	Reserved; use the `isNotRecordRef` value for this flag.

The Catalogs Extension uses the mapping of properties to attributes provided by the lookup table to check the user's access privileges for each attribute and to mark each property accordingly as either editable or uneditable. The CE can then use this information to allow or prevent a user from making changes in an information page. The CE assumes that properties not associated with any attribute are "internal" and therefore always editable (unless you explicitly make them uneditable with the `kDETcmdSetPropertyEditable` callback routine described on page 5-232).

When processing an output pattern (a lookup-table element that has the `useForOutput` flag set), the CE changes an existing attribute value if there is one or creates a new attribute value if one does not already exist. However, if there is no input pattern for that attribute type in the lookup table, the CE has no way of knowing an attribute value already exists, and so it creates a new one every time it processes the output pattern. In that case, the record ends up containing multiple attribute values corresponding to a single set of properties. Therefore, you must observe this rule:

IMPORTANT

You must always include an input pattern for every attribute type for which you provide an output pattern. s

If your lookup table or code resource writes a zero-length attribute value, the CE deletes that attribute value from the record. Note that an attribute type that contains an `RString` (for example) would not have zero-length attribute value unless the entire `RString` structure were removed; a zero-length `RString` still contains a length and a script code.

The input pattern can be separate from the output pattern, if you wish, and can be empty except for the resource declaration, the attribute type, the attribute tag, and the flags field.

Note

A lookup table can contain only one input pattern and one output pattern for each attribute type. Therefore, although the CE places no restriction on the number of attribute values that can be assigned to each attribute type, lookup-table patterns are designed to work only for those multivalued attributes that appear in sublists.

Multivalued attributes normally appear only in sublists, and the input and output patterns for an attribute in a sublist are normally located in the main aspect for that attribute type, not in the lookup table for the information page that contains the sublist. Therefore, you will not usually set the `useInSublist` flag and the `useForInput` or `useForOutput` flags for the same pattern element.

However, see "Conditional Element Types" on page 5-112 for pattern elements that you can use to develop exceptions to these rules. u

Basic Element Types

A number of pattern element types are available to process single pieces of the pattern. Table 5-6 shows the basic lookup-table element types available.

Table 5-6 Basic lookup-table element types

Element type	Data format	Property type
`'byte'`	Byte (8 bits)	Number
`'word'`	Word (16 bits)	Number
`'long'`	Long word (32 bits)	Number
`'pstr'`	Pascal-style string (8-bit length)	`RString`
`'wstr'`	Pascal-style string (16-bit length)	`RString`
`'cstr'`	C-style (null-terminated) string	`RString`
`'rstr'`	`RString` data structure	`RString`
`'type'`	4-character type	`RString`
`'blok'`	Block of data; the “extra parameter” field holds the number of bytes	Binary
`'bbit'`	Binary bit (8 per byte)	Number
`'abyt'`	Align to byte boundary	None
`'awrd'`	Align to word boundary	None
`'alng'`	Align to long boundary.	None
`'padz'`	Process the following element and pad it to the size specified in the extra field, using zero fill. (Used for fixed-width fields.)	None
`'rest'`	Take everything remaining in the attribute value and put it into a property.	Binary

Listing 5-9 shows the use of basic lookup-table elements to parse an attribute of type Album Track Info into several properties (the number of tracks and the hours, minutes, and seconds of playing time).

Listing 5-9 Lookup table with basic elements

```
// Properties
#define prNumTracks           kDETFirstDevProperty
#define prPlayingTimeHours    kDETFirstDevProperty + 1
#define prPlayingTimeMinutes  kDETFirstDevProperty + 2
#define prPlayingTimeSeconds  kDETFirstDevProperty + 3
```

```
// Lookup table
resource 'dett' (kMyAspectResourceID + kDETAspectLookup, purgeable) {
   {
   {"WAVE Album Track Info"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
         'word', prNumTracks, 0;
         'long', prPlayingTimeHours, 0;
         'long', prPlayingTimeMinutes, 0;
         'long', prPlayingTimeSeconds, 0;
      };
   }
};
```

Conditional Element Types

Lookup tables also provide several element types that implement “if” statements. The element type indicates the test to be performed, the property number indicates which property to test, and the extra field of the element indicates a property number of the property against which the test is performed. If the condition in the test is met, the following element or block is executed. Otherwise, the following element or block is skipped. You can test against a constant value either by using one of the constant property numbers—allowing comparisons against constants in the range 0–249—or by using a resource-based static property. The conditional elements for lookup tables are shown in Table 5-7.

Table 5-7 Conditional elements for lookup tables

Element type	Pattern
`'equa'`	Value of the property specified by the “property number” field equal to value of the property specified by the “extra parameter” field (any type)
`'nteq'`	Value of the property specified by the “property number” field not equal to value of the property specified by the “extra parameter” field (any type)
`'less'`	Value of the property specified by the “property number” field less than value of the property specified by the “extra parameter” field (long integers only)

Table 5-7 Conditional elements for lookup tables (continued)

Element type	Pattern
`'grea'`	Value of the property specified by the “property number” field greater than value of the property specified by the “extra parameter” field (long integers only)
`'leeq'`	Value of the property specified by the “property number” field less than or equal to value of the property specified by the “extra parameter” field (long integers only)
`'greq'`	Value of the property specified by the “property number” field greater than or equal to value of the property specified by the “extra parameter” field (long integers only)

You can use the `'p:=p'` element type to set a property equal to the value of an existing property. The property field in a `'p:=p'` element indicates the destination property. The source property is given by the extra field. The source property (that is, the extra field) can specify a resource.

Listing 5-10 tests whether an album has a playing time of over 1 hour. If it does, the lookup table sets the property `prLongPlay` to `true`.

Listing 5-10 Lookup table with conditional elements

```
resource 'dett' (kMyAspectResourceID + kDETAspectLookup, purgeable) {
   {
   {"WAVE Album Track Info"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
         'word', prNumTracks, 0;
         'long', prPlayingTimeHours, 0;
         'long', prPlayingTimeMinutes, 0;
         'long', prPlayingTimeSeconds, 0;
         'p:=p', prLongPlay, kDETFalseProperty;
         'greq', prPlayingTimeHours, kDETConstantProperty + 1;
         'p:=p', prLongPlay, kDETTrueProperty;
      };
   }
};
```

Block Elements

You can use block elements to form more complex patterns. You form a block by surrounding a list of elements with the `'((((' and '))))'` elements (Table 5-8). The block is then treated conceptually as a single element. Block elements are particularly

useful with repeating and conditional elements. In addition, if the destination property for the '((((' element is something other than kDETNoProperty, everything described by the block is put into the specified property as a binary block. (The destination property of the '))))' element must always be kDETNoProperty.)

Table 5-8 Block elements for lookup tables

Element type	Pattern
'(((('	Begin block
'))))'	End block

Listing 5-11 illustrates the use of the block elements ('((((' and '))))') with a conditional element. This lookup table tests whether an album has a playing time of over 1 hour. If it does, the lookup table sets the property prLongPlay to true and the property prComments to "Long-play album."

Listing 5-11 Lookup table with block elements

```
// Properties

#define prNumTracks           kDETFirstDevProperty
#define prPlayingTimeHours    kDETFirstDevProperty + 1
#define prPlayingTimeMinutes  kDETFirstDevProperty + 2
#define prPlayingTimeSeconds  kDETFirstDevProperty + 3
#define prComments            kDETFirstDevProperty + 4
#define prLongPlayComment     kDETFirstDevProperty + 5

resource 'rstr' (kMyAspectResourceID + prLongPlayComment, purgeable) {
   "Long-play album"
};

// Lookup table
resource 'dett' (kMyAspectResourceID + kDETAspectLookup, purgeable) {
   {
   {"WAVE Album Track Info"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
         'word', prNumTracks, 0;
         'long', prPlayingTimeHours, 0;
         'long', prPlayingTimeMinutes, 0;
         'long', prPlayingTimeSeconds, 0;
         'p:=p', prLongPlay, kDETFalseProperty;
```

```
            'greq', prPlayingTimeHours, kDETConstantProperty + 1;
            '((((', kDETNoProperty, 0;
            'p:=p', prLongPlay, kDETTrueProperty;
            'p:=p', prComments, prLongPlayComment;
            '))))', kDETNoProperty, 0;
        };
    }
};
```

Listing 5-16 on page 5-131 illustrates the use of conditional and block elements in the implementation of a conditional view.

Size Element Types

Lookup tables provide pattern elements that create patterns of a specific size. Table 5-9 shows lookup-table elements you can use for this purpose. An attribute created by one of these pattern elements begins with a length (either a byte, word, or long integer, depending on the element), which is followed by data. The data is in the format specified by the next pattern element. You can use the block elements shown in Table 5-8 on page 5-114 to specify more complex formats for the data. If the data is longer than the pattern element used to format it, the pattern element is repeated as many times as necessary. You can use one of these elements to ensure that a particular attribute is of a specified size and format so that it can be read by a program external to the AOCE catalog system or by a CSAM.

A property created by one of these pattern elements is of property type Binary and of the size specified by the length byte, word, or long integer in the attribute.

Table 5-9 Lookup-table elements that create patterns of a specific size

Element type	Pattern	Property type
`'bsiz'`	Byte-sized size, followed by enough repeats of the following element to use that many bytes	Binary (property does not include size byte)
`'wsiz'`	Word-sized size, followed by enough repeats of the following element to use that many bytes	Binary (property does not include size word)
`'lsiz'`	Long word-sized size, followed by enough repeats of the following element to use that many bytes	Binary (property does not include size long word)

The code fragment in Listing 5-12 stores two properties that are variable-length text strings in a sized attribute. A CSAM or other program reading this attribute from the record could determine how many bytes to read from the length word without having to interpret the constituent RString structures.

Listing 5-12 Lookup table with size and block elements

```
// Properties
#define prAlbumName           kDETFirstDevProperty
#define prPublisherName       kDETFirstDevProperty + 1

// Lookup table
resource 'dett' (kMyAspectResourceID + kDETAspectLookup, purgeable) {
   {
   {"WAVE Album Name Info"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
         'wsiz', kDETNoProperty, 0;
         '((((', kDETNoProperty, 0;
         'rstr', prAlbumName, 0;
         'rstr', prPublisherName, 0;
         '))))', kDETNoProperty, 0;
      };
   }
};
```

In this case, your information page template would assign the prAlbumName and prPublisherName properties to edit-text fields (see “View Lists” beginning on page 5-123). Your default value for each property should be a helpful text string such as “<enter text here>”.

Suppose the user types the string “Apple’s Top Hits” in the album name field and closes the information page. When it processes the lookup-table entry in Listing 5-12, the Catalogs Extension creates an attribute of type WAVE Album Name Info with the attribute tag typeBinary. On a roman script system (the script code is smRoman = 0), the attribute-value data looks like this:

```
0029
0000 0010 4170 706C 6527 7320 546F 7020 4869 7473    '....Apple's Top Hits'
0000 0011 3C65 6E74 6572 2074 6578 7420 6865 7265 3E '....<enter text here>'
```

The block of data begins with a word length (the length of the block of data, not including the length word itself), followed by two RString structures. Each RString begins with the script code (which is $0000 for roman script) followed by a length word followed by the string. Notice that the CE writes the value of the second string even

though it has not been changed, because the CE always fully specifies an attribute when any property derived from that attribute has changed.

The next time the user opens this information page, the CE looks through the lookup table for every attribute type for which the `useForInput` flag has been set. In this case, it finds the attribute type WAVE Album Name Info and finds one attribute value of this type, which has the attribute tag `typeBinary`. The CE finds the entry shown in Listing 5-12, which has this attribute type and tag type, and applies it to the data in the attribute value.

Because the lookup-table entry is of element type `'wsiz'`, the CE knows the first word of the data indicates the length of the rest of the data block. The rest of the lookup-table entry indicates that the data block consists of two `RString` structures, so the CE reads the length, script code, and string from the first `RString` structure and stores the value in the property `prAlbumName`. It then reads the second `RString` the same way, assigning the value to the property `prPublisherName`. The CE stores all strings internally as `RString` structures (see Table 5-2 on page 5-85).

You normally include a destination property for either a size element, for a block element, or for each element within the block, and make the destinations of all the other elements `kDETNoProperty`. To illustrate why this is so, Listing 5-13 shows a lookup-table entry similar to the one in Listing 5-12 except that, in Listing 5-13, the destination property for the `'wsiz'` element is `prNames` rather than `kDETNoProperty`.

Listing 5-13 Lookup-table entry with a destination property for the `'wsiz'` element type

```
// Properties
#define prAlbumName           kDETFirstDevProperty
#define prPublisherName       kDETFirstDevProperty + 1
#define prNames               kDETFirstDevProperty + 2

// Lookup table
resource 'dett' (kMyAspectResourceID + kDETAspectLookup, purgeable) {
   {
   {"WAVE Album Name Info"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
         'wsiz', prNames, 0;
         '((((', kDETNoProperty, 0;
         'rstr', prAlbumName, 0;
         'rstr', prPublisherName, 0;
         '))))', kDETNoProperty, 0;
      };
   }
};
```

In the case of Listing 5-13, when the user types a new value into the album name or publisher name field and closes the information page, the CE first checks whether the binary property `prNames` already exists. If not, the CE uses the new value to create an attribute of type WAVE Album Name Info with the attribute tag `typeBinary`, exactly as it did for the lookup-table element in Listing 5-12 on page 5-116. If the property `prNames` exists, however (as it would if the code resource had already created it), the CE uses the value of the property `prNames` to create the attribute, ignoring the block following the `'wsiz'` element and therefore ignoring the new value typed in by the user.

In either case, the next time the user opens the information page, the CE creates the binary property `prNames` from the value of the attribute WAVE Album Name Info. After creating the property `prNames`, the CE uses the same attribute value to create the `RString` properties `prAlbumName` and `prPublisherName`, as it did for the lookup-table element in Listing 5-12 on page 5-116.

Because the property `prNames` exists, the CE ignores the block following the `'wsiz'` element and uses the value of the property `prNames` to create the attribute. Therefore, if the user changes the value of the property `prAlbumName` or `prPublisherName`, the value of the property `prNames` is not affected, and the changes to `prAlbumName` and `prPublisherName` are not saved. For this reason, you would normally include a destination property for either the size element, for the block element, or for each element within the block, and make the destinations of all the other elements `kDETNoProperty`. If you have some special need to maintain properties for both the block element or the size element and the constituents of the block, you must use your code resource to update property values.

Providing Your Own Pattern Elements

The Catalogs Extension passes to the aspect template's code resource any pattern element type that begins with an uppercase letter. The code resource can then process that portion of the attribute value or property. You can freely mix together built-in and code resource pattern elements.

When creating or changing attribute values, the CE processes only those properties that have changed and that are included in the list of properties in the lookup table. Because you can use a code resource routine invoked by a single pattern element to process any number of properties, you must use the following pattern element to list each property that your code resource processes:

Element type	Pattern
`'prop'`	Include this property in the lookup table

The `'prop'` pattern element lets you associate properties with a lookup table, so that the CE uses that lookup table when those properties need to be saved.

Overriding Default Property-Type Assignments

Each property has a type (see “Properties” beginning on page 5-84). The Catalogs Extension automatically converts between types as needed. The type of the property is taken from the pattern. In most cases, the type is determined automatically—`'byte'` pattern elements produce number type properties; `'rstr'` pattern elements produce string type properties; and so forth. Two pattern elements are provided to allow you to override the default type determination:

Element type	Pattern
`'styp'`	Set the type of the property to the value in the “extra parameter” field
`'btyp'`	Use the value of the “extra parameter” field as the type of all subsequent “binary” properties—for instance, `'(((('`, `'rest'`, `'bsiz'`, `'wsiz'` elements

You can use the `'styp'` and `'btyp'` element types to assign custom property types to properties. The CE calls your code resource when necessary to convert between your custom property types and standard property types; see “Custom Property-Type Conversions” beginning on page 5-188.

Canceling Pattern Processing

Lookup tables provide an element type that aborts processing of the pattern.

Element type	Pattern
`'abrt'`	Abort processing of the pattern

The `'abrt'` lookup-table element stops the processing of the pattern but does not abort the process of reading or writing the attribute value.

Components of Information Page Templates

Information page templates specify the layout and provide the functions of the information pages that users see when they open a record or attribute. The primary content of an information page template is one or more view lists, indicating where on the information page to place the fields (or *views*) that display information to users and allow them to edit that information.

The Catalogs Extension fills in the views from an aspect specified by the information page. A view list specifies only the property numbers of properties in that aspect. All of the information in the main part of an information page (that is, all of the information page except for items in a sublist) comes from the same, specified aspect.

An information page template can contain the resources listed in Table 5-10.

Table 5-10 Resources in information page templates

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
'deti'	0	Identifies template as information page and provides a base resource ID. Required for all information page templates.
'rstr'	kDETTemplateName	Name of template. Required for all information page templates.
'rstr'	kDETRecordType	Type of record to which the template applies. Either this resource, the kDETAttributeType resource, or both must be included.
'rstr'	kDETAttributeType	Type of attribute to which the template applies. Either this resource, the kDETRecordType resource, or both must be included.
'detn'	kDETAttributeValueTag	Attribute tag of attributes to which the template applies. You can provide this resource if you have also provided the kDETAttributeType resource. If you don't provide this resource, the template applies to attributes with any tag value.
'detv'	The resource ID of a view list is independent of that of the information page signature resource.	View list. An information page template can contain any number of view lists describing specific items in the information page and in the sublist (if any). The information page signature resource lists the resource IDs of all of the applicable view list resources. Each information page must have at least one view list if it is to provide any useful information.
'rstr'	kDETInfoPageMainViewAspect	The name of the aspect template whose aspect provides all of the properties for the views in the main portion of the information page. This aspect also lists the objects that go in the sublist (if any). Required for every information page template.
'rstr'	kDETInfoPageName	The name of the information page that appears in the page-selection pop-up menu. Required for all information pages that appear in a page-selection pop-up menu.
'detm'	kDETInfoPageMenuEntries	A list of items that the CE adds to the end of the Catalogs menu. When the user chooses one of these items, the CE calls the code resource of the aspect. Optional.

Information Page Template Signature Resource

The information page seen by the user is the combination of one or more view lists specified by the template. The signature resource of the information page lists the resource IDs of the view lists to be used in the main and icon-list parts of the information page. Each view list resource ID is preceded by two property numbers. The view list is displayed only if the contents of the two properties are equal or if either property equals `kDETNoProperty`. This feature allows you to enable or disable a view list according to the current values of properties in the associated aspect. Thus, an information page can change to include different views according to the current state of the record or attribute represented by that page.

The signature resource also specifies the sort-order number of the information page, the presence or absence of a sublist in the information page, and the rectangle that contains the sublist. The Catalogs Extension displays the information pages in the sequence indicated by their sort-order numbers. When creating several information pages for a single catalog object, you should assign sort-order numbers at intervals of 1000 (1000, 2000, 3000, and so on) to allow a new page to be easily inserted between two existing pages.

The signature resource includes a Boolean value that indicates whether the first edit-text field of the information page should be automatically selected when the information page is opened. If you set this value to `selectFirstText`, the CE selects the first edit-text field when a user opens the information page (as is usual for a dialog box). If you set this value to `noSelectFirstText`, no field is initially selected. For most information pages, you should set the Boolean value to `noSelectFirstText` to lessen the likelihood that users will change a field unintentionally.

'**deti**' Resource

The signature resource for an information page template is of type `'deti'`.

```
type 'deti' {
   longInt = kDETInfoPageVersion;   /* template format version */
   longInt;                         /* sort order */
   rect;                            /* rectangle to put sublist in */
   boolean selectFirstText, noSelectFirstText;
                                    /* select the first text field
                                        when info-page opens? */
   align word;                      /* reserved */
   integer = $$CountOf(HeaderViewArray);
   array HeaderViewArray {          /* the header view lists */
      integer;                      /* property 1 */
      integer;                      /* property 2 */
      integer;                      /* 'detv' ID */
```

```
   };
   integer = $$CountOf(SubviewViewArray);
   array SubviewViewArray {      /* the subview view lists */
      integer;                   /* property 1 */
      integer;                   /* property 2 */
      integer;                   /* 'detv' ID */
   };
};
```

Listing 5-14 is an example of an information page template signature resource. This resource specifies four view lists. The first describes a view that always appears on the information page. The second and third are conditional views; each appears on the information page only if the two properties associated with that view are equal (or if either property number equals `kDETNoProperty`). In Listing 5-14, the view list with an ID of `rMyInfoPage` + 1 appears if the property `prEnable1` equals `kDETZeroProperty` (that is, 0) and the view list `rMyInfoPage` + 2 appears if `rEnable1` equals `kDETOneProperty` (that is, 1). The fourth view list in Listing 5-14 describes a line in the sublist and is always enabled.

Listing 5-14 Information page signature resource with conditional views

```
resource 'deti' (rMyInfoPage, purgeable) {
   1000,                                          // sort order
   kDETSublistRect,                               // rect for sublist
   noSelectFirstText,
   {
   // View list for main part of window:
   kDETNoProperty, kDETNoProperty, rMyInfoPage;    // always enabled

   // View lists for conditional views in main part of window
   prEnable1, kDETZeroProperty, rMyInfoPage + 1;   // enabled if prEnable1
                                                   //  property is 0
   prEnable1, kDETOneProperty, rMyInfoPage + 2;    // enabled if prEnable1
                                                   //  property is 1
   },
   // View lists for sublist:
   {
   kDETNoProperty, kDETNoProperty, rMyInfoPage + 3;// always enabled
   }
};
```

Listing 5-16 on page 5-131 further illustrates the implementation of conditional views.

If the information page has no sublist, you can use the following sublist rectangle:

```
#define kDETNoSublistRect  {0, 0, 0, 0}
```

View Lists

A view list specifies individual items on the 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. A view list need not use all of the properties in the aspect. On the other hand, a single property can provide information for more than one view. For example, a set of three radio buttons would require three views in a view list but could all display the information in a single property.

'detv' Resource

A view list is defined by the `'detv'` resource type.

```
type 'detv' {
   longint = 0;
   longint = 0;
   longint = 0;
   integer = 0;
   longint = 0;
   longint = 0;
   longint = 0;
   longint = 0;
   integer = 0;
   longint = 0;
   longint = 0;

   integer = $$CountOf(ItemArray);/* count */
   array ItemArray {
      rect;                  /* bounds of the view */
      longint = 0;           /* position flags (preset by CE) */
      longint;               /* flags (described following this struct) */
      integer;               /* property number */
      switch {               /* class of view */

         case StaticTextFromView:
            key longint = 7750;/* class ID */
            integer;         /* font ID */
            integer;         /* font size */
```

```
      integer;          /* justification */
      integer;          /* style */
      pstring;          /* string to display */

   case StaticCommandTextFromView:
      key longint = 22250;/* class ID */
      integer;          /* font ID */
      integer;          /* font size */
      integer;          /* justification */
      integer;          /* style */
      pstring;          /* string to display */
      align word;       /* reserved */
      longint;          /* property command */
      integer;          /* command parameter */

   case StaticText:
      key longint = 3250;/* class ID */
      integer;          /* font ID */
      integer;          /* font size */
      integer;          /* justification */
      integer;          /* style */

   case EditText:
      key longint = 8250;/* class ID */
      integer;          /* font ID */
      integer;          /* font size */
      integer;          /* justification */
      integer;          /* style */

   case Bitmap:
      key longint = 6250;
      integer;          /* size */

   case Box:
      key longint = 4750;/* class ID */
      integer;          /* box attributes */

   case DefaultButton:
      key longint = 7250;/* class ID */
      integer;          /* font ID */
      integer;          /* font size */
      integer;          /* justification */
      integer;          /* style */
```

```
   pstring;          /* label for button */
   align word;       /* reserved */
   longint;          /* property command */

case Button:
   key longint = 21000;/* class ID */
   integer;          /* font ID */
   integer;          /* font size */
   integer;          /* justification */
   integer;          /* style */
   pstring;          /* label for button */
   align word;       /* reserved */
   longint;          /* property command */

case CheckBox:
   key longint = 21250;/* class ID */
   integer;          /* font ID */
   integer;          /* font size */
   integer;          /* justification */
   integer;          /* style */
   pstring;          /* label for checkbox */
   align word;       /* reserved */
   longint;          /* property command */

case RadioButton:
   key longint = 21500;/* class ID */
   integer;          /* font ID */
   integer;          /* font size */
   integer;          /* justification */
   integer;          /* style */
   pstring;          /* label for button */
   align word;       /* reserved */
   longint;          /* property command */
   integer;          /* command parameter */

case Menu:
   key longint = 5750;/* class ID */
   integer;          /* font ID */
   integer;          /* font size */
   integer;          /* justification */
   integer;          /* style */
   pstring;          /* label for pop-up menu */
   align word;       /* reserved */
```

```
            longint;          /* property command */
            integer;          /* menu resource ID */

         case EditPicture:
            key longint = 0x00010000 + 24250; /* class ID */
            integer;          /* maximum pixel depth */

         case Custom:
            key longint = 6750;/* class ID */
            integer;          /* reference value for use by developer */
      };
      align word;
   };
};
```

Here are the possible values for the flags field near the beginning of the `'detv'` resource:

Flag	Meaning
`kDETNoFlags`	No flags.
`kDETEnabled`	If set to 1, this view should be highlighted if the user selects the item of which this view is a part. Not for use in sublists.
`kDETHilightIfSelected`	If set to 1, highlight the view when the entry is selected. For items in a sublist only.
`kDETNumericOnly`	If set to 1, the user is allowed to enter only numbers into this item. For editable text fields only.
`kDETMultiLine`	If set to 1, the user or template can enter more than one line into the field. If set to 0, pressing Return terminates text entry; if set to 1, pressing Return enters a carriage return and starts a new line. For text fields only. Note that you must set this flag to 1 for multiline static text fields.
`kDETDynamicSize`	If set to 1, a box appears around the item when the user clicks the item or tabs into it. The CE sizes the box dynamically as the user edits the text in the box. For editable text fields only.
`kDETAllowNoColons`	If set to 1, the user is not allowed to enter colons as part of the text. The CE substitutes a hyphen (-) for each colon (:) typed. For editable text fields in which the user is expected to type a filename.
`kDETPopupDynamicSize`	If set to 1, the CE automatically resizes a pop-up menu according to its contents. For pop-up menus only.
`kDETScaleToView`	If set to 1, a picture is scaled to the bounds of the view. If set to 0, the picture is cropped. For `EditPicture` views only.

Several constants are provided for use with fonts:

Constant	Value
kDETApplicationFont	1
kDETApplicationFontSize	9
kDETAppFontLineHeight	12
kDETSystemFont	0
kDETSystemFontSize	12
kDETSystemFontLineHeight	16
kDETDefaultFont	1
kDETDefaultFontSize	9
kDETDefaultFontLineHeight	12

Here are the possible values for text styles:

Constant	Meaning
kDETNormal	Normal font
kDETBold	Bold
kDETItalic	Italic
kDETUnderline	Underlined
kDETOutline	Outline style
kDETShadow	Shadow style
kDETCondense	Condensed
kDETExtend	Extended
kDETIconStyle	Same as normal text style for regular entries and as italic text style for aliases

Here are the possible values for text justification:

Value	Meaning
kDETLeft	Text in scripts written from left to right is left-justified; text in scripts written from right to left is right-justified.
kDETCenter	All text is centered in the text field.
kDETRight	All text is right-justified.
kDETForceLeft	All text is left-justified.

View types

`StaticTextFromView`

Text that cannot be edited by the user. The contents of the string come from the view itself; that is, from the information page template.

`StaticCommandTextFromView`
: Text that cannot be edited by the user. When the user clicks on the text, the CE calls your code resource with the routine selector `kDETcmdPropertyCommand` and with the property command and command parameter from the view list. Most commonly, the code resource does nothing in response to this property command; instead, this view type is used to provide headings for sublist columns, so that when the user clicks the column heading, the sort criteria for the sublist is changed. To accomplish this, set the property-command field to `kDETChangeViewCommand` and the command parameter field to the negative of the property number of the appropriate entry in the aspect's `kDETAspectViewMenu` resource (page 5-103). When you use `kDETChangeViewCommand` as the property command, the CE handles the command without calling your code resource. The contents of the string come from the "string to display" field in the view itself.

`StaticText`
: Text that cannot be edited by the user. The contents of the string come from the view property; that is, from the aspect.

`EditText`
: Text that the user can edit. The contents of the string come from the view property; that is, from the aspect.

`Bitmap`
: An icon, taken from an icon suite in the aspect template. The CE looks for an icon suite with a resource ID equal to the aspect template's base ID plus the property number. The property number is in the property number field of the `'detv'` structure. The size field indicates the size of the icon: `kDETLargeIcon`, `kDETSmallIcon`, or `kDETMiniIcon`.

`Box`
: A simple graphic rectangle or rounded rectangle, useful for drawing dividing lines between view elements and boxes around elements that do not normally have them, such as sublists. The `Box` view type takes a single integer as a parameter. The dimensions of the box are those of the view itself. The first 4 bytes of the integer are flags, as follows:

Bit	Flag	Meaning
none	`kDETUnused`	No flags.
0	`kDETBoxTakesContentClicks`	If this flag is set to 1 and the user clicks in the box, the CE calls your code resource with the property number.
1	`kDETBoxIsRounded`	Box is a rectangle with rounded corners.
2	`kDETBoxIsGrayed`	Box is dimmed.
3	`kDETBoxIsInvisible`	Box is invisible.

`DefaultButton` A standard button with a heavy border indicating that this is the default button; that is, pressing Return or Enter keys has the same effect as clicking the button. Clicking a default button closes any open edit-text field, whereas clicking a regular button does not. In every other respect, a `DefaultButton` view is identical to a `Button` view. You can have only one default button in a given information page.

`Button` A standard button. You must use the button's property number for the property-command field of the `'detv'` resource. Then, when the user clicks the button, the CE calls your code resource with the routine selector `kDETcmdPropertyCommand` and with the property number of the button in the `property` field of the parameter block. If your code resource does not handle this command, the CE does nothing. If you use the property numbers `kDETAddNewItem`, `kDETRemoveSelectedItems`, or `kDETOpenSelectedItems`, the CE handles the command without calling your code resource. These property numbers are described in Table 5-3 on page 5-86.

`Checkbox` A standard dialog checkbox, which can be selected or not. The property can be equal to 0 (checkbox is not selected) or 1 (checkbox is selected). You must use the checkbox's property number for the property-command field of the `'detv'` resource. Then when the user clicks the checkbox, the CE sends the property number to your code resource. If your code resource does not handle the command, the CE changes the property value to toggle the checkbox off or on.

Note that if your code resource does handle this command, you must call the `kDETcmdDirtyProperty` callback routine (page 5-233) to force the CE to redraw the checkbox.

`RadioButton` A standard dialog radio button, which can be on or off, as selected by the user. When multiple radio buttons are associated with the same property, only one can be on at a time. You must use the button's property number for the property-command field of the `'detv'` resource, and you should use a different value for the command parameter field of each button associated with the same property. Set the command parameter of the default button (the one you want the CE to select when the view is first displayed) equal to the value of the property. When the user clicks the radio button, the CE first calls your code resource. If your code resource does not handle the command, the CE sets the value of the property to the value of the parameter for that button, thereby selecting that button and deselecting all other radio buttons for that property.

Note that if your code resource does handle this command, you must call the `kDETcmdDirtyProperty` callback routine (page 5-233) to force the CE to redraw the radio button.

`Menu` A pop-up menu from which the user can select one item, which the information page then displays as the "state" of the menu. The menu resource ID field of the `'detv'` resource indicates the `'fmnu'` resource that specifies the contents of the menu. The menu resource can have any resource ID. You must use the menu's

property number for the property-command field of the 'detv' resource. Pop-up menus are limited to 31 items; if your 'fmnu' resource includes more than 31 items, the pop-up menu will not work properly. You cannot put a pop-up menu view in a sublist.

When the user chooses an item in the pop-up menu, the CE calls your code resource with the routine selector kDETcmdPropertyCommand and with the property number of the menu in the property field of the parameter block. The CE gets the value for the parameter field of the command's parameter block from the 'fmnu' resource; your code resource can use this parameter to determine which item in the pop-up menu the user has selected. You can use the kDETcmdAddMenu (page 5-238) and kDETcmdRemoveMenu (page 5-240) callback routines to add and remove menu items.

EditPicture — A picture that the user can select and copy onto the Clipboard, cut, or replace by pasting.

Custom — A custom view defined by the code resource of the aspect associated with the information page. A custom view can respond to mouse-down events if you provide code to handle these events. There is no way for the user to select the custom view, but if no other view is selected, your code resource can receive keypress events and interpret them as belonging to the custom view. See "Custom Views and Custom Menus" beginning on page 5-192 for more information about how code resources can handle custom views. You can specify any value you wish for the reference-value integer in the view list. Your code resource can use the kDETcmdGetCustomViewUserReference callback routine (page 5-242) to obtain this value.

Listing 5-15 shows a sample view list.

Listing 5-15 Sample view list

```
resource 'detv' (rMyInfoPage, purgeable) {
   {
   {kBitmapTop, kBitmapLeft, kBitmapBottom, kBitmapRight},
   kDETNoFlags, kDETAspectMainBitmap,
   Bitmap { kDETLargeIcon };

   {kStatTextTop, kStatTextLeft, kStatTextBottom, kStatTextRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
                        kDETRight, kDETBold, "Label:" };

   {kEditTextTop, kEditTextLeft, kEditTextBottom, kEditTextRight},
   kDETEnabled, prEditTextProperty,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
```

```
            kDETNormal};

   {kCheckboxTop, kCheckboxLeft, kCheckboxBottom, kCheckboxRight},
   kDETNoFlags, prCheckboxProperty,
   CheckBox { kDETApplicationFont, kDETApplicationFontSize,
            kDETLeft, kDETNormal, "Check Me", prCheckboxProperty };

   {kRadio1Top, kRadio1Left, kRadio1Bottom, kRadio1Right},
   kDETNoFlags, prRadioProperty,
   RadioButton { kDETApplicationFont, kDETApplicationFontSize,
               kDETLeft, kDETNormal, "Radio 1", prRadioProperty, 0 };

   {kRadio2Top, kRadio2Left, kRadio2Bottom, kRadio2Right},
   kDETNoFlags, prRadioProperty,
   RadioButton { kDETApplicationFont, kDETApplicationFontSize,
               kDETLeft, kDETNormal, "Radio 2", prRadioProperty, 1 };
   }
};
```

Implementing Conditional Views

Listing 5-16 shows a lookup table, information page signature resource, and view list used to implement a conditional view. In Listing 5-16, the information page contains a view-selection pop-up menu with three choices. When the user chooses an item from the menu, the value of that item becomes the value of the property `prMsgType`. In the `'deti'` resource, the value of `prMsgType` determines which view is active. In the lookup table (the `'dett'` resource), the value of `prMsgType` determines which properties are processed. Note the use of block and conditional elements in the lookup table (see “Conditional Element Types” on page 5-112 and “Block Elements” on page 5-113) to achieve this end. There is one view list for the pop-up menu and one view list for each of the conditional views.

Note

Listing 5-16 is not a complete, working set of templates. It is intended only to illustrate the interaction of the information page signature resource, lookup table, and view lists in the implementation of conditional views. u

Listing 5-16 Implementing a conditional view

```
#define prMsgType       kDETFirstDevProperty
#define prPMDate        kDETFirstDevProperty + 1
#define prPMTime        kDETFirstDevProperty + 2
#define prPMFrom        kDETFirstDevProperty + 3
```

```
#define prPMTo          kDETFirstDevProperty + 4
#define prNMessage      kDETFirstDevProperty + 5
#define prNReply        kDETFirstDevProperty + 6
#define prIBOFrom       kDETFirstDevProperty + 7
#define prIBOTo         kDETFirstDevProperty + 8
#define prIBOBecause    kDETFirstDevProperty + 9
resource 'deta' (kCondViewAspect, purgeable) {
   0,                // drop-operation order
   dropCheckAlways,  // drop-check flag
   notMainAspect     // not the main aspect
};

resource 'rstr' (kCondViewAspect + kDETTemplateName, purgeable) {
   "WAVE Conditional view aspect"   // Start with application signature
};

resource 'rstr' (kCondViewAspect + kDETRecordType, purgeable) {
   "WAVE Conditional View"          // Start with application signature
};

// Custom information page window with no default pop-up menu
resource 'detw' (kCondViewAspect + kDETAspectInfoPageCustomWindow, purgeable)
   {
   {0,0,224,224},
   discludePopup
   };

// Lookup table. Conditional elements correspond to conditional views.
resource 'dett' (kCondViewAspect + kDETAspectLookup, purgeable) {
   {
   {"aoce MailNote"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
         'long', prMsgType, 0;
         'equa', prMsgType, kDETConstantProperty + 0;
            '((((', kDETNoProperty, 0;
            'rstr', prPMDate, 0;
            'rstr', prPMTime, 0;
            '))))', kDETNoProperty, 0;
         'equa', prMsgType, kDETConstantProperty + 1;
            '((((', kDETNoProperty, 0;
            'rstr', prNMessage, 0;
            'rstr', prNReply, 0;
```

```
            '))))', kDETNoProperty, 0;
         'equa', prMsgType, kDETConstantProperty + 2;
            '((((', kDETNoProperty, 0;
            'rstr', prIBOFrom, 0;
            'rstr', prIBOTo, 0;
            'rstr', prIBOBecause, 0;
            '))))', kDETNoProperty, 0;
      };
   }
};

resource 'deti' (kCondViewInfoPage, purgeable) {
   1000,
   kDETNoSublistRect,
   noSelectFirstText,
   {
   kDETNoProperty, kDETNoProperty, kCondViewInfoPage;
   prMsgType, kDETConstantProperty + 0, kCondViewInfoPage + 1;
   prMsgType, kDETConstantProperty + 1, kCondViewInfoPage + 2;
   prMsgType, kDETConstantProperty + 2, kCondViewInfoPage + 3;
   },
   {
   }};

resource 'rstr' (kCondViewInfoPage + kDETTemplateName, purgeable) {
   "WAVE Conditional view info page"
};

resource 'rstr' (kCondViewInfoPage + kDETRecordType, purgeable) {
   "WAVE Conditional view"
};

resource 'rstr' (kMNInfoPage + kDETInfoPageMainViewAspect, purgeable) {
   "WAVE Conditional view aspect"
};

//View list for conditional-view-selection pop-up menu
resource 'detv' (kMNInfoPage, purgeable) {
   {
   kMenuTop, kMenuLeft, kMenuBottom, kMenuRight,
   kDETNoFlags, prMsgType,
   Menu {kDETSystemFont, kDETSystemFontSize, kDETCenter, kDETNormal, "",
         prMsgType, kMNInfoPage};
```

```
   }
}

//Menu resource for conditional-view-selection pop-up menu
resource 'fmnu' (kMNInfoPage, purgeable)
   {
   kMNInfoPage,
   {
   0,”Phone Message”;
   1,”Note”;
   2,”I’ll Be Out”;
   }
   };

//View lists for conditional views
resource 'detv' (kMNInfoPage + 1, purgeable) {
   {
   {kTextTop, kLabelLeft, kTextTop + kOneLineHeight, kLabelRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kLabelFont, kLabelSize, kDETRight, kLabelStyle,
   "Date:" };

   {kTextTop - 2, kTextColumnLeft, kTextTop + kOneLineHeight - 2,
   kTextRight},
   kDETEnabled, prPMDate,
   EditText { kTextFont, kTextSize, kDETLeft, kTextStyle };

   {kTextTop + kOneLineHeight + 2, kLabelLeft,
   kTextTop + kOneLineHeight + 2 + kOneLineHeight, kLabelRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kLabelFont, kLabelSize, kDETRight, kLabelStyle,
   "Time:" };

   {kTextTop + kOneLineHeight + 2 - 2, kTextColumnLeft,
   kTextTop + kOneLineHeight + 2 + kOneLineHeight - 2, kTextRight},
   kDETEnabled, prPMTime,
   EditText { kTextFont, kTextSize, kDETLeft, kTextStyle };
   };
};

resource 'detv' (kMNInfoPage + 2, purgeable) {
   {
   {kTextTop - 2, kTextLeft, kTextTop + kSixLineHeight - 2, kTextRight},
```

```
kDETMultiLine, rNMessage,
   EditText { kTextFont, kTextSize, kDETLeft, kTextStyle };

   {kTextTop + kSixLineHeight + 2, kLabelLeft, kTextTop + kSixLineHeight +
2 + kOneLineHeight, kLabelRight}, kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kLabelFont, kLabelSize, kDETRight, kLabelStyle,
"Reply:" };

   {kTextTop + kSixLineHeight + 2 + kOneLineHeight + 2 -6, kTextLeft,
kTextBottom, kTextRight}, kDETMultiLine, rNReply,
   EditText { kTextFont, kTextSize, kDETLeft, kTextStyle };
   };
};

resource 'detv' (kMNInfoPage + 3, purgeable) {
   {
   {kTextTop, kLabelLeft, kTextTop + kOneLineHeight, kLabelRight},
kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kLabelFont, kLabelSize, kDETRight, kLabelStyle,
"From:" };

   {kTextTop - 2, kTextColumnLeft, kTextTop + kOneLineHeight - 2,
kTextRight}, kDETEnabled, rIBOFrom,
   EditText { kTextFont, kTextSize, kDETLeft, kTextStyle };

   {kTextTop + kOneLineHeight + 2, kLabelLeft, kTextTop + kOneLineHeight +
2 + kOneLineHeight, kLabelRight}, kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kLabelFont, kLabelSize, kDETRight, kLabelStyle, "To:"
};

   {kTextTop + kOneLineHeight + 2 - 2, kTextColumnLeft, kTextTop +
kOneLineHeight + 2 + kOneLineHeight - 2, kTextRight}, kDETEnabled, rIBOTo,
   EditText { kTextFont, kTextSize, kDETLeft, kTextStyle };

   {kTextTop + kOneLineHeight + 2 + kOneLineHeight + 2, kLabelLeft,
kTextTop + kOneLineHeight + 2 + kOneLineHeight + 2 + kOneLineHeight,
kLabelRight}, kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kLabelFont, kLabelSize, kDETRight, kLabelStyle,
"Because:" };

   {kTextTop + kOneLineHeight + 2 + kOneLineHeight + 2 - 2,
kTextColumnLeft, kTextTop + kOneLineHeight + 2 + kOneLineHeight + 2 +
kTwoLineHeight - 2, kTextRight}, kDETEnabled, rIBOBecause,
```

```
   EditText { kTextFont, kTextSize, kDETLeft, kTextStyle };
   };
};
```

Sublists

If the information page includes a sublist, the information page template includes a view list that describes a line in the list. Each line in a sublist typically contains an icon, a "name" field, and a "kind" field, and might contain other fields or controls. For each item in the sublist, the Catalogs Extension takes the properties for the views from the main aspect of that item.

Note that, if the sublist contains items of more than one type, each type of item has an associated main aspect template. For example, if the sublist contains Direct Dialup mail addresses, PowerTalk serverless mail addresses, and PowerShare server mail addresses, there are three main aspect templates, one for each type of mail address. The CE uses the appropriate main aspect template to create the main aspect for each item that appears in the list. The CE uses a single view list in the information page template to format all of the lines in the sublist but takes the values to display in the sublist from a separate main aspect for each line.

The CE does not automatically draw a box around a sublist; if you want a box around a sublist, you must draw it yourself.

View lists are described in the preceding section. Main aspect templates are described in "Main Aspect Template Resources" beginning on page 5-88. Listing 5-4 on page 5-44 shows an information page template for an information page with a sublist.

Information Page Resources

In addition to view lists, an information page template contains resources that name the template, specify the type of record or attribute to which the template applies, provide the name of the associated aspect, and specify items for the Catalogs menu. The information page template resources that are common to aspect templates are described in "Template Names" on page 5-75 and "Specifying Record and Attribute Types for Templates" on page 5-75. The remaining resources are described in this section. For a complete list of information page resources, see Table 5-10 on page 5-120.

kDETInfoPageMainViewAspect

All property numbers listed by the view lists for the main portion of the information page come from one aspect, the *main view aspect*. The main view aspect also provides the list of objects to be included in the sublist (if any). Name the main view aspect with an `RString` resource that has a resource ID with an offset of `kDETInfoPageMainViewAspect` from the template's base resource.

Note
Do not confuse the *main view aspect* with a *main aspect*. A main aspect provides the properties that describe an item in a sublist. A main view aspect provides all of the properties needed by the information page *except* the contents of the sublist. u

```
resource 'rstr' (rMyInfoPage + kDETInfoPageMainViewAspect, purgeable) {
   "WAVE Associated Aspect Name"
};
```

IMPORTANT
If the information page template does not name a main view aspect, the Catalogs Extension does not load the template. s

kDETInfoPageName

To specify the name of the information page that appears in the page-selection pop-up menu, use an `RString` resource that has a resource ID with an offset of `kDETInfoPageName` from the template's base resource.

```
resource 'rstr' (rMyInfoPage + kDETInfoPageName, purgeable) {
   "Information Page Pop-up Name"
};
```

You must include this resource in every information page template if the information page window includes a page-selection pop-up menu. Only custom information pages can exclude page-selection pop-up menus; see the description of the `kDETAspectInfoPageCustomWindow` resource on page 5-97.

kDETInfoPageMenuEntries

You can use a resource of type `'detm'` to add items to the Catalogs menu. Give the resource an ID with an offset of `kDETInfoPageMenuEntries` from the template's base resource ID.

```
resource 'detm' (rMyInfoPage + kDETInfoPageMenuEntries, purgeable) {
   rMyInfoPage + kDETInfoPageMenuEntries,
   {
      menuParameter1, "Entry 1";
      menuParameter2, "Entry 2";
   }
};
```

When the user chooses an item, the Catalogs Extension calls the code resource in the aspect with the routine selector `kDETcmdCustomMenuSelected` (page 5-195), passing your code the menu parameter from the `kDETInfoPageMenuEntries` resource for the item the user selected. Your code resource is also called (with the `kDETcmdCustomMenuEnabled` routine selector described on page 5-194) to determine if the menu item should be enabled.

Components of Forwarder Templates

Forwarder templates provide a list of names of aspect and information page templates that can be used with the record or attribute type to which the template applies.

A forwarder template can contain the resources listed in Table 5-11.

Table 5-11 Resources in forwarder templates

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
`'detf'`	0	Identifies template as forwarder and provides a base resource ID. Required for all forwarder templates.
`'rstr'`	`kDETTemplateName`	Name of template. Required for all forwarder templates.
`'rstr'`	`kDETRecordType`	Type of record to which the template applies. Either this resource, the `kDETAttributeType` resource, or both must be included.
`'rstr'`	`kDETAttributeType`	Type of attribute to which the template applies. Either this resource, the `kDETRecordType` resource, or both must be included.
`'detn'`	`kDETAttributeValueTag`	Attribute tag of attributes to which the template applies. You can provide this resource if you have also provided the `kDETAttributeType` resource. If you don't provide this resource the template applies to attributes with any tag value.
`'rst#'`	`kDETForwarderTemplateNames`	A list of names of aspect and information page templates that the CE can use with the record or attribute type to which this forwarder template applies.

Forwarder Template Signature Resource

The forwarder template signature resource provides a base resource ID from which the other resource IDs in the template are offset and provides the forwarder template version number of the template.

'detf' Resource

Use a resource of type `'detf'` for the forwarder template signature resource.

```
type 'detf' {
   longInt = kDETForwarderVersion;  /* template format version */
};
```

Forwarder Template Resources

A forwarder template contains resources that name the template, specify the type of record or attribute to which the template applies, and specify the names of templates that the Catalogs Extension can use with the record or attribute to which the template applies. The forwarder template resources that are common to aspect and information page templates are described in “Template Names” on page 5-75 and “Specifying Record and Attribute Types for Templates” on page 5-75. The remaining resource is described in this section. For a complete list of forwarder template resources, see Table 5-11 on page 5-138.

kDETForwarderTemplateNames

To specify the templates that the Catalogs Extension can use with the record or attribute to which the forwarder template applies, use a resource of type `'rst#'` that has a resource ID with an offset of `kDETForwarderTemplateNames` from the template’s base resource.

```
resource 'rst#' (kForwarderTemplate + kDETForwarderTemplateNames,
purgeable) {
   { "WAVE Album", "WAVE Track" }
};
```

Components of Killer Templates

A killer template provides a list of names of templates that the Catalogs Extension should ignore. You can use killer templates to disable any type of template except another killer template. A killer template can contain the resources listed in Table 5-12.

Table 5-12 Resources in killer templates

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
`'detk'`	0	Identifies template as killer and provides a base resource ID. Required for all killer templates.
`'rstr'`	`kDETTemplateName`	Name of template. Required for all killer templates.
`'rst#'`	`kDETKillerName`	A list of names of templates to disable.

Killer Template Signature Resource

The killer template signature resource provides a base resource ID from which the other resource IDs in the template are offset and provides the killer template version number of the template.

'**detk**' **Resource**

Use a resource of type `'detk'` for the killer template signature resource.

```
type 'detk' {
   longInt = kDETKillerVersion;  /* template format version */
};
```

Killer Template Resources

A killer template contains resources that name the template and specify the names of templates that the Catalogs Extension should ignore. The killer template name resource is described in “Template Names” on page 5-75. The other resource is described in this section. For a complete list of killer template resources, see Table 5-12.

kDETKillerName

To specify the templates that the CE should disable, use a resource of type `'rst#'` that has a resource ID with an offset of `kDETKillerName` from the template’s base resource.

```
resource 'rst#' (kKillerTemplate + kDETKillerName, purgeable) {
   { "WAVE Album", "WAVE Track" }
};
```

Components of File Type Templates

A file type template provides a list of file types that the Catalogs Extension should search for AOCE templates. A file type template can contain the resources listed in Table 5-13.

Table 5-13 Resources in file type templates

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
`'detx'`	0	Identifies template as file type and provides a base resource ID. Required for all file type templates.
`'rstr'`	`kDETTemplateName`	Name of template. Required for all file type templates.

File Type Template Signature Resource

The file type template signature resource provides a base resource ID from which the other resource IDs in the template are offset, provides the file type template version number of the template, and lists the file types that the CE is to search for templates.

'**detx**' Resource

Use a resource of type `'detx'` for the file type template signature resource.

```
type 'detx' {
   longInt = kDETFileTypeVersion;      /* template format version */
   integer = $$CountOf(ItemArray);     /* count */
   array ItemArray {
      longInt;                         /* type of additional file */
   };
};
```

File Type Template Resources

A file type template contains a resource that names the template. The file type template name resource is described in “Template Names” on page 5-75.

Code Resources Reference

This section describes the data types and CE-provided routines you can use in a code resource (resource type `'detc'`). It also describes the routines that your code resource can provide and the circumstances in which the CE calls each of these routines.

Rules for Writing Code Resources

Because AOCE templates extend the Finder and are called by the Finder, it is possible for a code resource routine to corrupt the Finder or cause it to crash. To make sure that your code resource causes no problems, follow these rules:

- Use as little memory as possible. Try to allocate all the memory you need when you initialize the template (in your `kDETcmdInit` routine, page 5-150) and provide error handling for insufficient-memory cases whenever you allocate memory.
- Don't use global variables. The CE does not maintain an A5 world for template code resources. If your compiler uses global space for inline code, you must not use such code in your routines.
- Don't assume that the CE locks down your code resource. In the interval between calls by the CE to your code resource, your code is unlocked and purgeable. You cannot use callback or completion routines for operations that don't complete before they return to the CE. If you must use a completion or callback routine for a function that you call asynchronously, you must load your own code resource into memory and lock it. Note, however, that doing so interferes with the Finder's efficient use of memory, causing problems for the user.
- Before changing anything, always save the state of the system, including the graphics state, resource chain, and current file, and restore them before returning to the CE or calling any CE callback routines.

Data Types

The routines in an AOCE template code resource use the data types described in this section.

Target Specifier

Many routines in an AOCE template code resource refer to a specific aspect. The AOCE template target specifier specifies the aspect to which the routine applies. The target specifier is defined by the `DETTargetSpecification` structure.

```
struct DETTargetSpecification
{
      DETTargetSelector selector;   /* target selector */
      RStringPtr aspectName;        /* aspect name */
      long itemNumber;              /* sublist index number */
      PackedDSSpecPtr dsSpec;       /* DSSpec */
};
```

Field descriptions

`selector` A value that indicates whether the specified aspect is the current aspect (the one with which the code resource is associated) or some other aspect. The possible values for this field are listed following these field descriptions.

`aspectName` A pointer to the name of the aspect. You can specify `nil` for this field if the target is a main aspect and the value of the `selector` field is not `kDETSelf`. For target specifiers that the CE sends to your code resource, however, this field is always filled in if the target is an aspect, even if it's a main aspect. If you receive a target specifier with a `nil` in this field, the target is not an aspect (it might be a template, for example).

`itemNumber` If the value of the `selector` field is `kDETSublistItem`, then the `itemNumber` field contains the index number of an item in the current aspect's sublist. Item numbers start with 1. If the `selector` field is set to `kDETSelectedSublistItem`, then the index number counts only items in the sublist that the user has selected. If the `selector` field is set to `kDETAspectTemplate`, then the target is the aspect template indexed by the `itemNumber` field (the CE assigns an index number to every template that it loads into memory). If the `selector` field is set to `kDETInfoPageTemplate`, the target is the information page template indexed by the `itemNumber` field. If the `selector` field is set to any other value, the CE ignores this field.

`dsSpec` A pointer to a `DSSpec` structure. If the `selector` field is set to `kDETDSSpec`, then the `dsSpec` field indicates the target item. If the `selector` field is set to any other value, the CE ignores this field.

```
enum DETTargetSelector {
   kDETSelf = 0,              /* the current item */
   kDETSelfOtherAspect,       /* another aspect of the current item */
   kDETParent,                /* the parent of the current item */
   kDETSublistItem,           /* the ith item in the sublist */
   kDETSelectedSublistItem,   /* the ith selected item in the sublist */
   kDETDSSpec,                /* DSSpec */
   kDETAspectTemplate,        /* specific aspect template */
```

```
   kDETInfoPageTemplate,      /* specific info-page template */
   kDETHighSelector = 0xF000  /* force type to be short */
};

typedef enum DETTargetSelector DETTargetSelector;
```

Constant descriptions

`kDETSelf` The target aspect is the current one; that is, the aspect that originated the call to the code resource. The CE ignores all fields other than the `selector` field. When the CE calls your code resource to handle a targeted event, it sets the target selector type to `kDETSelf`. If your code resource doesn't handle the event and the aspect is an attribute, the CE calls the aspect's parent record and sets the selector type to `kDETSublistItem`.

`kDETSelfOtherAspect` The target is another aspect of the record or attribute to which the current aspect applies. The `aspectName` field points to the name of the target aspect.

`kDETParent` The target is an aspect of the object in whose sublist the current object resides. That is, the current aspect is for an attribute, and the target aspect is an aspect of the record that contains that attribute. The `aspectName` field points to the name of the target aspect, which can be any aspect of the parent.

`kDETSublistItem` The target is an aspect of an item in the sublist of the current aspect. The `itemNumber` field contains the index number of the item in the sublist. Index numbers start with 1. The `aspectName` field points to the name of the target aspect. When you call a routine provided by the CE, you can set the `aspectName` field to `nil` to target the main aspect. This selector type is useful for iterating through all of the items in a sublist. When the CE calls your code resource to handle a targeted event, it sets the target selector type to `kDETSelf`. If your code resource doesn't handle the event, the CE calls the aspect's parent and sets the selector type to `kDETSublistItem`.

`kDETSelectedSublistItem` The target is an aspect of an item in the sublist of the current aspect. The `itemNumber` field contains the index number of the item in the sublist, counting only the items the user has selected. Index numbers start with 1. The `aspectName` field points to the name of the target aspect. When you call a routine provided by the CE, you can set the `aspectName` field to `nil` to target the main aspect. This selector type is useful for iterating through all of the items that the user has selected in a sublist.

`kDETDSSpec` The target is the item specified by the `dsSpec` field. You must wait until the `kDETPastFirstLookup` metaproperty changes to 1 before you can target a catalog object. Metaproperties are listed in Table 5-3 on page 5-86.

`kDETAspectTemplate`
: The target is the aspect template indexed by the `itemNumber` field. The CE assigns an index number to every aspect template that it loads into memory. You can use this target selector only with the callback routines `kDETcmdGetResource` (page 5-207) and `kDETcmdGetTemplateFSSpec` (page 5-206).

`kDETInfoPageTemplate`
: The target is the information page template indexed by the `itemNumber` field. The CE assigns an index number to every information page template that it loads into memory. You can use this target selector only with the callback routines `kDETcmdGetResource` (page 5-207) and `kDETcmdGetTemplateFSSpec` (page 5-206).

Forwarder List

Your `kDETcmdDynamicForwarders` code-resource routine (page 5-155) returns a linked list of forwarder items, each of which contains the same information as a forwarder template (see "Components of Forwarder Templates" beginning on page 5-138). A forwarder item is defined by the `DETForwarderListItem` structure.

```
struct DETForwarderListItem {
   struct DETForwarderListItem** next;/* handle to next item, or nil */
   AttributeTag attributeValueTag;    /* attribute value tag (0 for none) */
   PackedPathName rstrs;              /* forwarder list */
};
```

The `rstrs` field is a list of packed `RString` structures in the format defined by the `PackedPathName` data type. This field contains the record type (an empty, or zero-length, string if none), the attribute type (empty if none), and a list of template names to forward to. The `PackedPathName` data type and functions for working with `PackedPathName` and `RString` structures are defined in the chapter "AOCE Utilities" in this book.

Call Block Headers

When the Catalogs Extension calls your code resource, it passes it a pointer to an AOCE template call block. The call block indicates which event occurred and includes additional parameters specific to each type of event. Every call block starts with the same fields, described here. The fields specific to each event are listed and described with the description of the code-resource routine that you must provide to handle the event. See "Functions You Can Provide as Part of Your Code Resource" beginning on page 5-148 for these descriptions.

There are three headers for call blocks: the AOCE template call block header, the AOCE template call block targeted header, and the AOCE template call block property header. These headers all have several fields in common. All of the fields are described in this section following the header definitions.

```
#define DETCallBlockHeader \
   DETCallFunctions reqFunction; /* requested function */\
   DETCallBack callBack;         /* pointer to callback routine */\
   long callBackPrivate;         /* private data for the callback routine */\
   long templatePrivate;         /* private data stored in template */

#define DETCallBlockTargetedHeader \
   DETCallFunctions reqFunction; /* requested function */\
   DETCallBack callBack;         /* pointer to callback routine */\
   long callBackPrivate;         /* private data for the callback routine */\
   long templatePrivate;         /* private data stored in template */\
   long instancePrivate;         /* private data stored in aspect */\
   DETTargetSpecification target;/* the target (originator) of the call */\
   Boolean targetIsMainAspect;   /* true if the target is the main aspect */

#define DETCallBlockPropertyHeader \
   DETCallFunctions reqFunction; /* requested function */\
   DETCallBack callBack;         /* pointer to callback routine */\
   long callBackPrivate;         /* private data for the callback routine */\
   long templatePrivate;         /* private data stored in template */\
   long instancePrivate;         /* private data stored in aspect */\
   DETTargetSpecification target;/* the target (originator) of the call */\
   Boolean targetIsMainAspect;   /* true if the target is the main aspect */\
   short property;               /* the property number the call refers to */
```

Field descriptions

`reqFunction` A routine selector that tells you which of your code resource routines to execute. For a list of the routine selectors and a description of the routines, see "Functions You Can Provide as Part of Your Code Resource" beginning on page 5-148.

`callBack` A pointer to the CE's entry point for CE routines that you can call from your code resource. If you want to call one of the CE's callback routines, pass the parameters described with that routine to the routine at the address in this field. You can use the `CallBackDET` macro described on page 5-197 for this purpose. The available AOCE template callback routines are described in "CE-Provided Functions That Your Code Resource Can Call" starting on page 5-196.

`callBackPrivate` Reserved.

`templatePrivate`
: Private storage for use by the code resource. You provide a value for this field when the code resource first calls your `kDETcmdInit` routine. The CE saves this value until you execute your `kDETcmdExit` routine and includes it in the parameter block every time it calls the code for any aspect created from this aspect template. Your code resource can change this value at any time.

`instancePrivate`
: Private storage for use by the code resource. The CE maintains a separate `instancePrivate` field for each instance of an aspect template; that is, for each aspect. You provide a value for this field when the code resource first calls your `kDETcmdInstanceInit` routine. The CE saves this value until you execute your `kDETcmdInstanceExit` routine and includes it in the parameter block every time it calls the code for this aspect. Your code resource can change this value at any time.

`target`
: A target specifier structure indicating which aspect was the original target of the event. For example, if the CE calls the code resource for an attribute and that code resource doesn't handle the event, the CE calls the code resource for the record that contains the attribute. In that case, the target specifier identifies the attribute that was called initially. See "Target Specifier" on page 5-142.

`targetIsMainAspect`
: A Boolean value that indicates whether the target is a main aspect (`true`) or not (`false`).

`property`
: The property number of the property the routine refers to.

Callback Block Headers

When your code resource calls a function supplied by the Catalogs Extension, the code resource passes a pointer to an AOCE template callback block. The callback block indicates which routine it wants the CE to execute and includes additional parameters specific to each type of routine. Every callback block starts with the same fields, described here. The fields specific to each routine are listed and described with the description of the callback routine. See "CE-Provided Functions That Your Code Resource Can Call" beginning on page 5-196 for these descriptions.

There are three headers for callback blocks: the AOCE template callback block header, the AOCE template callback block targeted header, and the AOCE template callback block property header.

```
#define DETCallBackBlockHeader \
   DETCallBackFunctions reqFunction;   /* requested function */
```

```
#define DETCallBackBlockTargetedHeader \
   DETCallBackFunctions reqFunction;   /* requested function */\
   DETTargetSpecification target;      /* the target for the request */

#define DETCallBackBlockPropertyHeader \
   DETCallBackFunctions reqFunction;   /* requested function */\
   DETTargetSpecification target;      /* the target for the request */\
   short property;                     /* the property to apply the
                                          request to */
```

Functions You Can Provide as Part of Your Code Resource

The AOCE Catalogs Extension calls your code resource when certain events occur, such as a change in an attribute value or a mouse-down event in a custom view. Your code resource must be reentrant. The CE might call the routines in your code resource at any time and in any order (except for a few routines, such as your initialization and exit routines, as indicated in this chapter).

If your code resource does not handle an event, it must return the `kDETDidNotHandle` result code. If it successfully handles the event, your code resource should return the `noErr` result code. You can return any negative number as a result code to indicate an error.

If an attribute does not have a code resource, or your code resource for the attribute doesn't handle an event, the CE calls the code resource (if any) in the aspect for the record that is the parent of the code resource it called originally.

The CE passes to your code resource a pointer to a parameter block. The fields of the parameter block are described in the preceding section and in the individual routine descriptions in this section. The function prototype for your code resource's main routine is

```
pascal OSErr MyCode(DETCallBlockPtr callBlockPtr);
```

The `DETCallBlock` structure is a union of all the parameter blocks for the code resource routines. The routine selector is specified by the `reqFunction` field of the parameter block header. You can read this field as follows:

```
callBlockPtr->protoCall.reqFunction
```

IMPORTANT

The CE does not save your code resource's A5 world. You cannot use application global variables in your code resource. s

Call-For Mask

Most code resources do not need to respond to the majority of events for which the Catalogs Extension can call your code resource. To avoid being called unnecessarily, each template's code resource has a "call-for" mask that indicates the events for which it should be invoked. Your code-resource initialization routine must return the call-for mask when it is called for initialization. In addition, your code resource can use the `kDETcmdChangeCallFors` callback routine (page 5-198) to change the call-for mask.

Not every possible event has a corresponding bit in the call-for mask. There are two classes of events excepted from the call-for mask: events that occur very infrequently (such as initialization), and events that occur only because the template specifically caused them (for instance, by including a custom view in an information page view list). Your code resource is always called for all such events unless you specify a value of `kDETCallForNothing` for your call-for mask. To be called *only* for such events, specify `kDETCallForOther`.

A parent object is not given calls that its children failed to handle unless the `kDETCallForEscalation` bit is set in the call-for mask.

```
/* Call-for list: */

#define kDETCallForOther         1     /* call for events not listed below */
#define kDETCallForIdle          2     /* kDETcmdIdle */
#define kDETCallForCommands      4     /* kDETcmdPropertyCommand,
                                          kDETcmdSelfOpen */
#define kDETCallForViewChanges   8     /* kDETcmdViewListChanged,
                                          kDETcmdPropertyDirtied,
                                          kDETcmdMaximumTextLength */
#define kDETCallForDrops         0x10  /* kDETcmdDropQuery,
                                          kDETcmdDropMeQuery */
#define kDETCallForAttributes    0x20  /* kDETcmdAttributeCreation,
                                          kDETcmdAttributeNew,
                                          kDETcmdAttributeChange,
                                          kDETcmdAttributeDelete */
#define kDETCallForValidation    0x40  /* kDETcmdValidateSave */
#define kDETCallForKeyPresses    0x80  /* kDETcmdKeyPress and
                                          kDETcmdPaste */
#define kDETCallForSyncing       0x200 /* kDETcmdShouldSync, kDETcmdDoSync */
#define kDETCallForResources     0x100 /* kDETcmdDynamicResource */
#define kDETCallForEscalation    0x8000/* all calls escalated to the
                                          next level */

#define kDETCallForNothing       0     /* do not call */
#define kDETCallForEverything    0xFFFFFFFF  /* all of the above */
```

Initializing and Removing Templates

The Catalogs Extension calls the code resource routines in this section when it loads aspect templates into memory (`kDETcmdInit`), creates aspects (`kDETcmdInstanceInit`), creates attributes or records (`kDETcmdItemNew`), removes an aspect template from memory (`kDETcmdExit`), and removes an aspect from memory (`kDETcmdInstanceExit`).

kDETcmdInit

The CE calls your code resource with this routine selector when the CE first loads the template.

```
struct DETInitBlock {
            DETCallBlockHeader
            long newCallFors;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdInit`
↔	`templatePrivate`	`long`	Data stored in template
→	`callBack`	`DETCallBack`	Callback pointer
←	`newCallFors`	`long`	Call-for list

DESCRIPTION

The Catalogs Extension calls your code resource with the `kDETcmdInit` routine selector only when the Finder loads your aspect template during template initialization (such as during system initialization or the first time the CE needs a template after you have called the `kDETcmdUnloadTemplates` callback routine). You should use this opportunity to initialize the call-for mask for your template and to allocate any memory your template needs. Return the call-for mask in the `newCallFors` field. Place a pointer to your template's data in the `templatePrivate` field of the parameter block.

You can call the CE callback routines `kDETcmdGetTemplateFSSpec`, `kDETcmdBeep`, or `kDETcmdAboutToTalk`. You should use the routine `kDETcmdAboutToTalk` only if you need to report a problem to the user.

Return the `noErr` result code if you return a new call-for list. If you set the call-for mask to `kDETCallForEverything`, return the `kDETDidNotHandle` result code. If for some reason you do not want the template to be loaded (for example, if you cannot allocate the memory you need), return an error code.

SPECIAL CONSIDERATIONS

Because the CE has not yet created any aspects, you cannot call any targeted callback routines from your initialization routine.

Because the CE might not have loaded all main aspect templates, the Standard Catalog Package does not yet have information on the icons, record types, and record categories available. Therefore, your initialization routine cannot call the Standard Catalog Package functions `SDPGetIconByType`, `SDPGetDSSpecIcon`, `SDPGetCategories`, and `SDPGetCategoryTypes`.

Because the Collaboration toolbox might not yet be available, do not call any Collaboration toolbox functions unless you have used the Gestalt Manager to check for its availability.

CALL-FOR MASK VALUE

None

SEE ALSO

Call-for masks are described in “Call-For Mask” on page 5-149.

All Standard Catalog Package functions are described in the chapter “Standard Catalog Package” in this book.

kDETcmdExit

The CE calls your code resource with this routine selector before it removes the template.

```
struct DETExitBlock{
            DETCallBlockHeader
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdExit`
→	`templatePrivate`	`long`	Data stored in template
→	`callBack`	`DETCallBack`	Callback pointer

DESCRIPTION

The Catalogs Extension calls your exit routine just before it removes your template. The CE removes templates when the system shuts down or when you call the `kDETcmdUnloadTemplates` callback routine. Your exit routine should free any memory you allocated. You can call the CE callback routines `kDETcmdGetTemplateFSSpec`, `kDETcmdBeep`, or `kDETcmdAboutToTalk`. You

should use the routine `kDETcmdAboutToTalk` only if you need to report a problem to the user.

Your exit routine should return the `noErr` or `kDETDidNotHandle` result code or a specific error code.

SPECIAL CONSIDERATIONS

Because the AOCE toolbox might have already been shut down, your exit routine should not call any AOCE functions.

CALL-FOR MASK VALUE

None

kDETcmdInstanceInit

The CE calls your code resource with this routine selector when it creates an aspect from your aspect template.

```
struct DETInstanceInitBlock {
            DETCallBlockTargetedHeader
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdInstanceInit`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?

DESCRIPTION

The Catalogs Extension calls your instance-initialization routine once each time it creates an aspect from your aspect template. You should allocate any memory needed by this aspect and place a pointer to the aspect's data in the `instancePrivate` field of the parameter block.

If your routine returns an error (any negative result code), the CE disables your code resource and does not call it again for this aspect. In all other respects the aspect continues to function normally, and the CE can still call your code resource for other aspects for the same template.

If your routine returns either the `noErr` or `kDETDidNotHandle` result codes, the CE processes the aspect normally.

The CE can remove this aspect from memory at any time that the aspect is not in use and the Finder needs the memory. In that case, the CE calls your `kDETcmdInstanceExit` routine. The CE will then call your `kDETcmdInstanceInit` routine again whenever it needs the aspect, such as when the user opens a record or causes a catalog folder window to redraw.

CALL-FOR MASK VALUE

None

SEE ALSO

The `kDETcmdInstanceExit` routine is described on page 5-154.

kDETcmdItemNew

The CE calls your code resource with this routine selector when it creates a new record or attribute.

```
struct DETItemNewBlock{
            DETCallBlockTargetedHeader
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdItemNew`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?

DESCRIPTION

After the Catalogs Extension creates an aspect and calls your `kDETcmdInstanceInit` routine, the CE calls your new item routine each time it creates a new record or attribute. You can use this opportunity to specify initial values for attributes or perform other actions appropriate to a new attribute or record of the type supported by this aspect.

CALL-FOR MASK VALUE

None

SEE ALSO

The `kDETcmdInstanceInit` routine is described on page 5-152.

kDETcmdInstanceExit

The CE calls your code resource with this routine selector before it removes an aspect from memory.

```
struct DETInstanceExitBlock {
            DETCallBlockTargetedHeader
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdInstanceExit
↔	templatePrivate	long	Data stored in template
→	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?

DESCRIPTION

The Catalogs Extension can remove an aspect from memory at any time the aspect is not in use and the Finder needs additional memory. Your instance exit routine should release any memory allocated by the `kDETcmdInstanceInit` routine for this aspect. The CE ignores the result code your instance exit routine returns.

CALL-FOR MASK VALUE

None

SEE ALSO

The `kDETcmdInstanceInit` routine is described on page 5-152.

Dynamic Creation of Resources

Your code resource can extend the use of your templates much as a forwarder template does. Your code resource can also substitute resources for those in the template file. Because the Catalogs Extension loads resources as needed, it can call your code resource at any time for this purpose. The CE calls the code resource routines described in this section to achieve these ends.

kDETcmdDynamicForwarders

The CE calls your code resource with this routine selector to allow you to apply the template to additional record or attribute types.

```
struct DETDynamicForwardersBlock {
            DETCallBlockHeader
            DETForwarderListHandle forwarders;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdDynamicForwarders
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
←	forwarders	DETForwarderListHandle	List of forwarders

DESCRIPTION

When the Catalogs Extension is loading your template, after it calls your `kDETcmdInit` routine, it calls your `kDETcmdDynamicForwarders` routine to allow you to add record or attribute types to those to which the template applies. The `forwarders` field is a handle to a linked list of elements of type `DETForwarderListItem`. Each contains the same information as is found in a forwarder template: a record type, attribute type, or both; an attribute value tag (0 for none); and a list of template names (including both aspect and information page templates).

Your `kDETcmdDynamicForwarders` routine must allocate the handles containing the returned data, but the CE disposes of them when done.

If your routine returns `kDETDidNotHandle` or an error, the CE does not process the forwarders list.

CALL-FOR MASK VALUE

None

SEE ALSO

The `DETForwarderListItem` structure is defined in “Forwarder List” on page 5-145.

The forwarder template is described in “Components of Forwarder Templates” beginning on page 5-138.

kDETcmdDynamicResource

The CE calls your code resource with this routine selector when it is about to load a resource from your template file to give you the opportunity to substitute a different resource.

```
struct DETDynamicResourceBlock {
            DETCallBlockTargetedHeader
            ResType resourceType;
            short propertyNumber;
            short resourceID;
            Handle theResource;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdDynamicResource
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	resourceType	ResType	Type of resource being requested
→	propertyNumber	short	Property number of requested resource
→	resourceID	short	Resource ID
←	theResource	Handle	Replacement resource

DESCRIPTION

Before the Catalogs Extension loads a resource from your template file, it calls your `kDETcmdDynamicResource` routine to give you the opportunity to substitute a different resource for the one in the file. The CE calls this routine for any resource except the aspect template signature resource (`'deta'`) and those used by a forwarder template or the `kDETcmdDynamicForwarders` routine (the attribute value tag, attribute type, and record type resources; see Table 5-11 on page 5-138.)

The `resourceType` field contains the type of resource required. The `propertyNumber` field contains the property number of the resource, and the `resourceID` field contains the resource ID of the resource (that is, the base template resource ID plus the property number).

If you want to substitute a different resource for the one in the template file, return a handle to the new resource in the `theResource` field. You must allocate the handle; the CE disposes of it when finished with it.

If your routine returns `kDETDidNotHandle` or an error, then the CE uses the resource from the template file.

SPECIAL CONSIDERATIONS

Do not allocate a resource handle for the `theResource` field; you must own this handle.

Because the CE calls your `kDETcmdDynamicResource` routine every time it loads a resource, you should include such a routine only if you have a specific reason to do so. Otherwise, use the call-for mask to avoid having the CE call your code resource for resource loading.

CALL-FOR MASK VALUE

`kDETCallForResources`

SEE ALSO

Before loading resources from a forwarder template file, the CE calls your `kDETcmdDynamicForwarders` routine (page 5-155).

Processing Idle-Time Tasks

When an information page that uses your aspect template is the frontmost window, the CE calls your code resource periodically with the `kDETcmdIdle` routine selector.

kDETcmdIdle

The CE calls your code resource with this routine selector periodically during idle times.

```
struct DETInstanceIdleBlock {
            DETCallBlockTargetedHeader
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdIdle`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?

DESCRIPTION

The Catalogs Extension calls your code resource with this routine selector during idle times when an information page that uses your aspect template is the Finder's frontmost window and the Finder is the frontmost application. An aspect code resource cannot perform an idle-time task unless its window is frontmost.

The CE ignores the result code returned by this routine. Therefore, the CE does not call the parent record's code resource when the aspect for an attribute returns `kDETDidNotHandle`.

CALL-FOR MASK VALUE

`kDETCallForIdle`

Property and Information Page Functions

The routines in this section interact directly with an information page. The Catalogs Extension calls your `kDETcmdOpenSelf` routine, described next, to give you the opportunity to override the standard behavior when the user opens an information page. Your `kDETcmdPropertyCommand` routine (page 5-159) processes a command sent by an information page property, such as a button or menu item. Your `kDETcmdKeyPress` (page 5-163) and `kDETcmdPaste` (page 5-164) routines handle keypresses and paste operations that occur when the user is using your information page.

Your `kDETcmdMaximumTextLength` routine (page 5-166) specifies the maximum permitted length for a text string in an information page.

The CE calls your `kDETcmdViewListChanged` routine (page 5-166) when the list of enabled views has changed in an information page. The CE calls your `kDETcmdPropertyDirtied` routine (page 5-167) to give you an opportunity to update the information page display when a property value changes. The CE calls your `kDETcmdValidateSave` routine (page 5-168) when the CE is about to save property values.

kDETcmdOpenSelf

The CE calls your code resource with this routine selector before it opens an information page to give you the opportunity to override the normal behavior.

```
struct DETOpenSelfBlock {
            DETCallBlockTargetedHeader
            short modifiers;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdOpenSelf
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	modifiers	short	Modifier keys

DESCRIPTION

When a user attempts to open a catalog object for which you provided the main aspect, the Catalogs Extension calls the code resource of that main aspect with the `kDETcmdOpenSelf` routine selector before opening the information page. You can use this opportunity to do something other than opening the information page or to set default values for the information page before it opens.

Because the target is always a main aspect when you receive this routine selector, the value of `targetIsMainAspect` is always `true`.

If your routine returns the `kDETDidNotHandle` result code, the CE opens the information page normally. If it returns `noErr`, the CE does not open the information page. If it returns an error, the CE displays a Finder error dialog box and does not open the information page.

CALL-FOR MASK VALUE

`kDETCallForCommands`

kDETcmdPropertyCommand

The CE calls your code resource with this routine selector when the user takes certain actions in an information page.

```
struct DETPropertyCommandBlock {
            DETCallBlockPropertyHeader
            long parameter;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdPropertyCommand
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	property	short	Property number
↔	parameter	long	Command parameter

DESCRIPTION

The Catalogs Extension calls your property-command routine when the user clicks a button or checkbox in an information page, selects an item in a pop-up menu, or clicks on static command text. The CE also calls your property-command routine when you return a property number in response to a drop operation that affects your aspect or when you call the `kDETcmdDoPropertyCommand` callback command.

When it calls your property command, the CE always includes a value in the `property` field of the parameter block. This is always a number you have supplied; either in the command field of the `'detv'` resource for the view that originated the property command, or as a parameter to your `kDETcmdDropQuery` routine or the `kDETcmdDoPropertyCommand` callback routine. Most commonly, you use the property number of the view as the value of the command field of the `'detv'` resource so that your code resource can tell which view sent the property command. Some property commands, such as those related to drop operations, do not originate from views. In this case, you use the value of the `property` field of the parameter block as a routine selector rather than as a property number.

Some property commands include a parameter in the `parameter` field of the parameter block; for example, if the user selects an item in a pop-up menu, the CE includes the number of the menu item in the `parameter` field of the parameter block when it calls your property-command routine. Table 5-14 shows the various property commands that you can handle in your code resource and describes the origin of the value in the `property` and `parameter` fields of the parameter block for each type of property command.If your routine returns a result code of `noErr`, the CE assumes that you have handled the command and takes no further action. If your routine returns a result code of `kDETDidNotHandle`, the CE calls the code resource of the parent of the object whose code resource was called originally (that is, if the code resource was for an attribute, the CE calls the code resource, if any, of the record that contains that attribute). If your routine returns a value in the `parameter` field when it returns a result code of `kDETDidNotHandle` for radio buttons, checkboxes, and pop-up menus, the CE sets the value of the property equal to the value in the `parameter` field.

If your routine returns an error code (any nonzero result code), the CE displays a dialog box specifying the error and leaves the value of the property unchanged.

Table 5-14 Property commands

Source of command	`property` field of parameter block	`parameter` field of parameter block
Button	Value in the property-command field of the `'detv'` resource for the button; this must equal the property number of the button.	Not used.
	NOTE If your routine returns `kDETDidNotHandle`, the CE ignores the button click. If you use the property numbers `kDETAddNewItem`, `kDETRemoveSelectedItems`, or `kDETOpenSelectedItems`, the CE handles the command without calling your code resource (see Table 5-3 on page 5-86).	
Default button	Value in the property-command field of the `'detv'` resource for the button; this must equal the property number of the button.	Not used.
	NOTE If your routine returns `kDETDidNotHandle`, the CE ignores the button click.	
Radio button	Value in the property-command field of the `'detv'` resource for the button; this must equal the property number of the button.	Value in the command-parameter field of the `'detv'` resource for the button. Each button in a set of radio buttons must have a distinct value.
	NOTE The button displayed as "on" is the one for which the command-parameter field is equal to the value of the property. If your property-command routine returns `kDETDidNotHandle`, the CE sets the value of the property equal to the value in the `parameter` field of the parameter block. Therefore, if you do not alter the value in the `parameter` field, the button the user clicked is displayed as "on." If you do handle the radio-button command yourself, you must call the `kDETcmdDirtyProperty` callback routine (page 5-233) to force the CE to redraw the radio button.	
Checkbox	Value in the property-command field of the `'detv'` resource for the checkbox; this must equal the property number of the checkbox.	Set by the CE to the opposite of the current property value (that is, 1 if the property value is 0, or 0 if the property value is 1).
	NOTE The property value for a checkbox can be equal to 0 or 1; the checkbox is off if this value is 0 and on if 1. If your code resource returns `kDETDidNotHandle`, the CE sets the property value to equal the value in the `parameter` field, toggling the checkbox off or on. If you do handle the checkbox command yourself, you must call the `kDETcmdDirtyProperty` callback routine (page 5-233) to force the CE to redraw the checkbox.	

continued

Table 5-14 Property commands (continued)

Source of command	`property` field of parameter block	`parameter` field of parameter block
Pop-up menu	Value in the property-command field of the `'detv'` resource for the menu; this must equal the property number of the menu.	Value in the command-ID field of the `'fmnu'` resource that defines the menu. There is a distinct command-ID value for each menu item.
	NOTE Your code resource can use the `command` parameter to determine which item in the pop-up menu the user has chosen.	
Static command text from view	Value in the property-command field of the `'detv'` resource for the view.	Value in the command-parameter field of the `'detv'` resource for the view.
	NOTE If you use the value `kDETChangeViewCommand` for the property-command field of the `'detv'` resource, the CE uses this property command to sort a sublist and does not call your code resource.	
Drop-operation command	Value you specified in the `commandID` parameter to your `kDETcmdDropQuery` routine. Treat this value as a routine selector to determine what course of action to take.	The location of the cursor when the mouse button was released, in global coordinates, as two shorts in the order x, y.
	NOTE When the user drops one or more objects on a catalog object for which you have provided an aspect template, the CE calls your `kDETcmdDropQuery` routine once for each item dropped. If your routine returns a property number, the CE calls your property-command routine. The CE combines all of the drop operations that return the same property number and calls your property command only once. You can then call the `kDETcmdGetCommandSelectionCount` callback routine to determine how many objects are being dropped and the `kDETcmdGetCommandItemN` callback routine to determine the nature of each object being dropped.	
Drop-me operation command	Value you specified in the `commandID` parameter to your `kDETcmdDropMeQuery` routine. Treat this value as a routine selector to determine what course of action to take.	The location of the cursor when the mouse button was released, in global coordinates, as two shorts in the order x, y.
	NOTE When the user drags and drops a catalog object for which you have provided an aspect template, the CE calls your `kDETcmdDropMeQuery` routine. If your routine returns a property number, the CE calls your property-command routine. You can then call the `kDETcmdGetCommandItemN` callback routine to determine the nature of the object upon which the item is being dropped.	

Table 5-14 Property commands (continued)

Source of command	`property` field of parameter block	`parameter` field of parameter block
Do-property-command callback	Value you specified in the `property` parameter to the `kDETcmdDoPropertyCommand` callback routine.	Value you specified in the `parameter` parameter to the `kDETcmdDoPropertyCommand` callback routine.

NOTE The `kDETcmdDoPropertyCommand` callback routine allows your code resource to send a property command to any code resource you can target.

CALL-FOR MASK VALUE

None

SEE ALSO

To force the CE to redraw a view after you handle a property event, use the `kDETcmdDirtyProperty` callback routine (page 5-233).

The aspect's `kDETAspectViewMenu` resource is described on page 5-103.

View types are described on page 5-127.

The `kDETcmdDropMeQuery` routine is described on page 5-170. The `kDETcmdDropQuery` routine is described on page 5-172.

Use the `kDETcmdGetCommandSelectionCount` callback routine (page 5-201) to determine how many objects are being dropped and the `kDETcmdGetCommandItemN` callback routine (page 5-202) to determine the nature of each object dropped.

To initiate a property command from a code resource, use the `kDETcmdDoPropertyCommand` routine (page 5-245).

kDETcmdKeyPress

The CE calls your code resource with this routine selector when the user presses a key while using an information page.

```
struct DETKeyPressBlock {
            DETCallBlockPropertyHeader
            EventRecord *theEvent;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdKeyPress
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	property	short	Property number
→	theEvent	EventRecord	The event record for the keypress

DESCRIPTION

You can use your `kDETcmdKeyPress` routine to respond to a keypress that occurs while the user is using your information page. If the user is editing a text view, the `property` field identifies the view. If the cursor is not in a text view, the `property` field contains the value `kDETNoProperty`. The Catalogs Extension does not call your `kDETcmdKeyPress` routine for Command-key keypress combinations.

If your routine returns `kDETDidNotHandle`, the CE handles the keypress. If your routine returns an error, the CE displays an error dialog box. If your routine returns `noErr`, the CE assumes you handled the keypress and does no further processing.

You can use this routine, for example, to prevent the user from entering certain characters in a text field.

CALL-FOR MASK VALUE

None

SEE ALSO

To control what a user pastes into your information page, use the `kDETcmdPaste` routine, described next.

kDETcmdPaste

The CE calls your code resource with this routine selector when the user attempts to paste text while using your information page.

```
struct DETPasteBlock {
            DETCallBlockPropertyHeader
            short modifiers;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdPaste
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	property	short	Property number
→	modifiers	short	Modifier keys at time of paste

DESCRIPTION

The Catalogs Extension calls your `kDETcmdPaste` routine when the user chooses Paste from the Edit menu (or presses Command-V when Paste is enabled) while the user is using your information page. If the user is editing a text view, the `property` field identifies the view. If the cursor is not in a text view, the `property` field contains the value `kDETNoProperty`.

If your routine returns `kDETDidNotHandle`, the CE handles the paste. If your routine returns an error, the CE displays an error dialog box. If your routine returns `noErr`, the CE assumes you handled the paste and does no further processing.

You can use this routine, for example, to prevent the user from pasting certain characters in a text field.

CALL-FOR MASK VALUE

None

SEE ALSO

To determine what the user is attempting to paste you must read the data in the scrap. For information on how to read the scrap, see the chapter “Scrap Manager” in *Inside Macintosh: More Macintosh Toolbox*.

kDETcmdMaximumTextLength

The CE calls your code resource with this routine selector to determine the maximum permitted length of a property that is displayed in an editable text view.

```
struct DETMaximumTextLengthBlock {
            DETCallBlockPropertyHeader
            long maxSize;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdMaximumTextLength
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	property	short	Property number
←	maxSize	long	Maximum text length

DESCRIPTION

If the user tries to type more into an editable text view than the maximum you specify with your `kDETcmdMaximumTextLength` routine, the Catalogs Extension displays a dialog box informing the user that the text has reached its maximum length. The maximum size you can specify in the `maxSize` field is 255 bytes. When counting the length of a text string for this routine, count the first byte as 1, not as 0. If your routine returns an error or a result code of `kDETDidNotHandle`, the CE limits the text string to 255 bytes. If your routine returns `noErr`, the CE limits the text string to the length you specify.

CALL-FOR MASK VALUE

`kDETCallForViewChanges`

kDETcmdViewListChanged

The CE calls your code resource with this routine selector when the list of enabled views has changed in one of the information pages associated with this aspect.

```
struct DETViewListChangedBlock {
            DETCallBlockTargetedHeader
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdViewListChanged
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?

DESCRIPTION

The list of enabled views in an information page changes when the Catalogs Extension displays a conditional view. The list also changes when the page is first opened, either because the record has just been opened or because the user has used the pop-up menu to select a different information page.

CALL-FOR MASK VALUE

kDETCallForViewChanges

SEE ALSO

Conditional views are described in “Conditional Views” on page 5-26 and in “Information Page Template Signature Resource” beginning on page 5-121.

kDETcmdPropertyDirtied

The CE calls your code resource with this routine selector when you call the kDETcmdDirtyProperty callback routine and when the kDETPastFirstLookup metaproperty changes.

```
struct DETPropertyDirtiedBlock {
            DETCallBlockPropertyHeader
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdPropertyDirtied
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	property	short	Property number

DESCRIPTION

The Catalogs Extension calls your `kDETcmdPropertyDirtied` 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 and the `kDETPastFirstLookup` metaproperty changes to 1. Although the CE updates the display when you call the `kDETcmdDirtyProperty` callback routine and when it completes a catalog search, you might want to redraw other property views that are dependent on the one that changed initially. Also, if your routine returns the `kDETDidNotHandle` result code when the CE calls your code resource for an attribute with the `kDETcmdPropertyDirtied` routine selector, the CE calls the code resource for the record that contains that attribute. You can use this technique to inform a parent of a change that occurred in a child. The CE ignores any other function results of this routine.

CALL-FOR MASK VALUE

`kDETCallForViewChanges`

SEE ALSO

The `kDETcmdDirtyProperty` callback routine is described on page 5-233.

Metaproperties are listed in Table 5-3 on page 5-86.

kDETcmdValidateSave

The CE calls your code resource with this routine selector when the CE is about to save the property values associated with an aspect.

```
struct DETValidateSaveBlock {
            DETCallBlockTargetedHeader
            RStringHandle errorString;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdValidateSave`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
←	`errorString`	`RStringHandle`	Handle to error string

DESCRIPTION

If you wish to allow the Catalogs Extension to save the new data entered into an information page, return a result code of `noErr` or `kDETDidNotHandle`. If you do not want the CE to save the data, return an error (a negative result code) and, in the `errorString` parameter, specify a handle to an error string telling why the information page should not be saved. You must allocate the handle to the error string; the CE deallocates it. The CE displays the error string in a dialog box to inform the user why the data could not be saved.

Normally, the CE saves new property values only when the user leaves the aspect; that is, when the user closes the information page or flips to another information page that uses a different aspect. You can call the `kDETcmdSaveProperty` command to save a property value at any time.

The CE does not call your `kDETcmdValidateSave` routine when someone changes a sublist field or the name of a stand-alone attribute, or when your code resource calls the `kDETcmdSaveProperty` command.

CALL-FOR MASK VALUE

`kDETCallForValidation`

SEE ALSO

Call the `kDETcmdSaveProperty` command (page 5-234) to save a property value.

Supporting Drops

If the standard drop-operation resources are not adequate for your needs, you can provide a code-resource routine to handle drops. You can write a `kDETcmdDropMeQuery` routine for the aspect template of the object being dropped and a `kDETcmdDropQuery` routine for the aspect template of the destination for the drop. The Catalogs Extension calls the code resource of the object being dropped first and then the code resource of the destination. Thus, the code resource of the destination can override that of the object being dropped.

Drags and drops are described in “Drags and Drops” on page 5-28 and drop-operation resources are described in “Supporting Drags and Drops” beginning on page 5-98.

kDETcmdDropMeQuery

The CE calls your code resource with this routine selector when the user attempts to drop the object to which your aspect template applies onto another object.

```
struct DETDropMeQueryBlock {
            DETCallBlockTargetedHeader
            short modifiers;
            long commandID;
            AttributeType destinationType;
            Boolean copyToHFS;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdDropMeQuery
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	modifiers	short	Modifier keys at drop time
↔	commandID	long	Command ID
↔	destinationType	AttributeType	Attribute type of new attribute
←	copyToHFS	Boolean	Copy to HFS?

DESCRIPTION

When the user drags an AOCE catalog object and drops it onto another catalog object or onto an HFS object, the Catalogs Extension calls the code resource in the aspect of the dragged object with the `kDETcmdDropMeQuery` routine selector. (If a dragged attribute's aspect does not contain a code resource or if its code resource returns the `kDETDidNotHandle` result code, the CE calls the code resource of the aspect for the record that contains that attribute.) You can call the `kDETcmdGetCommandItemN` callback routine to get information about the destination object.

The `modifiers` field indicates which modifier keys, if any, the user was pressing when the mouse button was released.

The `commandID` and `destinationType` fields contain the CE's best guess as to the correct drop action. Possible values for the `commandID` parameter are as follows:

```
#define kDETDoNothing   'xxx0'
#define kDETMove        'move'
#define kDETDrag        'drag'
#define kDETAlias       'alis'
```

Constant descriptions

`kDETDoNothing`	Do nothing. The CE has no standard behavior for a drop of this sort. For example, the user might try dragging an attribute from a sublist and dropping it onto an application.
`kDETMove`	Move the object to a new location. For example, if the user drags an attribute from one sublist to another on the same volume, the CE's default behavior is to change the location of the attribute.
`kDETDrag`	Make a copy of the object. For example, if the user drags an attribute from a sublist on one volume to a sublist on another volume, the CE copies the attribute.
`kDETAlias`	Make an alias to the object. For example, if the user drags a record from a catalog folder and drops it onto another record, the CE creates an alias to the record and places it in an attribute in the destination record.

The `destinationType` field indicates the attribute type of the attribute that the CE creates as a result of the drop if the CE copies an attribute or creates an alias to a record.

If your routine returns a result code of `kDETDidNotHandle` or an error, the CE continues to try to determine the appropriate action. If the dragged object is an attribute, the CE looks for a code resource in an aspect of the parent record. Then the CE looks for code resources in the aspects of the destination object and in the parent of the destination object, if any.

If you wish, you can set new values for the `commandID` and `destinationType` fields and return a result code of `noErr`. The CE then uses the values you set for these parameters as input to any other code resources it finds. If no other aspect or code resource overrides these values, the CE carries out the action you specified. If the CE can't carry out the specified action, it displays a dialog box describing the problem.

You can also specify a number in the developers' property-number range (that is, `kDETFirstDevProperty` through 249) for the `commandID` parameter. In this case, the CE sends a property command (`kDETcmdPropertyCommand`) with that number to the target aspect (that is, the code resource of the aspect that the CE indicated as the target when it called your code resource). Your property-command routine can then call the `kDETcmdGetCommandItemN` callback routine to determine the nature of the destination object.

If you set the `copyToHFS` parameter to `true`, the CE displays a dialog box asking the user to copy the object to the desktop (that is, to create an HFS version of the object) before performing the operation. For example, if the item is a record and you set the `copyToHFS` parameter to `true`, the CE asks the user to create an information card before performing the operation. Your property-command routine can use the `kDETcmdGetCommandItemN` callback routine to get the file system specification (`FSSpec` structure) for the information card. If you set the `copyToHFS` parameter to `true`, you must set the `commandID` parameter to a property number; otherwise, the CE ignores the `copyToHFS` parameter.

Note
In future versions of the AOCE software, the CE might create the HFS version of the object rather than requesting the user to do so. u

CALL-FOR MASK VALUE

`kDETCallForDrops`

SEE ALSO

For more information on how the CE handles drags and drops, see "Drags and Drops" on page 5-28.

If the object on which the AOCE catalog object was dropped is also a catalog object, the CE calls the destination object with the `kDETcmdDropQuery` routine selector, described next.

Your property command can use the `kDETcmdGetCommandSelectionCount` callback routine (page 5-201) to determine how many objects are being dropped.

You can use the `kDETcmdGetCommandItemN` callback routine (page 5-202) to determine the nature of the object on which the item is being dropped.

The `kDETcmdPropertyCommand` routine is described on page 5-159.

kDETcmdDropQuery

The CE calls your code resource with this routine selector when the user attempts to drop an object on the object to which your aspect template applies.

```
struct DETDropQueryBlock {
            DETCallBlockTargetedHeader
            short modifiers;
            long commandID;
            AttributeType destinationType;
            Boolean copyToHFS;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdDropQuery
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	modifiers	short	Modifier keys at drop time
↔	commandID	long	Command ID
↔	destinationType	AttributeType	Attribute type of new attribute
←	copyToHFS	Boolean	Copy to HFS?

DESCRIPTION

When the user drags an AOCE catalog object or HFS object and drops it onto a catalog object, the Catalogs Extension calls the code resource in the aspect of the destination object with the `kDETcmdDropQuery` routine selector. (If a destination attribute's aspect does not contain a code resource or if its code resource returns the `kDETDidNotHandle` result code, the CE calls the code resource of the aspect for the record that contains that attribute.) The `modifiers` parameter indicates which modifier keys, if any, the user was pressing when the mouse button was released.

If the user drops more than one object simultaneously on a destination, the CE calls the code resource for the destination's aspect once for each object dropped. You can call the `kDETcmdGetCommandItemN` callback routine to get information about the item being dropped.

The `commandID` and `destinationType` fields contain the CE's best guess as to the correct drop action. Possible values for the `commandID` parameter are a property number or the constants `kDETDoNothing`, `kDETMove`, `kDETDrag`, or `kDETAlias` (see page 5-171). Note that because the CE calls any code resource for the dragged object with the `kDETcmdDropMeQuery` routine selector before calling the code resource for the destination object, the values in the `commandID` and `destinationType` fields might have been provided by another code resource rather than by the CE itself.

The `destinationType` field indicates the attribute type of the attribute that the CE creates as a result of the drop if the CE copies an attribute or creates an alias to a record.

If your routine returns a result code of `kDETDidNotHandle`, the CE continues to try to determine the appropriate action. If the destination object is an attribute, the CE looks for a code resource in an aspect of the parent record.

If you wish, you can set new values for the `commandID` and `destinationType` fields and return a result code of `noErr`. The CE then uses the values you set for these parameters as input to any other code resources it finds. If no other aspect or code resource overrides these values, the CE carries out the action you specified. If the CE can't carry out the specified action, it displays a dialog box describing the problem.

You can also specify a number in the developers' property-number range (that is, `kDETFirstDevProperty` through 249) for the `commandID` parameter. The CE then sends a property command (`kDETcmdPropertyCommand`) with that number to the target aspect (that is, the code resource of the aspect that the CE indicated as the target when it called your code resource). Note that your code resource' property-command routine should treat this property number as a routine selector to determine what course of action to take. The property number you use for this purpose need not correspond to any view in the information page.

The CE combines drop operations whenever possible. Therefore, if your `kDETcmdDropQuery` routine returns the same property command for two or more dragged objects, the CE calls your code resource only once with a property command (`kDETcmdPropertyCommand`). Your property-command routine then must use the `kDETcmdGetCommandSelectionCount` and `kDETcmdGetCommandItemN` callback routines to determine which objects are being dragged and perform the appropriate action.

If you set the `copyToHFS` parameter to `true`, the CE displays a dialog box asking the user to copy the object to the desktop (that is, to create an HFS version of the object) before performing the operation. For example, if the item is a record and you set the `copyToHFS` parameter to `true`, the CE asks the user to create an information card before performing the operation. Your property-command routine can use the `kDETcmdGetCommandItemN` callback routine to get the file system specification (`FSSpec` structure) for the information card. If you set the `copyToHFS` parameter to `true`, you must set the `commandID` parameter to a property number; otherwise, the CE ignores the `copyToHFS` parameter.

CALL-FOR MASK VALUE

`kDETCallForDrops`

SEE ALSO

If the object being dragged is also a catalog object, the CE first calls the dragged object with the `kDETcmdDropMeQuery` routine selector, described on page 5-170.

Your property command can use the `kDETcmdGetCommandSelectionCount` callback routine (page 5-201) to determine how many objects are being dropped.

You can use the `kDETcmdGetCommandItemN` callback routine (page 5-202) to determine the nature of each object being dropped.

The `kDETcmdPropertyCommand` routine is described on page 5-159.

Attribute-Related Commands

The Catalogs Extension calls one of the code resource routines described in this section before the CE creates, changes, or deletes an attribute value.

kDETcmdAttributeCreation

The CE calls your code resource with this routine selector when it is about to add a new attribute value to a sublist.

```
struct DETAttributeCreationBlock {
            DETCallBlockHeader
            PackedDSSpecPtr parent;
            short refNum;
            AuthIdentity identity;
            AttributeType attrType;
            AttributeTag attrTag;
            Handle value;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdAttributeCreation
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	parent	PackedDSSpecPtr	Record in which the CE creates the new attribute
→	refNum	short	Reference number for the catalog containing the record that will contain the attribute (used only if the catalog is a personal catalog; only personal catalogs use reference numbers)
→	identity	AuthIdentity	Authentication identity used when gaining access to the parent record
↔	attrType	AttributeType	Type of attribute being created
↔	attrTag	AttributeTag	Tag of attribute being created
↔	value	Handle	Value to write (preallocated to the size of the default attribute value, if any, or to 1 if no default; resize as needed)

DESCRIPTION

When the user clicks the Add button in an information page to add a new attribute value to a sublist, the Catalogs Extension calls the code resource for the main aspect template for attributes of that type with the `kDETcmdAttributeCreation` routine selector. The `attrType`, `attrTag`, and `value` parameters indicate the default values for the new attribute type, attribute tag, and attribute value. You can return different values for any of these parameters if you wish. Whether or not you change any of these values, return

the kDETDidNotHandle result code if you want the CE to create the new attribute value. If your routine returns noErr, the CE assumes you used the Catalog Manager to create the new attribute value and does not create it for you. Likewise, if your routine returns an error, the CE does not create the attribute value.

After the CE calls your kDETcmdAttributeCreation routine and before it creates the attribute value, it calls your kDETcmdAttributeNew routine.

Note that the CE does not specify a target when it calls your kDETcmdAttributeCreation routine because the object hasn't been created yet; it always calls the code resource of the template that will be the object's main aspect template.

SPECIAL CONSIDERATIONS

You should limit sublist items to one line. Multiline sublist items are not guaranteed to work correctly.

CALL-FOR MASK VALUE

kDETCallForAttributes

SEE ALSO

When the CE is about to create a new attribute value, whether in a sublist or not, it calls the kDETcmdAttributeNew routine, described next.

You can use the Catalog Manager's DirAddAttributeValue function to add an attribute value; see the chapter "Catalog Manager" in this book for details.

kDETcmdAttributeNew

The CE calls your code resource with this routine selector when it is about to add a new attribute value to a record.

```
struct DETAttributeNewBlock {
            DETCallBlockTargetedHeader
            PackedDSSpecPtr parent;
            short refNum;
            AuthIdentity identity;
            AttributeType attrType;
            AttributeTag attrTag;
            Handle value;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdAttributeNew
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	parent	PackedDSSpecPtr	Record to which the CE adds the new attribute value
→	refNum	short	Reference number for the catalog containing the record that contains the attribute (used only if the catalog is a personal catalog; only personal catalogs use reference numbers)
→	identity	AuthIdentity	Authentication identity used when gaining access to the parent record
↔	attrType	AttributeType	Type of attribute being created
↔	attrTag	AttributeTag	Tag of attribute being created
↔	value	Handle	Value to write (preallocated to default attribute size; resize as needed)

DESCRIPTION

When the user adds a new attribute value to a record, the Catalogs Extension calls the code resource for the parent record with the `kDETcmdAttributeNew` routine selector. The user can add a new attribute value by clicking the Add button in a template to add a new attribute value to a sublist, dragging an attribute value and dropping it onto a record, or editing a property that has not been previously edited (that is, whose value is the default value assigned by the template). The CE adds the new attribute value when the user closes the information page or when you call the `kDETcmdSaveProperty` callback routine.

The `attrType`, `attrTag`, and `value` parameters indicate the default values for the new attribute type, attribute tag, and attribute value. You can return different values for any of these parameters if you wish. Whether or not you change any of these values, return the `kDETDidNotHandle` result code if you want the CE to create the new attribute value. If your routine returns `noErr`, the CE assumes you used the Catalog Manager to create the new attribute value and does not create it for you. Likewise, if your routine returns an error, the CE does not create the attribute value.

The target selector in the `DETTargetSpecification` structure is always `kDETSelf` when the CE calls your `kDETcmdAttributeNew` routine; the target is the aspect of the record that will contain the attribute.

When the user adds a new attribute value to a sublist, the CE calls your `kDETcmdAttributeCreation` routine before it calls your `kDETcmdAttributeNew` routine.

SPECIAL CONSIDERATIONS

If there is no input pattern for an attribute type that has an output pattern (a lookup-table element that has the `useForOutput` flag set), the CE has no way of knowing an attribute value already exists; therefore the CE calls your `kDETcmdAttributeNew` routine every time it processes the output pattern. If your `kDETcmdAttributeNew` routine returns the `kDETDidNotHandle` result code, the record ends up containing multiple attribute values corresponding to a single set of properties. Therefore, if you do not include an input pattern for an attribute type for which you provide an output pattern, your `kDETcmdAttributeNew` routine should return the `noErr` result code when called for that attribute type.

CALL-FOR MASK VALUE

`kDETCallForAttributes`

SEE ALSO

If the attribute value is being added to a sublist, the CE calls your `kDETcmdAttributeCreation` routine (page 5-175) before calling the `kDETcmdAttributeNew` routine.

You can use the Catalog Manager's `DirAddAttributeValue` function to add an attribute value; see the chapter "Catalog Manager" in this book for details.

kDETcmdAttributeChange

The CE calls your code resource with this routine selector when it is about to change an existing attribute value.

```
struct DETAttributeChangeBlock {
            DETCallBlockTargetedHeader
            PackedDSSpecPtr parent;
            short refNum;
            AuthIdentity identity;
            AttributeType attrType;
            AttributeTag attrTag;
            AttributeCreationID attrCID;
            Handle value;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdAttributeChange
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	parent	PackedDSSpecPtr	Record containing the attribute
→	refNum	short	Reference number for the catalog containing the record that contains the attribute (used only if the catalog is a personal catalog; only personal catalogs use reference numbers)
→	identity	AuthIdentity	Authentication identity used when gaining access to the parent record
→	attrType	AttributeType	Type of attribute being changed
↔	attrTag	AttributeTag	Tag of attribute being changed
↔	attrCID	AttributeCreationID	CID of attribute being changed
↔	value	Handle	Value to write (preallocated to the size of the proposed attribute value; resize as needed)

DESCRIPTION

When the user changes an attribute value, the Catalogs Extension calls the code resource of the aspect that's writing the attribute with the `kDETcmdAttributeChange` routine selector. The user can change an attribute value by editing a property that has been previously edited. The CE updates the attribute value when the user closes the information page or when you call the `kDETcmdSaveProperty` callback routine.

The `attrType`, `attrTag`, and `value` parameters indicate the new values for the attribute type, attribute tag, and attribute value. You can return different values for the attribute tag and attribute value parameters if you wish. Whether or not you change either of these values, return the `kDETDidNotHandle` result code if you want the CE to change the attribute value. If your routine returns `noErr`, the CE assumes you used the Catalog Manager to change the attribute value and does not change it for you. Likewise, if your routine returns an error, the CE does not change the attribute value.

SPECIAL CONSIDERATIONS

If there is no input pattern for an attribute type that has an output pattern (a lookup-table element that has the `useForOutput` flag set), the CE has no way of knowing an attribute value already exists. Therefore, every time it processes the output pattern, the CE calls your `kDETcmdAttributeNew` routine rather than your `kDETcmdAttributeChange` routine.

CALL-FOR MASK VALUE

`kDETCallForAttributes`

SEE ALSO

When the CE is about to create a new attribute value, it calls your `kDETcmdAttributeNew` routine (page 5-176).

You can use the Catalog Manager's `DirChangeAttributeValue` function to change an attribute value; see the chapter "Catalog Manager" in this book for details.

kDETcmdAttributeDelete

The CE calls your code resource with this routine selector when it is about to delete an existing attribute value.

```
struct DETAttributeDeleteBlock {
            DETCallBlockTargetedHeader
            PackedDSSpecPtr dsSpec;
            short refNum;
            AuthIdentity identity;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdAttributeDelete`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`dsSpec`	`PackedDSSpecPtr`	Catalog system specifier of the attribute value about to be deleted
→	`refNum`	`short`	Reference number for the catalog containing the record that contains the attribute (used only if the catalog is a personal catalog; only personal catalogs use reference numbers)
→	`identity`	`AuthIdentity`	Authentication identity used when gaining access to the parent record

DESCRIPTION

When the Catalogs Extension is about to delete an attribute value, it calls the code resource of the main aspect of the attribute that's about to be deleted with the `kDETcmdAttributeDelete` routine selector. You can use the `DSSpec` structure provided in the parameter block to determine exactly which attribute value is about to be deleted.

Return the `kDETDidNotHandle` result code if you want the CE to delete the attribute value. If your routine returns `noErr`, the CE assumes you used the Catalog Manager to delete the attribute value and does not delete it for you. Likewise, if your routine returns an error, the CE does not delete the attribute value.

The `DSSpec` structure that describes attribute values has a type of `'entn'` and the following extension value:

```
OSType               'spat'
AttributeCreationID  attributeCreationID
AttributeType        attributeName
```

The attribute creation ID uniquely identifies a specific attribute value even if there is more than one value of the same attribute type.

The `AttributeType` structure is defined as follows:

```
struct AttributeType {
   RStringHeader
   Byte body[kAttributeTypeMaxBytes];
};
```

The `attributeName` field must be packed and padded to an even number of bytes. The `AttributeType` structure is equivalent to an `RString` structure that has a length of `kAttributeTypeMaxBytes` bytes.

CALL-FOR MASK VALUE

`kDETCallForAttributes`

SEE ALSO

You can use the Catalog Manager's `DirDeleteAttributeValue` function to delete an attribute value; see the chapter "Catalog Manager" in this book for details.

The `DSSpec`, `PackedDSSpec`,`AttributeType`, and `RString` structures are described in the chapter "AOCE Utilities" in this book. That chapter also describes utility routines that you can use to pack, unpack, and manipulate these structures.

Attribute creation IDs are returned by the `DirAddAttributeValue` function, the `DirVerifyAttributeValue` function, and the `DirFindValue` function. All of these functions are described in the chapter "Catalog Manager" in this book.

Processing Custom Lookup-Table Pattern Elements

The Catalogs Extension passes to your code resource any lookup-table attribute pattern element types that start with an uppercase letter. When the CE is processing an attribute value to set a property value and encounters a custom pattern element type, it calls your `kDETcmdPatternIn` routine. When the CE is processing a property value to create an attribute value, it calls your `kDETcmdPatternOut` routine.

kDETcmdPatternIn

The CE calls your code resource with this routine selector when it needs to use a custom lookup-table pattern element to set the value of a property.

```
struct DETPatternInBlock {
            DETCallBlockPropertyHeader
            long elementType;
            long extra;
            AttributePtr attribute;
            long dataOffset;
            short bitOffset;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdPatternIn`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`property`	`short`	Property number
→	`elementType`	`long`	Element type
→	`extra`	`long`	Extra field
→	`attribute`	`AttributePtr`	Attribute being parsed
↔	`dataOffset`	`long`	Offset to next byte
↔	`bitOffset`	`short`	Bit offset

DESCRIPTION

The Catalogs Extension passes to your code resource any lookup-table attribute pattern element types that start with an uppercase letter. The `property`, `elementType`, and `extra` fields contain the corresponding parts of the pattern element. The `attribute` field is a pointer to the attribute value being parsed. The `dataOffset` field is the byte offset into the attribute data of the byte to be parsed. The `bitOffset` field is the bit offset within the byte of the next bit to be parsed. The data in the attribute value is at

```
callBlockPtr->patternIn.attribute->value.bytes
```

The next bit to parse is

```
*(callBlockPtr->patternIn.attribute->value.bytes +
  callBlockPtr->patternIn.dataOffset)>>callBlockPtr->
  patternIn.bitOffset++
```

Your `kDETcmdPatternIn` routine should parse the specified attribute data starting at the byte and bit specified by the `dataOffset` and `bitOffset` fields. You can create as many properties as you wish from the attribute data. Use the `kDETcmdSetPropertyNumber`, `kDETcmdSetPropertyRString`, or `kDETcmdSetPropertyBinary` callback routines to set property values. When you are finished processing the attribute data, update the `dataOffset` and `bitOffset` fields to point to the next bit to be processed and return the `noErr` result code.

You should check the access mask of the item you are processing and set the `propertyEditable` flag accordingly.

If your routine returns the `kDETDidNotHandle` result code, the CE calls the code resource of the parent record for the attribute. If your routine returns an error, the CE stops processing attribute data.

SPECIAL CONSIDERATIONS

When the CE creates or changes attribute values, it processes only those properties that have changed and that are included in the list of properties in the lookup table. Therefore, you must use the `'prop'` pattern element in your lookup table to list each property that your code resource processes.

CALL-FOR MASK VALUE

None

SEE ALSO

Lookup-table patterns and pattern elements are described in “The Lookup-Table Resource” beginning on page 5-105.

You can use the `kDETcmdSetPropertyNumber` callback routine (page 5-227), the `kDETcmdSetPropertyRString` callback routine (page 5-228), or the `kDETcmdSetPropertyBinary` callback routine (page 5-229) to set property values.

You can use the `kDETcmdSetPropertyEditable` callback routine (page 5-232) to set the `propertyEditable` flag.

kDETcmdPatternOut

The CE calls your code resource with this routine selector when it needs to use a custom lookup-table pattern element to write an attribute value from a property.

```
struct DETPatternOutBlock {
            DETCallBlockPropertyHeader
            long elementType;
            long extra;
            AttributePtr attribute;
            Handle data;
            long dataOffset;
            short bitOffset;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdPatternOut
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	property	short	Property number
→	elementType	long	Element type
→	extra	long	Extra field
↔	attribute	AttributePtr	Attribute being created
↔	data	handle	Attribute data
↔	dataOffset	long	Offset to next byte to write
↔	bitOffset	short	Bit offset

DESCRIPTION

The Catalogs Extension passes to the code resource any lookup-table attribute pattern element types that start with an uppercase letter. The `property`, `elementType`, and `extra` fields contain the corresponding parts of the pattern element. The `attribute` field points to the attribute being created (the attribute value already has an attribute tag assigned, but the data length and data fields of the value have not yet been filled in). The `data` field is a handle that you can use to contain the data portion of the attribute value (and which you should resize as needed to hold the value). The `dataOffset` field is the byte offset into the attribute of the byte currently being parsed. The `bitOffset` field is the offset within the byte of the next bit to be parsed. You can change these offsets as necessary.

You can use the callback routines described in “Getting Information About Properties” beginning on page 5-213 to determine the property value. You then return the attribute data in the `data` field, resizing it as needed, and updating the `dataOffset` and `bitOffset` fields appropriately.

SPECIAL CONSIDERATIONS

Because when creating or changing attribute values, the CE processes only those properties that have changed and that are included in the list of properties in the lookup table, you must use the `'prop'` pattern element in your lookup table to list each property that your code resource processes.

CALL-FOR MASK VALUE

None

SEE ALSO

lookup-table patterns and pattern elements are described in "The Lookup-Table Resource" beginning on page 5-105.

You can use the callback routines in "Getting Information About Properties" beginning on page 5-213 to determine property values.

Synchronizing Property Values

If you derive any of your property values (including sublist items) from data outside the catalog system or from records or attributes other than the one to which your aspect applies, you can use the code resource routines described in this section to ensure that your property values are updated whenever the Catalogs Extension updates the property values that it maintains.

kDETcmdShouldSync

The CE calls your code resource with this routine selector to check whether the code resource wants to update all property values.

```
struct DETShouldSyncBlock {
            DETCallBlockTargetedHeader
            Boolean shouldSync;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdOpenSelf
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
←	shouldSync	Boolean	Should CE update data?

DESCRIPTION

The Catalogs Extension checks a catalog system flag periodically (and whenever someone calls the `kDETcmdRequestSync` callback routine) to see if the data in the catalog system has changed. If it has, the CE recalculates all the properties derived from the catalog system and updates aspects and information pages accordingly. At the time the CE checks for changes, it calls your code resource with the `kDETcmdShouldSync` routine selector. If you have derived any properties from data outside the catalog system or from records or attributes other than the one to which your aspect applies and you have reason to believe their values have changed, you should return `true` in the `shouldSync` field of the parameter block. In response, the CE updates all the properties whether data in the catalog system has changed or not, giving your code resource a chance to update all of its property values.

You can use this routine, for example, to maintain sublist items that are not directly from the catalog system.

If your routine returns the `kDETDidNotHandle` result code or an error, the CE ignores the `shouldSync` field and behaves as if its value were `false`.

CALL-FOR MASK VALUE

`kDETCallForSyncing`

SEE ALSO

When the CE updates property values, it sends the `kDETcmdDoSync` routine selector, described next, to your code resource.

You can call the `kDETcmdRequestSync` callback routine (page 5-237) at any time to force the CE to check immediately whether the sublist or any properties need updating.

kDETcmdDoSync

The CE calls your code resource with this routine selector to give your code resource a chance to read in and parse its attributes.

```
struct DETDoSyncBlock {DETCallBlockTargetedHeader
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdOpenSelf
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?

DESCRIPTION

When the Catalogs Extension updates all the property values in an aspect—either because data in the catalog system has changed or because you returned `true` in the `shouldSync` field of the parameter block for the `kDETcmdShouldSync` routine—the CE calls your code resource with the `kDETcmdDoSync` routine selector. If you have derived any of your properties from outside the catalog system or from a record or attribute other than the one to which your aspect applies, you should call the `kDETcmdBreakAttribute` routine to update your sublist items and the `kDETcmdBreakAttribute` routine and the set-property routines to update your other properties.

The CE ignores the result code returned by this routine.

CALL-FOR MASK VALUE

`kDETCallForSyncing`

SEE ALSO

Call the `kDETcmdBreakAttribute` routine (page 5-224) to send to the lookup table an attribute value from outside the catalog system or from a record or attribute other than the one to which your aspect applies.

Use the set-property routines (see “Setting Value, Type, and Other Features of Properties” beginning on page 5-223) to set property values, types, and so forth for properties not derived from the catalog system or from the record or attribute to which your aspect applies

You can call the `kDETcmdRequestSync` callback routine (page 5-237) at any time to force the CE to check immediately whether the sublist or any properties need updating.

Custom Property-Type Conversions

You can assign a custom property type to a property value. When the Catalogs Extension encounters a property with a custom type that it has to use as a number or as a string, or when it encounters a number or a string that it has to store as a property with a custom type, the CE calls one of the code resource routines described in this section.

kDETcmdConvertToNumber

The CE calls your code resource with this routine selector when it encounters a property with a custom type that it has to use as a number in a view or when you call the `kDETcmdGetPropertyNumber` callback routine for a property with a custom type.

```
struct DETConvertToNumberBlock {
            DETCallBlockPropertyHeader
            long theValue;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdConvertToNumber`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`property`	`short`	Property number
←	`theValue`	`long`	Converted value

DESCRIPTION

If you assign a custom property type to a property and then use that property in a view where a number is called for (as in a radio button, checkbox, or pop-up menu), the Catalogs Extension calls your code resource with the `kDETcmdConvertToNumber` routine selector. The CE also calls this routine if you call the `kDETcmdGetPropertyNumber` callback routine and specify your custom property. The `property` field of the parameter block indicates the property number of the custom property. You must convert the property value to a number and return it in the field `theValue`.

If your routine returns the `kDETDidNotHandle` result code or an error, the CE uses 0 as the value of the custom property. If your routine returns the `noErr` result code, the CE uses the number your routine returns in the `theValue` field.

CALL-FOR MASK VALUE

None

SEE ALSO

You can use the `'styp'` and `'byte'` lookup-table element types (see "Overriding Default Property-Type Assignments" on page 5-119) or the `kDETcmdSetPropertyType` callback routine (page 5-225) to assign a custom property type to a property.

You use the `kDETcmdGetPropertyNumber` callback routine (page 5-216) to get the value of a number-type property.

Property types are discussed in "Properties" beginning on page 5-84. The property types currently defined by Apple Computer, Inc., are shown in Table 5-2 on page 5-85.

kDETcmdConvertToRString

The CE calls your code resource with this routine selector when it encounters a property with a custom type that it has to use as a string in a view or when you call the `GetPropertyRString` callback routine for a property with a custom type.

```
struct DETConvertToRStringBlock {
            DETCallBlockPropertyHeader
            RStringHandle theValue;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdConvertToRString`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`property`	`short`	Property number
←	`theValue`	`RStringHandle`	Handle to converted value

DESCRIPTION

If you assign a custom property type to a property and then use that property in a view that calls for a string (editable or static text), the Catalogs Extension calls your code resource with the `kDETcmdConvertToRString` routine selector. The CE also calls this routine if you call the `kDETcmdGetPropertyRString` callback routine and specify your custom property. The `property` field of the parameter block indicates the property number of the custom property. You must convert the property value to an `RString` structure, allocate a handle to the `RString`, and return the handle in the field `theValue`. The CE disposes of the handle when it no longer needs it.

If your routine returns the `kDETDidNotHandle` result code or an error, the CE uses an empty `RString` structure as the value of the property. If your routine returns the `noErr` result code, the CE uses the `RString` your routine returns in the `theValue` field.

CALL-FOR MASK VALUE

None

SEE ALSO

You can use the `'styp'` and `'byte'` lookup-table element types (see "Overriding Default Property-Type Assignments" on page 5-119) or the `kDETcmdSetPropertyType` callback routine (page 5-225) to assign a custom property type to a property.

You use the `kDETcmdGetPropertyRString` callback routine (page 5-217) to get the value of a number-type property.

Property types are discussed in "Properties" beginning on page 5-84. The property types currently defined by Apple Computer, Inc., are shown in Table 5-2 on page 5-85.

kDETcmdConvertFromNumber

The CE calls your code resource with this routine selector when it needs to write a number to a property that has a custom property type.

```
struct DETConvertFromNumberBlock {
            DETCallBlockPropertyHeader
            long theValue;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdConvertFromNumber`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`property`	`short`	Property number
→	`theValue`	`UNSIGNED long`	Value to convert

DESCRIPTION

If you have assigned a custom property type to a property and then use that property in a view where the Catalogs Extension uses a number (as in a radio button, checkbox, or pop-up menu), the CE calls your code resource with the `kDETcmdConvertFromNumber` routine selector when it needs to update the property value. The CE also calls this routine if you call the `kDETcmdSetPropertyNumber` callback routine and specify your custom property. The `property` field of the parameter block indicates the property number of the custom property. The `theValue` field contains the value of the property in the form of a number (unsigned long word). You must convert the property value to your custom property type and use the `kDETcmdSetPropertyBinary` callback routine

to write the result directly to the property. The CE ignores the function result of this routine.

CALL-FOR MASK VALUE

None

SEE ALSO

You can use the `'styp'` and `'byte'` lookup-table element types (see "Overriding Default Property-Type Assignments" on page 5-119) or the `kDETcmdSetPropertyType` callback routine (page 5-225) to assign a custom property type to a property.

You use the `kDETcmdSetPropertyNumber` callback routine (page 5-227) to set the value of a number-type property.

You use the `kDETcmdSetPropertyBinary` callback routine (page 5-229) to write an uninterpreted binary block to a property.

Property types are discussed in "Properties" beginning on page 5-84. The property types currently defined by Apple Computer, Inc., are shown in Table 5-2 on page 5-85.

kDETcmdConvertFromRString

The CE calls your code resource with this routine selector when it needs to write a string to a property that has a custom property type.

```
struct DETConvertFromRStringBlock {
            DETCallBlockPropertyHeader
            RStringHandle theValue;
};
```

Parameter block

→	`reqFunction`	`DETCallFunctions`	`kDETcmdConvertFromRString`
↔	`templatePrivate`	`long`	Data stored in template
↔	`instancePrivate`	`long`	Data stored in aspect
→	`callBack`	`DETCallBack`	Callback pointer
→	`target`	`DETTargetSpecification`	Target specifier
→	`targetIsMainAspect`	`Boolean`	Is target main aspect?
→	`property`	`short`	Property number
→	`theValue`	`RStringHandle`	Handle to value to convert

DESCRIPTION

If you have assigned a custom property type to a property and then use that property in a view where the Catalogs Extension uses a string (as in a text field), the CE calls your code resource with the `kDETcmdConvertFromRString` routine selector when it needs

to update the property value. The CE also calls this routine if you call the `kDETcmdSetPropertyRString` callback routine and specify your custom property. The `property` field of the parameter block indicates the property number of the custom property. The `theValue` field contains a handle to the value of the property in the form of an `RString` structure. You must convert the property value to your custom property type and use the `kDETcmdSetPropertyBinary` callback routine to write the result directly to the property. The CE ignores the function result of this routine.

CALL-FOR MASK VALUE

None

SEE ALSO

You can use the `'styp'` and `'byte'` lookup-table element types (see "Overriding Default Property-Type Assignments" on page 5-119) or the `kDETcmdSetPropertyType` callback routine (page 5-225) to assign a custom property type to a property.

You use the `kDETcmdSetPropertyRString` callback routine (page 5-228) to set the value of a string-type property.

You use the `kDETcmdSetPropertyBinary` callback routine (page 5-229) to write an uninterpreted binary block to a property.

Property types are discussed in "Properties" beginning on page 5-84. The property types currently defined by Apple Computer, Inc., are shown in Table 5-2 on page 5-85.

Custom Views and Custom Menus

You can add a custom view to a view list, and you can add custom items to the Catalogs menu. The Catalogs Extension calls the code resource routines in this section to draw custom views, handle mouse-down events in custom views, determine whether a custom menu item should be enabled, and handle the selection of your custom menu items.

kDETcmdCustomViewDraw

The CE calls your code resource with this routine selector when it needs you to draw your custom view.

```
struct DETGetCustomViewDrawBlock {
            DETCallBlockPropertyHeader
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdCustomViewDraw
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	property	short	Property number

DESCRIPTION

If you include a custom view in a view list, the Catalogs Extension calls your `kDETcmdCustomViewDraw` routine when it is drawing or updating the information page that includes your custom view and when you call the `kDETcmdDirtyProperty` callback routine to indicate that the property value associated with the custom view has changed. The `property` field identifies the view to be drawn. The CE sets the graphics port to the window containing the view before calling your routine.

The CE ignores the function result for this routine.

SPECIAL CONSIDERATIONS

Your routine that draws your custom view must leave the QuickDraw state unchanged. If you change the QuickDraw state (pen pattern, background color, and so forth) while drawing your custom view, you must restore it to its original values before returning.

CALL-FOR MASK VALUE

None

SEE ALSO

View lists are described in “View Lists” beginning on page 5-123.

The `kDETcmdDirtyProperty` callback routine is described on page 5-233.

kDETcmdCustomViewMouseDown

The CE calls your code resource with this routine selector when a mouse-down event occurs in your custom view.

```
struct DETCustomViewMouseDownBlock {
            DETCallBlockPropertyHeader
            EventRecord *theEvent;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdCustomViewMouseDown
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	property	short	Property number
→	theEvent	EventRecord	The event record for the mouse-down

DESCRIPTION

If you include a custom view in a view list, the Catalogs Extension calls your `kDETcmdCustomViewMouseDown` routine when a mouse-down event occurs in your custom view. The mouse coordinates in the event record are global; you can use the `GlobalToLocal` QuickDraw routine to obtain the local coordinates.

The CE sets the graphics port to the window containing the view before calling your routine.

The property field identifies the view in which the event occurred.

If your routine returns an error, the CE displays an error dialog box. If your routine returns `noErr` or `kDETDidNotHandle`, the CE does no further processing of the event.

CALL-FOR MASK VALUE

None

SEE ALSO

View lists are described in "View Lists" beginning on page 5-123.

kDETcmdCustomMenuEnabled

The CE calls your code resource with this routine selector to determine whether to enable a menu item that you have added to the Catalogs menu.

```
struct DETCustomMenuEnabledBlock {
            DETCallBlockTargetedHeader
            short menuTableParameter;
            Boolean enable;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdCustomMenuSelected
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	menuTableParameter	short	property field from custom menu table
←	enable	Boolean	Enable the menu item?

DESCRIPTION

If you add a custom menu item to the Catalogs menu by including a kDETInfoPageMenuEntries resource in your information page template, the Catalogs Extension calls your kDETcmdCustomMenuEnabled routine when the user opens the Catalogs menu. To enable the menu item identified by the menuTableParameter field, return noErr with the enable parameter set to true or return kDETDidNotHandle. To disable the menu item, return noErr with the enable parameter set to false or return an error.

CALL-FOR MASK VALUE

None

SEE ALSO

If you enable the menu item and the user chooses it, the CE calls your code resource with the kDETcmdCustomMenuSelected routine selector, described next.

The kDETInfoPageMenuEntries resource is described on page 5-137.

kDETcmdCustomMenuSelected

The CE calls your code resource with this routine selector when the user chooses a menu item that you have added to the Catalogs menu.

```
struct DETCustomMenuSelectedBlock {
            DETCallBlockTargetedHeader
            short menuTableParameter;
};
```

Parameter block

→	reqFunction	DETCallFunctions	kDETcmdCustomMenuSelected
↔	templatePrivate	long	Data stored in template
↔	instancePrivate	long	Data stored in aspect
→	callBack	DETCallBack	Callback pointer
→	target	DETTargetSpecification	Target specifier
→	targetIsMainAspect	Boolean	Is target main aspect?
→	menuTableParameter	short	property field from custom menu table

DESCRIPTION

If you add a custom menu item to the Catalogs menu by including a kDETInfoPageMenuEntries resource in your information page template, the Catalogs Extension calls your kDETcmdCustomMenuSelected routine when the user chooses your menu item. The menuTableParameter field contains the menu parameter from the kDETInfoPageMenuEntries resource for the item the user chose.

If your routine returns an error, the CE displays an error dialog box. If your routine returns noErr or kDETDidNotHandle, the CE does no further processing of the menu selection.

CALL-FOR MASK VALUE

None

SEE ALSO

The kDETInfoPageMenuEntries resource is described on page 5-137.

The CE calls your kDETcmdCustomMenuEnabled routine (page 5-194) to determine whether to enable your menu item.

CE-Provided Functions That Your Code Resource Can Call

Your code resource can call certain routines within the AOCE Catalogs Extension to perform such functions as changing the call-for mask, returning information from the CE, or changing the value of a variable maintained by the CE.

Note

Because the CE first calls your code resource, and then your code calls the CE back to perform these functions, these routines are referred to here as "callback routines." u

Calling CE-Provided Functions

The parameter block that the Catalogs Extension passes to your code resource includes the address of the entry point for CE callback routines. To execute one of these routines, you can use the `CallBackDET` macro, described in this section.

s **WARNING**

Because the parameter block passed by the CE to your code resource contains data that is private to the CE in addition to data intended for your code resource, it is essential that you pass back to the CE the same parameter block that it passed you without altering any of the reserved fields. You cannot reuse a parameter block you saved from a previous time that the CE called your code resource; doing so can cause the Finder to crash. s

IMPORTANT

When you call a template callback routine, you must make sure that the system is in the same state as it was in when the CE called your code resource. Changing such things as the current resource or the heap zone will cause the callback routine to fail. s

CallBackDET

The `CallBackDET` macro calls any AOCE template callback routine.

```
CallBackDET(callBlockPtr, callBackBlockPtr);
```

`callBlockPtr`
A pointer to the parameter block that the CE passed to your code resource.

`callBackBlockPtr`
A pointer to the parameter block that you are providing to the CE callback routine.

The `CallBackDET` macro passes the parameter blocks you provide to the Catalogs Extension callback routine entry point. The function gets the address for this entry point from the `callBack` field of the AOCE template call block that the CE passes to your code resource.

ASSEMBLY-LANGUAGE INFORMATION

The `CallBackDET` macro is implemented entirely in the interface file. There is no trap that corresponds to this macro.

Testing Your Code Resource

The Catalogs Extension provides the `kDETcmdBeep` callback routine so that you can make sure your code resource is being called correctly and is making callbacks correctly.

kDETcmdBeep

This callback routine calls the toolbox `SysBeep` routine.

```
struct DETBeepBlock {
            DETCallBackBlockHeader
};
```

Parameter block

```
→    reqFunction    DETCallBackFunctions    kDETcmdBeep
```

DESCRIPTION

You can use the `kDETcmdBeep` callback routine to test that your code resource and its callback routines are working as you expect.

RESULT CODES

`noErr`	0	No error

Changing the Call-For Mask

You can use the `kDETcmdChangeCallFors` callback routine to modify the call-for mask and therefore change the list of events that result in calls to your code resource.

kDETcmdChangeCallFors

This callback routine changes the call-for mask to a new value.

```
struct DETChangeCallForsBlock {
            DETCallBackBlockTargetedHeader;
            long newCallFors;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdChangeCallFors
→	target	DETTargetSpecification	Target specifier
→	newCallFors	long	New call-for mask

DESCRIPTION

You can modify the call-for mask for the code resource associated with your aspect template at any time. You use the `target` parameter to specify the aspect template whose code resource is your target and the `newCallFors` parameter to provide a new call-for mask. The Catalogs Extension uses the same the call-for mask for every aspect created from the aspect template.

Most code resources set the call-for mask at template initialization time and never change it. However, you might want to change the call-for mask before you call a callback that might result in additional unwanted calls to your code resource. In that case you can set the call-fors mask to `kDETCallForNothing`, call the callback routine, and then reset the call-for mask to its former value.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	`DSSpec` in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect

SEE ALSO

The target specifier is described in "Target Specifier" on page 5-142.

The call-for mask is described in "Call-For Mask" on page 5-149.

Process Control

The routines in this section give you some control over process switching on the user's computer. You should use the `kDETcmdAboutToTalk` callback routine when you want to display a dialog box or otherwise interact with the user outside of an information page. You can use the `kDETcmdBusy` routine to initiate a process switch to allow some other process to complete before returning control to you.

kDETcmdAboutToTalk

This callback routine brings the Finder to the front and disables the watch cursor.

```
struct DETAboutToTalkBlock {
            DETCallBackBlockHeader
};
```

Parameter block

```
→    reqFunction    DETCallBackFunctions    kDETcmdAboutToTalk
```

DESCRIPTION

You should call this routine whenever you are about to display a dialog box or interact with the user in any way outside of the information page. When your code resource returns control to the CE, the CE terminates this state.

RESULT CODES

noErr	0	No error

kDETcmdBusy

This callback routine initiates a process switch and prevents user action.

```
struct DETBusyBlock {
            DETCallBackBlockHeader
};
```

Parameter block

```
→    reqFunction    DETCallBackFunctions    kDETcmdBusy
```

DESCRIPTION

The `kDETcmdBusy` callback routine initiates a process switch; its effect is similar to that of the `WaitNextEvent` function. It sounds a system beep if the user presses the mouse button or a key. You can use this routine to give other processes some time to complete an operation.

In general, code resource routines should complete operation quickly and return. You should use this routine only if you have a special need to cause a process switch before returning control to the Finder.

RESULT CODES

noErr	0	No error

SEE ALSO

The `WaitNextEvent` function is described in the "Event Manager" chapter of *Inside Macintosh: Macintosh Toolbox Essentials*.

Handling Drags and Drops

When the user drags one or more objects and drops them onto another object, the Catalogs Extension calls your code resource if either object was an AOCE catalog object for which you provided an aspect template. Your code resource can use the routines in this section to determine the number of objects dropped and the natures of the objects involved.

kDETcmdGetCommandSelectionCount

This callback routine returns the command selection count.

```
struct DETGetCommandSelectionCountBlock {
            DETCallBackBlockHeader
            long count;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdGetCommandSelectionCount
←	count	long	Command selection count

DESCRIPTION

When the user drops one or more objects onto a catalog object for which you have provided an aspect template, the Catalogs Extension calls your code resource (if any) with the `kDETcmdDropQuery` routine selector. Your code resource returns a value telling the CE how to handle the drop. One possible value you can return is a property number, which causes the CE to call your code resource with a property command. When the CE calls your code resource with the `kDETcmdPropertyCommand` routine selector resulting from a drop, you can call the `kDETcmdGetCommandSelectionCount` callback routine to find out how many objects are being dropped. Then for each item, call the `kDETcmdGetCommandItemN` callback routine to determine the nature of the object being dropped.

The count of objects begins with 1; that is, if one object is being dropped, the `count` field contains a 1.

RESULT CODES

noErr	0	No error

SEE ALSO

Call the kDETcmdGetCommandItemN callback routine (described next) to determine the nature of the object being dropped.

The kDETcmdDropQuery routine is described on page 5-172.

The kDETcmdPropertyCommand routine is described on page 5-159.

kDETcmdGetCommandItemN

This callback routine returns a specific command selection item.

```
struct DETGetCommandItemNBlock {
      DETCallBackBlockHeader
      long itemNumber;
      DETItemType itemType;
      union {
         DETFSInfo** fsInfo;
         struct {
                  PackedDSSpecPtr* dsSpec;
                  short refNum;
                  AuthIdentity identity;
         } ds;
         PackedDSSpecPtr* dsSpec;
         LetterSpec** ltrSpec;
      } item;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdGetCommandItemN
→	itemNumber	long	Number of item to retrieve, starting at 1
←	itemType	DETItemType	Type of item to be returned
←	item	union	Address or letter specifier of item

DESCRIPTION

When the user drops one or more objects onto a catalog object for which you have provided an aspect template, the Catalogs Extension calls your code resource (if any) with the kDETcmdDropQuery routine selector once for each object dropped. Your

drop-query routine can call the kDETcmdGetCommandItemN callback routine to determine the nature of the object being dropped. When the user drags a catalog object and drops it onto another catalog object or onto an HFS object, the CE calls the code resource in the aspect of the dragged object with the kDETcmdDropMeQuery routine selector. Your drop-me query routine can call the kDETcmdGetCommandItemN callback routine to determine the nature of the destination object. Both your drop-query and drop-me query routines return a value telling the CE how to handle the drop.

One possible value you can return is a property number, which causes the CE to call your code resource with a property command. The CE groups all property commands that use the same property number resulting from a drop and calls your code resource once. When the CE calls your code resource with the kDETcmdPropertyCommand routine selector resulting from a drop, you can call the kDETcmdGetCommandSelectionCount callback routine to find out how many items are being dropped. Then for each item, call the kDETcmdGetCommandItemN callback routine to determine the nature of the object being dropped.

The kDETcmdGetCommandItemN callback routine returns information about the item you specify in the format you specify with the itemType parameter. The possible values of the itemType parameter are as follows:

```
enum DETItemType {
   kDETHFSType = 0,              /* HFS item type */
   kDETDSType,                   /* catalog service item type */
   kDETMailType,                 /* mail (letter) item type */
   kDETMoverType,                /* sounds, fonts, etc., from inside
                                     a suitcase or system file */
   kDETLastItemType = 0xF0000000 /* force itemType to be a long */
};

typedef enum DETItemType DETItemType;
```

The item parameter is a union of several structures, as shown in the DETGetCommandItemNBlock structure at the beginning of this routine description.

If you request an HFS item type, the routine returns a handle to a file system information structure. This structure includes the file system specification structure for the HFS object, plus its file type, file creator, and Finder flags. The file system information structure is defined by the DETFSInfo data type.

```
struct DETFSInfo {
   OSType fileType;        /* file type */
   OSType fileCreator;     /* file creator */
   unsigned short fdFlags; /* Finder flags */
   FSSpec fsSpec;          /* FSSpec */
};

typedef struct DETFSInfo DETFSInfo;
```

If you request a catalog service item type, the routine returns a ds structure.

```
struct {
        PackedDSSpecPtr* dsSpec;   /* DSSpec for item */
        short refNum;              /* refnum for returned address */
        AuthIdentity identity;     /* identity for returned address */
     } ds;
```

This structure includes a handle to a catalog service specification structure (DSSpec) that identifies the item, a personal catalog reference number if the item is in a personal catalog, and an authentication identity if the item is in a catalog other than a personal catalog. The CE allocates the handle to the DSSpec structure but you must dispose of the handle when you are finished with it.

If you request a mail item type, the routine returns a handle to a letter-specification (ltrSpec) structure. The CE allocates the handle but you must dispose of the handle when you are finished with it. You can use the letter-specification structure in the SMPGetLetterInfo function to get information about the letter. The letter-specification structure is defined by the LetterSpec data type.

```
struct LetterSpec {
   unsigned long  spec[3];
};
```

Your kDETcmdDropQuery or kDETcmdDropMeQuery routine might receive a kDETMoverType item type, indicating a Finder object, such as a font or sound, that is inside a suitcase or system file. To manipulate such objects, you must set the copyToHFS parameter to true in your kDETcmdDropQuery or kDETcmdDropMeQuery routine so that the user will copy them to HFS objects and try the drop again.

If the object for which you request information is not available in the format you request, the routine returns the kDETRequestedTypeUnavailable result code.

It is generally best to request item types in the order you prefer to deal with them. For example, if you want to do something with a catalog object, you might ask first for an item of type kDETDSType. If there are no such objects and your code resource can handle HFS objects (such as information cards), you might next try the kDETHFSType item type.

RESULT CODES

noErr	0	No error
kDETInvalidCommandItemNumber	–15007	Command item number out of range
kDETUnableToGetCommadnItemSpec	–15008	Unable to retrieve information about item (possibly out of memory)
kDETRequestedTypeUnavailable	–15009	Item could not be represented in the specified format

SEE ALSO

The `kDETcmdDropQuery` routine is described on page 5-172.

The `kDETcmdPropertyCommand` routine is described on page 5-159.

You can use the letter-specification structure in the `SMPGetLetterInfo` function to get information about the letter; the `SMPGetLetterInfo` function is described in the chapter "Standard Mail Package" in this book.

Working With Templates

The routines in this section allow your code resource to determine how many templates have been loaded by the system, to locate template files and template resources, and to close and unload all templates.

kDETcmdTemplateCounts

This callback routine returns the numbers of aspect and information page templates in the system.

```
struct DETTemplateCounts {
            DETCallBackBlockHeader
            long aspectTemplateCount;
            long infoPageTemplateCount;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdTemplateCounts`
←	`aspectTemplateCount`	`long`	Number of aspect templates
←	`infoPageTemplateCount`	`long`	Number of information page templates

DESCRIPTION

You can use the information returned by the `DETcmdTemplateCounts` callback routine if you want to iterate through all of the templates in the system; for example, to search for a custom resource of a specific type.

RESULT CODES

`noErr`	0	No error

kDETcmdGetTemplateFSSpec

This callback routine returns the file system specification for a template file.

```
struct DETGetTemplateFSSpecBlock {
            DETCallBackBlockTargetedHeader
            FSSpec fsSpec;
            short baseID;
            long aspectTemplateNumber;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdGetTemplateFSSpec
→	target	DETTargetSpecification	Target specifier
←	fsSpec	FSSpec	FSSpec of file containing the template
←	baseID	short	Base resource ID of this template
←	aspectTemplateNumber	long	The template number for this aspect template

DESCRIPTION

The kDETcmdGetTemplateFSSpec callback routine returns an FSSpec structure for the file containing the target template.

You can use a template index number (the number that the Catalogs Extension assigned to this aspect template when it loaded the template into memory) for the target specifier in the target field. Whatever type of target selector you use, this function returns the index number of the template.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect

SEE ALSO

The `FSSpec` structure is described in the chapter "Introduction to File Management" in *Inside Macintosh: Files*.

Target selectors are described in "Target Specifier" on page 5-142.

kDETcmdGetResource

This callback routine returns a template resource.

```
struct DETGetResourceBlock {
            DETCallBackBlockPropertyHeader
            ResType resourceType;
            Handle theResource;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdGetResource`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
→	`resourceType`	`ResType`	Resource type
←	`theResource`	`Handle`	Handle to the resource

DESCRIPTION

The `kDETcmdGetResource` callback routine returns a handle to a resource. This resource has a resource ID equal to the property number plus the template's base ID and has the resource type you specify. If the call-for mask is set appropriately, the routine calls your code resource with the `kDETcmdDynamicResource` routine selector for all resources except those listed in a forwarder template. It takes the record type resource (at offset `kDETRecordType`), attribute type resource (`kDETAttributeType`), and attribute value tag resource (`kDETAttributeValueTag`) from the version of the template that's stored in memory, because these resources might have been added by a forwarder template or by the `kDETcmdDynamicForwarders` code-resource routine.

You can use the `kDETAspectTemplate` and `kDETInfoPageTemplate` target selectors in the target specifier you use with this callback routine. These target selectors allow you to specify the index number of the template assigned by the Catalogs Extension when it loads the template into memory. You might want to use these target selectors, for example, to search for every template in memory that contains a resource of a specific type.

If the targeted template does not contain the specified resource, the routine returns the `resNotFound` result code.

You must dispose of the resource handle when you have finished using it.

RESULT CODES

noErr	0	No error
resNotFound	–192	Could not find specified resource
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect

SEE ALSO

The kDETcmdDynamicResource routine is described on page 5-156.

Target selectors are described in "Target Specifier" on page 5-142.

Forwarder templates are described in "Components of Forwarder Templates" on page 5-138, and the kDETcmdDynamicForwarders code-resource routine is described on page 5-155.

kDETcmdUnloadTemplates

This callback routine unloads all templates from memory.

```
struct DETUnloadTemplatesBlock {
   DETCallBackBlockHeader
};
```

Parameter block

→ reqFunction DETCallBackFunctions kDETcmdUnloadTemplates

DESCRIPTION

This callback routine causes the Catalogs Extension to close all template-related windows, release all memory used by templates, and delete all templates and template-related data structures from memory. At that point, you can put templates and new versions of templates in the Extensions folder. The CE loads the new templates the next time they are needed.

This routine should not normally be called by a template. It is provided for the convenience of template developers so that you don't need to reboot your test system every time you want to try a new version of a template or add a new template to the system.

RESULT CODES

noErr	0	No error

Working With Catalog Objects

The routines in this section return a catalog system specification for an object and let your code resource open a catalog object.

kDETcmdGetDSSpec

This callback routine returns a catalog system specification structure for the targeted object.

```
struct DETGetDSSpecBlock {
            DETCallBackBlockTargetedHeader
            PackedDSSpecPtr* dsSpec;
            short refNum;
            AuthIdentity identity;
            Boolean isAlias;
            Boolean isRecordRef;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdGetDSSpec
→	target	DETTargetSpecification	Target specifier
←	dsSpec	PackedDSSpecPtr*	Handle to DSSpec
←	refNum	short	Reference number for the catalog containing the object (used only if the catalog is a personal catalog; only personal catalogs use reference numbers)
←	identity	AuthIdentity	Authentication identity used to gain access to the catalog containing the object
←	isAlias	Boolean	True if this DSSpec is for an alias to a record
←	isRecordRef	Boolean	Reserved

DESCRIPTION

The Catalogs Extension allocates the handle to store the `PackedDSSpec` structure returned by this function. Your code resource must deallocate the handle when done.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect

SEE ALSO

The `PackedDSSpec` structure and functions that you can use to unpack it are described in the chapter "AOCE Utilities" in this book.

You can use the `kDETcmdOpenDSSpec` callback routine (described next) to open the object described by the `DSSpec` structure.

kDETcmdOpenDSSpec

This callback routine opens the object for which you supply a catalog specification (`DSSpec`) structure.

```
struct DETOpenDSSpecBlock {
            DETCallBackBlockHeader
            PackedDSSpecPtr dsSpec;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdOpenDSSpec`
→	`dsSpec`	`PackedDSSpecPtr`	`DSSpec` of object to be opened

DESCRIPTION

You can use the `kDETcmdOpenDSSpec` callback routine to open any object for which you have a catalog specification structure (`DSSpec`). The exact effect of opening the object depends on the object; the Catalogs Extension might open an information page, or the object's code resource might perform some other action. The CE does not actually open the object until after your code resource returns.

RESULT CODES

`noErr`	0	No error
`kDETInvalidDSSpec`	–15010	Could not resolve `DSSpec`

SEE ALSO

You can use the preceding routine, `kDETcmdGetDSSpec`, to obtain the `DSSpec` structure for a catalog object.

Edit-Text Routines

The callback routines in this section give your code resource some control over edit-text views. The first routine, `kDETcmdGetOpenEdit`, returns the property number of an edit-text view. The `kDETcmdCloseEdit` routine closes a specific edit-text view.

kDETcmdGetOpenEdit

This callback routine returns the property number of the edit-text view that the user is currently editing.

```
struct DETGetOpenEditBlock {
            DETCallBackBlockTargetedHeader
            short viewProperty;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdGetOpenEdit`
→	`target`	`DETTargetSpecification`	Target specifier
←	`viewProperty`	`short`	The property number of the view being edited

DESCRIPTION

If no edit-text view is currently being edited, this function returns the value `kDETNoProperty` in the `viewProperty` field.

Note that, because this routine can be targeted, you can use it in the code resource for a parent to get information about the information page of a child.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found
`kDETNoSuchView`	–15013	No view found with specified property number

kDETcmdCloseEdit

This callback routine closes the currently open edit-text view.

```
struct DETCloseEditBlock {
            DETCallBackBlockTargetedHeader
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdCloseEdit`
→	`target`	`DETTargetSpecification`	Target specifier

DESCRIPTION

This callback routine removes the focus box (if any) from the currently open edit-text view, removes the insertion point from the view, and finalizes the edit. After you call this routine, the user must click again within the view to reopen the edit text. The information page must be open when you call this routine; if it is not, the function returns the `kDETInfoPageNotOpen` result code.

Note that, because this routine can be targeted, you can use it in the code resource for a parent to affect the information page of a child.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETInfoPageNotOpen`	–15012	Information page not open

SEE ALSO

To determine the property number of the currently open edit-text view, use the `kDETcmdGetOpenEdit` callback routine (page 5-211).

Getting Information About Properties

The routines described in this section provide information about properties. Note that because the Catalogs Extension looks up information in catalogs asynchronously, it might not have found the information you are asking for if it has not had time to complete its search. You can use the value of the property `kDETPastFirstLookup` to determine whether the CE has completed its catalog search. This property value equals 0 until the search is complete, after which it equals 1.

When your code resource requests the value of a property, you use the `kDETcmdGetPropertyNumber`, `kDETcmdGetPropertyRString`, or `kDETcmdGetPropertyBinary` callback routine to obtain the property as a specific type. If the actual property containing the value is of a different type, the CE automatically converts the value to the requested type, calling the template code resource if the source property is a custom type.

Table 5-15 summarizes the CE's actions when you request a property value. See Table 5-16 on page 5-223 for the conversions the CE performs when you set a property value.

Table 5-15 Property-type conversions on requesting a property value

Callback routine	Property type	Conversion
`kDETcmdGetPropertyNumber`	Number	None
`kDETcmdGetPropertyNumber`	String	Interprets as number, ignoring non-numeric characters
`kDETcmdGetPropertyNumber`	Binary	Takes first 4 bytes of binary data
`kDETcmdGetPropertyNumber`	Custom	Calls code resource `kDETcmdConvertToNumber` routine
`kDETcmdGetPropertyRString`	Number	Converts unsigned number to string
`kDETcmdGetPropertyRString`	String	None
`kDETcmdGetPropertyRString`	Binary	Interprets binary data as an `RString` structure
`kDETcmdGetPropertyRString`	Custom	Calls code resource `kDETcmdConvertToRString` routine
`kDETcmdGetPropertyBinary`	Number	None
`kDETcmdGetPropertyBinary`	String	None
`kDETcmdGetPropertyBinary`	Binary	None
`kDETcmdGetPropertyBinary`	Custom	None

The `kDETcmdGetPropertyChanged` and `kDETcmdGetPropertyEditable` routines get the values of the property-changed and property-editable flags for a specific property.

kDETcmdGetPropertyType

This callback routine returns the type of the specified property.

```
struct DETGetPropertyTypeBlock {
            DETCallBackBlockPropertyHeader
            short propertyType;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdGetPropertyType
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
←	propertyType	short	Property type

DESCRIPTION

Standard property types are `kDETPrTypeNumber` for numbers, `kDETPrTypeString` for strings, or `kDETPrTypeBinary` for binary blocks. You can also define your own property types. If you have never explicitly set the type of a property—either by using a lookup-table pattern element, by using a resource type for the property that confers a default property type (`'rstr'`, `'detn'`, or `'detb'`), or by using the `kDETcmdSetPropertyType` callback routine—then the property is of type `kDETPrTypeBinary`.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETUnableToAccessProperty	–15011	Property could not be found

SEE ALSO

Property types are described in "Properties" beginning on page 5-84.

Code-resource routines that you can provide to convert custom property types to and from standard property types are described in "Custom Property-Type Conversions" beginning on page 5-188.

You can use lookup-table elements to set property types. Lookup tables are described in "The Lookup-Table Resource" beginning on page 5-105.

You can use the `kDETcmdSetPropertyType` callback routine (page 5-225) to change the type of a property.

kDETcmdGetPropertyNumber

This callback routine returns the value of a property as a number.

```
struct DETGetPropertyNumberBlock {
            DETCallBackBlockPropertyHeader
            unsigned long propertyValue;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdGetPropertyNumber
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
←	propertyValue	long	Property value

DESCRIPTION

A property of type kDETPrTypeNumber is stored internally as an unsigned long word, and this function returns that value.

If the property is of type kDETPrTypeString, the kDETcmdGetPropertyNumber function removes all nonnumeric characters and returns the remaining string as a number. The function does not recognize minus signs (–), hexadecimal signs ($ or 0x), or other special symbols when converting strings to numbers.

If the property is of type kDETPrTypeBinary, the function returns the first 4 bytes of the property value as a number.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETUnableToAccessProperty	–15011	Property could not be found

SEE ALSO

You can use the `kDETcmdGetPropertyType` callback routine (page 5-214) to determine the type of a property before you get its value.

To get a property value as a string, use the `kDETcmdGetPropertyRString` callback routine, described next.

To get a property value as a binary block, use the `kDETcmdGetPropertyBinary` callback routine (page 5-219).

kDETcmdGetPropertyRString

This callback routine returns the value of a property as an `RString` structure.

```
struct DETGetPropertyRStringBlock {
            DETCallBackBlockPropertyHeader
            RStringHandle propertyValue;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdGetPropertyRString`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
←	`propertyValue`	`RStringHandle`	Handle to property value

DESCRIPTION

A property of type `kDETPrTypeString` is stored internally as an `RString` structure, and this callback routine returns that value. If the property is of type `kDETPrTypeNumber`, this routine converts the number to an `RString`. If the property is of type `kDETPrTypeBinary`, this routine assumes the binary block contains an `RString` and returns it as such.

When this callback routine completes with the `noErr` result code, the Catalogs Extension allocates the handle in the `propertyValue` field. It is your responsibility to deallocate it when done. The function always returns a valid handle, even if the string is of length 0.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETUnableToAccessProperty	–15011	Property could not be found

SEE ALSO

You can use the kDETcmdGetPropertyType callback routine (page 5-214) to determine the type of a property before you get its value.

To get a property value as a number, use the kDETcmdGetPropertyNumber callback routine (page 5-216).

To get a property value as a binary block, use the kDETcmdGetPropertyBinarySize callback routine, described next.

kDETcmdGetPropertyBinarySize

This callback routine returns the size of a property value.

```
struct DETGetPropertyBinarySizeBlock {
            DETCallBackBlockPropertyHeader
            long propertyBinarySize;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdGetPropertyBinarySize
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
←	propertyBinarySize	long	Property size

DESCRIPTION

This function treats the property as a binary block regardless of the property type, returning the number of bytes in the property value. You can use this function to determine how many bytes of data will be returned by the `kDETcmdGetPropertyBinary` function.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found

SEE ALSO

This function tells you how many bytes of data will be returned by the `kDETcmdGetPropertyBinary` function (described next) for a given property.

kDETcmdGetPropertyBinary

This callback routine returns the value of a property as a binary block.

```
struct DETGetPropertyBinaryBlock {
            DETCallBackBlockPropertyHeader
            Handle propertyValue;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdGetPropertyBinary`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
←	`propertyValue`	`Handle`	Handle to property value

DESCRIPTION

The `kDETcmdGetPropertyBinary` function returns the value of a property as an uninterpreted binary block, regardless of the type of the property. If the property is of type `kDETPrTypeString`, for example, this function returns the `RString` character set and data length fields along with the string itself as binary data.

When this callback routine completes with the `noErr` result code, the Catalogs Extension allocates the handle in the `propertyValue` field. It is your responsibility to deallocate the handle when done.

SPECIAL CONSIDERATIONS

The size of the handle returned by this routine is not the size of the property value. Use the `kDETcmdGetPropertyBinarySize` callback routine to determine the size of a property value.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found

SEE ALSO

You can use the `kDETcmdGetPropertyType` callback routine (page 5-214) to determine the type of a property before you get its value.

You can use the `kDETcmdGetPropertyBinarySize` callback routine (page 5-218) to determine the size of a property value before calling the `kDETcmdGetPropertyBinary` function.

To get a property value as a number, use the `kDETcmdGetPropertyNumber` callback routine (page 5-216).

To get a property value as a string, use the `kDETcmdGetPropertyRString` callback routine (page 5-217).

kDETcmdGetPropertyChanged

This callback routine indicates whether a property value has been changed.

```
struct DETGetPropertyChangedBlock {
            DETCallBackBlockPropertyHeader
            Boolean propertyChanged;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdGetPropertyChanged
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
←	propertyChanged	Boolean	Is property-changed flag set?

DESCRIPTION

This function returns the value of the property-changed flag, which indicates whether the user has changed this property.

If the property-changed flag for this property is set, the Catalogs Extension saves the value of the property when the user closes the information page. You can check the value of this field and save the property value yourself if you have a special need to do so. In addition, if other portions of your display depend on the value of this property, you can use this knowledge to update the display.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETUnableToAccessProperty	–15011	Property could not be found

SEE ALSO

You can use the kDETcmdSetPropertyChanged callback routine (page 5-231) to set the property-changed flag for a property.

kDETcmdGetPropertyEditable

This callback routine indicates whether a property can be edited by the user or whether a control view is enabled.

```
struct DETGetPropertyEditableBlock {
            DETCallBackBlockPropertyHeader
            Boolean propertyEditable;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdGetPropertyEditable
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
←	propertyEditable	Boolean	Is property editable?

DESCRIPTION

The access controls for the dNode, record, and attribute determine whether a property is editable. You can also use the kDETcmdSetPropertyEditable callback routine to make a text view uneditable or to disable a control view. Note that if a property is not editable, neither is a text view based on that property. Also, controls that would change the value of that property are not enabled.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETUnableToAccessProperty	–15011	Property could not be found

SEE ALSO

You can use the kDETcmdSetPropertyEditable callback routine (page 5-232) to set or clear the property-editable flag.

Setting Value, Type, and Other Features of Properties

The routines in this section let your code resource set property values and other property features. The first routine, `kDETcmdBreakAttribute`, sends an attribute to the lookup table to create or update one or more properties. The `kDETcmdSetPropertyType` routine sets the type of a property. The `kDETcmdSetPropertyNumber`, `kDETcmdSetPropertyRString`, and `kDETcmdSetPropertyBinary` commands set the values of properties, converting the types of the values as shown in Table 5-16. See Table 5-15 on page 5-214 for the conversions the Catalogs Extension performs when you get a property value.

Table 5-16 Property-type conversions on setting a property value

Callback routine	Property type	Conversion
`kDETcmdSetPropertyNumber`	Number	None
`kDETcmdSetPropertyNumber`	String	Converts unsigned number to string
`kDETcmdSetPropertyNumber`	Binary	Sets type to number, then sets value
`kDETcmdSetPropertyNumber`	Custom	Calls code resource `kDETcmdConvertFromNumber` routine
`kDETcmdSetPropertyRString`	Number	Interprets as number, ignoring non-numeric characters
`kDETcmdSetPropertyRString`	String	None
`kDETcmdSetPropertyRString`	Binary	Sets type to string, then sets value
`kDETcmdSetPropertyRString`	Custom	Calls code resource `kDETcmdConvertFromRString` routine
`kDETcmdSetPropertyBinary`	Number	Sets value, leaving type as number
`kDETcmdSetPropertyBinary`	String	Sets value, leaving type as string
`kDETcmdSetPropertyBinary`	Binary	None
`kDETcmdSetPropertyBinary`	Custom	Sets value, leaving custom type as defined by developer

The `kDETcmdSetPropertyChanged` and `kDETcmdSetPropertyEditable` routines set the property-changed and property-editable flags for a specific property. The `kDETcmdDirtyProperty` routine causes the CE to redraw the view associated with a property. The `kDETcmdSaveProperty` causes the CE to save a property immediately.

kDETcmdBreakAttribute

This callback routine causes the CE to parse an attribute.

```
struct DETBreakAttributeBlock {
            DETCallBackBlockTargetedHeader
            AttributePtr breakAttribute;
            Boolean isChangeable;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdBreakAttribute
→	target	DETTargetSpecification	Target specifier
→	breakAttribute	AttributePtr	Attribute to parse
→	isChangeable	Boolean	Can user change value?

DESCRIPTION

The Catalogs Extension uses the lookup table of the target aspect to process the attribute pointed to by the `breakAttribute` field. This routine allows you to use an attribute value from a different record or from outside the catalog system. The `isChangeable` field indicates whether the user can edit the value so that the CE can set the property-editable flag for the property.

Note
A lookup table can contain only one input pattern and one output pattern for each attribute type. Therefore, although the CE places no restriction on the number of attribute values that can be assigned to each attribute type, lookup-table patterns are designed to work only for those multivalued attributes that appear in sublists. u

SPECIAL CONSIDERATIONS

If your `kDETcmdDoSync` code resource routine uses the `kDETcmdBreakAttribute` callback to supply sublist items from outside the AOCE catalog system, you must supply a unique type and CID to each item, and you must use the same type and CID for that item every subsequent time the CE calls your `kDETcmdDoSync` routine. Otherwise, the CE deletes the item as obsolete.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect

SEE ALSO

You must call the kDETcmdBreakAttribute callback routine from your kDETcmdDoSync code resource routine (page 5-186) if you are providing attribute values from outside the AOCE catalog system.

Lookup tables are described in "The Lookup-Table Resource" beginning on page 5-105.

kDETcmdSetPropertyType

This callback routine sets a property's type.

```
struct DETSetPropertyTypeBlock {
            DETCallBackBlockPropertyHeader
            short newType;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdSetPropertyType
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
→	newType	short	New property type

DESCRIPTION

You can use the `kDETcmdSetPropertyType` callback routine to set the type of a property. The standard AOCE property types are `kDETPrTypeNumber` for numbers, `kDETPrTypeString` for strings, or `kDETPrTypeBinary` for binary blocks. You can also define your own property types. Because Apple Computer, Inc., reserves all property-type values less than or equal to 0, you must give your property type a positive value.

Note that this routine just sets the property's type; it does not convert the property value to the new type. You should convert the property value or redraw the display as appropriate.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found
`kDETPropertyBusy`	–15020	Specified property is being edited

SEE ALSO

Property types are described in "Properties" beginning on page 5-84.

You can use the `kDETcmdGetPropertyType` callback routine (page 5-214) to determine the type of a property.

You can use the `kDETcmdDirtyProperty` callback routine (page 5-233) to cause the CE to redraw the view.

Whenever the CE needs to convert to or from one of your private property types, it calls your code resource. Code resource routines that you can provide to convert custom property types to and from standard property types are described in "Custom Property-Type Conversions" beginning on page 5-188.

You can use lookup-table elements to set property types. Lookup tables are described in "The Lookup-Table Resource" beginning on page 5-105.

kDETcmdSetPropertyNumber

This callback routine sets the value of a property using a number as input.

```
struct DETSetPropertyNumberBlock {
            DETCallBackBlockPropertyHeader
            unsigned long newValue;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdSetPropertyNumber
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
→	newValue	long	New property value

DESCRIPTION

This routine sets the value of a property to the value in the `newValue` field and causes the affected views to be redrawn. If the property is of type `kDETPrTypeString`, the Catalogs Extension converts the unsigned number in the `newValue` field to an `RString`. If the property is of type `kDETPrTypeBinary`, the CE sets the property type to `kDETPrTypeNumber` before setting its value. If the property is a custom type, the CE calls the code resource's `kDETcmdConvertFromNumber` routine to convert the value and does not change the property's type.

Note that setting the value of a property does not automatically set its changed flag. You must call the `kDETcmdSetPropertyChanged` callback routine to set the changed flag if you want the CE to save the new value when the user closes the information page.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETUnableToAccessProperty	–15011	Property could not be found
kDETPropertyBusy	–15020	Specified property is being edited

SEE ALSO

You can use the `kDETcmdGetPropertyNumber` callback routine (page 5-216) to determine the value of a number property.

To cause the CE to save the new property value, call the `kDETcmdSetPropertyChanged` callback routine (page 5-231).

kDETcmdSetPropertyRString

This callback routine sets the value of a property using an `RString` as input.

```
struct DETSetPropertyRStringBlock {
            DETCallBackBlockPropertyHeader
            RStringPtr newValue;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdSetPropertyRString`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
→	`newValue`	`RStringPtr`	Pointer to new property value

DESCRIPTION

This routine sets the value of a property to the value in the `newValue` field and causes the affected views to be redrawn. If the property is of type `kDETPrTypeNumber`, the Catalogs Extension removes all nonnumeric characters and uses the remaining number to set the property value. The function does not recognize minus signs (–), hexadecimal signs ($ or 0x), or other special symbols when converting strings to numbers. If the property is of type `kDETPrTypeBinary`, the CE sets the property type to `kDETPrTypeString` before setting its value. If the property is a custom type, the CE calls the code resource's `kDETcmdConvertFromRString` routine to convert the value and does not change the property's type.

Note that setting the value of a property does not automatically set its changed flag. You must call the `kDETcmdSetPropertyChanged` callback routine to set the changed flag if you want the CE to save the new value when the user closes the information page.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETUnableToAccessProperty	–15011	Property could not be found
kDETPropertyBusy	–15020	Specified property is being edited

SEE ALSO

You can use the kDETcmdGetPropertyRString callback routine (page 5-217) to determine the value of a string property.

To cause the CE to save the new property value, call the kDETcmdSetPropertyChanged callback routine (page 5-231).

kDETcmdSetPropertyBinary

This callback routine sets the value of a property using a binary value as input.

```
struct DETSetPropertyBinaryBlock {
            DETCallBackBlockPropertyHeader
            Ptr newValue;
            long newValueSize;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdSetPropertyBinary
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
→	newValue	Ptr	Pointer to new property value
→	newValueSize	long	Size of new value

DESCRIPTION

This routine sets the value of a property to the value in the `newValue` field and causes the affected view to be redrawn. If the property is of type `kDETprTypeNumber`, the Catalogs Extension assumes the binary value is a number and sets the property length accordingly. If the property is of type `kDETprTypeString`, the CE uses the `newValueSize` parameter as the length of the property but and sets the property value to the binary block you provide. (Note that the CE will subsequently assume this property value to be an `RString` structure, interpreting the first 4 bytes as the `charSet` and `dataLength` fields.) If the property is a custom type, the CE sets the property length to the size in the `newValueSize` parameter and does not change the property's type.

Note that setting the value of a property does not automatically set its changed flag. You must call the `kDETcmdSetPropertyChanged` callback routine to set the changed flag if you want the CE to save the new value when the user closes the information page.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found
`kDETPropertyBusy`	–15020	Specified property is being edited

SEE ALSO

You can use the `kDETcmdGetPropertyBinary` callback routine (page 5-219) to determine the value of a binary property.

To cause the CE to save the new property value, call the `kDETcmdSetPropertyChanged` callback routine (described next).

The `RString` data structure is defined in the chapter "AOCE Utilities" in this book.

kDETcmdSetPropertyChanged

This callback routine sets or clears the property-changed flag for a specified property.

```
struct DETSetPropertyChangedBlock {
            DETCallBackBlockPropertyHeader
            Boolean propertyChanged;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdSetPropertyChanged
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
→	propertyChanged	Boolean	Property-changed flag

DESCRIPTION

If you set the `propertyChanged` field to `true`, the Catalogs Extension saves the property the next time the user closes the information page. Note that setting the value of a property does not automatically set its changed flag.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETUnableToAccessProperty	–15011	Property could not be found
kDETPropertyBusy	–15020	Specified property is being edited

SEE ALSO

You can use the `kDETcmdGetPropertyChanged` callback routine (page 5-221) to determine the current value of a property's changed flag.

kDETcmdSetPropertyEditable

This callback routine sets the property-editable flag for a specific property.

```
struct DETSetPropertyEditableBlock {
            DETCallBackBlockPropertyHeader
            Boolean propertyEditable;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdSetPropertyEditable
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
→	propertyEditable	Boolean	Property-editable flag

DESCRIPTION

The Catalogs Extension normally sets the value of the property-editable flag for a property based on the user's authentication identity and the access control settings of the dNode (catalog folder), record, and attribute. The property-editable flag determines whether an edit-text view is editable or a control in an information page is enabled. You can set the `propertyEditable` field to `false` to disable an edit-text view or a control, overriding the default setting, or to `true` to reenable a view or control once you have disabled it.

The setting of the `propertyEditable` flag persists only as long as the aspect remains in memory.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETUnableToAccessProperty	–15011	Property could not be found
kDETPropertyBusy	–15020	Specified property is being edited

SEE ALSO

You can use the `kDETcmdGetPropertyEditable` callback routine (page 5-222) to determine the current setting of the property-editable flag.

kDETcmdDirtyProperty

This callback routine causes the CE to redraw a view and calls the code resource for the target with the `kDETcmdPropertyDirtied` routine selector.

```
struct DETDirtyPropertyBlock {
            DETCallBackBlockPropertyHeader
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdDirtyProperty`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number

DESCRIPTION

When you make a change that affects the view associated with a property (by adding an item to a pop-up menu, for example), you can call the `kDETcmdDirtyProperty` callback routine to cause the Catalogs Extension to redraw the views associated with the property. This routine also calls the code resource for the target with the `kDETcmdPropertyDirtied` routine selector, giving you the opportunity to redraw other views affected by the views that were just redrawn.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found
`kDETPropertyBusy`	–15020	Specified property is being edited

SEE ALSO

Calling the `kDETcmdDirtyProperty` routine does not cause the CE to save a property; when you change the value of a property, call the `kDETcmdSetPropertyChanged` routine (page 5-231) to cause the CE to save the new value.

kDETcmdSaveProperty

This callback routine saves the value of the specified property.

```
struct DETSavePropertyBlock {
            DETCallBackBlockPropertyHeader
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdSaveProperty`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number

DESCRIPTION

Normally, the Catalogs Extension saves all changed property values (that is, all property values for which the changed flag is set) when the user closes the information page. You can use the `kDETcmdSaveProperty` callback routine to force the CE to save a specific property immediately. The CE applies all the appropriate lookup-table patterns and writes the property values to the attributes specified by the lookup table.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Could not find or change property

SEE ALSO

Lookup tables are described in “The Lookup-Table Resource” beginning on page 5-105.

Working With Sublists

The routines in this section return information about sublists and force the Catalogs Extension to update a sublist. Note that the CE looks up information in catalogs asynchronously. Thus, it might not have finished setting up a sublist because it has not had time to complete its search. You can use the value of the property `kDETPastFirstLookup` to determine whether the CE has completed its catalog search. This property equals 0 until the search is complete, after which it equals 1.

kDETcmdSublistCount

This callback routine returns the number of items in the targeted aspect’s sublist.

```
struct DETSublistCountBlock {
            DETCallBackBlockTargetedHeader
            long count;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdSublistCount`
→	`target`	`DETTargetSpecification`	Target specifier
←	`count`	`long`	The number of items in the targeted aspect’s sublist

DESCRIPTION

You can use this routine to determine the total number of items in your aspect’s sublist when you are using a targeted callback routine to iterate through every item in the sublist.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect

SEE ALSO

The target-specifier structure requires you to specify the index number of a sublist item. The target specifier is described in "Target Specifier" on page 5-142.

Use the kDETcmdSelectedSublistCount callback routine (described next) to determine the number of selected items in the sublist.

kDETcmdSelectedSublistCount

This callback routine returns the number of items that the user has selected in the targeted aspect's sublist.

```
struct DETSelectedSublistCountBlock {
            DETCallBackBlockTargetedHeader
            long count;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdSelectedSublistCount
→	target	DETTargetSpecification	Target specifier
←	count	long	The number of selected items in the targeted aspect's sublist

DESCRIPTION

You can use this routine to determine the number of selected items in your aspect's sublist when you are using a targeted callback routine to iterate through all the selected items in the sublist.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect

SEE ALSO

The target-specifier structure requires you to specify the index number of a sublist item. The target specifier is described in "Target Specifier" on page 5-142.

Use the kDETcmdSublistCount callback routine (page 5-235) to determine the total number of items in the sublist.

kDETcmdRequestSync

This callback routine causes the CE to synchronize a sublist and properties with the catalog system.

```
struct DETRequestSyncBlock {
            DETCallBackBlockTargetedHeader
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdRequestSync
→	target	DETTargetSpecification	Target specifier

DESCRIPTION

This routine forces the Catalogs Extension to check immediately whether the sublist or any properties in the targeted aspect need updating to match what's present in the catalog system. Normally, the CE performs this operation periodically. You can use this callback routine if you need the synchronization done immediately; for example, if you use a Catalog Manager function to add something to the sublist and want it displayed without delay.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect

SEE ALSO

When you call the kDETcmdRequestSync callback routine, the CE calls your kDETcmdShouldSync routine (page 5-185) to determine whether any of your properties need updating.

Working With Pop-Up Menus

The commands in this section add and remove dynamic pop-up menu items and return the text of a pop-up menu item.

kDETcmdAddMenu

This callback routine adds an item to a dynamic pop-up menu.

```
struct DETAddMenuBlock {
            DETCallBackBlockPropertyHeader
            RString* name;
            long parameter;
            long addAfter;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdAddMenu
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
→	name	RString*	Pointer to name of new menu item
→	parameter	long	Parameter to return to code resource when this item is selected
→	addAfter	long	Parameter of menu item to add this item after, or –1 to add item at end of menu

DESCRIPTION

Provide a pointer to the text for the new menu item in the `name` field. The Catalogs Extension sends the value in the `parameter` field to your code resource as a parameter to the `kDETcmdPropertyCommand` routine when the user chooses this menu item. Use the `addAfter` parameter to indicate where in the menu to add the item: immediately after the menu item whose parameter you specify, or at the end of the menu if you specify –1.

SPECIAL CONSIDERATIONS

You cannot call this routine for a menu that is not visible: the information page must be open and, if the menu is in a conditional view, that view must be currently drawn.

If you have a dynamic pop-up menu in a conditional view, you must set up the menu each time the conditional view appears.

Pop-up menus are limited to 31 items. If you try to add more than 31 items, the `kDETcmdAddMenu` callback routine returns the `kDETCouldNotAddMenuItem` result code.

RESULT CODES

`noErr`	0	No error
`kDETInvalidTargetAspectName`	–15000	Could not find aspect named in target selector
`kDETInvalidTargetItemNumber`	–15001	Item number in target selector out of range
`kDETInvalidTargetFromNonAspect`	–15002	Targeted item doesn't have an aspect
`kDETInvalidTargetDSSpec`	–15003	`DSSpec` in target selector could not be resolved
`kDETUnknownTargetSelector`	–15004	Selector type in target selector invalid
`kDETInvalidTarget`	–15005	Target selector invalid
`kDETTargetNotAnAspect`	–15006	Specified target object not an aspect
`kDETUnableToAccessProperty`	–15011	Property could not be found
`kDETInfoPageNotOpen`	–15012	Information page not open
`kDETNoSuchView`	–15013	No view found with specified property number
`kDETCouldNotAddMenuItem`	–15014	Could not add item to menu
`kDETCouldNotFindMenuItem`	–15016	Could not find menu item

SEE ALSO

Use the `kDETcmdDirtyProperty` callback routine (page 5-233) to cause the CE to redraw the menu when you add a new menu item.

Pop-up menus are described in "View Lists" beginning on page 5-123.

kDETcmdRemoveMenu

This callback routine removes an item from a dynamic pop-up menu.

```
struct DETRemoveMenuBlock {
            DETCallBackBlockPropertyHeader
            long itemToRemove;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdRemoveMenu
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
→	itemToRemove	long	Parameter of menu item to remove

DESCRIPTION

This routine removes the item that has the parameter value specified in the `itemToRemove` field.

SPECIAL CONSIDERATIONS

You cannot call this routine for a menu that is not visible: the information page must be open and, if the menu is in a conditional view, that view must be currently drawn.

If you have a dynamic pop-up menu in a conditional view, you must set up the menu each time the conditional view appears.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETUnableToAccessProperty	–15011	Property could not be found
kDETInfoPageNotOpen	–15012	Information page not open

kDETNoSuchView	–15013	No view found with specified property number
kDETCouldNotRemoveMenuItem	–15015	Could not remove item from dynamic menu
kDETCouldNotFindMenuItem	–15016	Could not find menu item

SEE ALSO

Use the kDETcmdDirtyProperty callback routine (page 5-233) to cause the CE to redraw the menu when you remove a menu item.

Pop-up menus are described in "View Lists" beginning on page 5-123.

kDETcmdMenuItemRString

This callback routine returns the text of an item in a dynamic pop-up menu.

```
struct DETMenuItemRStringBlock {
            DETCallBackBlockPropertyHeader
            long itemParameter;
            RStringHandle rString;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdMenuItemRString
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
→	itemParameter	long	Parameter of menu item for which you want the text string
←	rString	RStringHandle	Handle to string containing text of menu item

DESCRIPTION

Use the itemParameter field to specify the parameter of the menu item whose text you want. The Catalogs Extension allocates the handle for the rString field. It is your responsibility to deallocate the handle when it is no longer needed.

SPECIAL CONSIDERATIONS

You cannot call this routine for a menu that is not visible: the information page must be open and, if the menu is in a conditional view, that view must be currently drawn.

If you have a dynamic pop-up menu in a conditional view, you must set up the menu each time the conditional view appears.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETUnableToAccessProperty	–15011	Property could not be found
kDETInfoPageNotOpen	–15012	Information page not open
kDETNoSuchView	–15013	No view found with specified property number
kDETCouldNotFindMenuItem	–15016	Could not find menu item

SEE ALSO

Pop-up menus are described in "View Lists" beginning on page 5-123.

Custom Views

The routines in this section return information about custom views. The first routine, kDETcmdGetCustomViewUserReference, returns the reference value associated with a custom view. The kDETcmdGetCustomViewBounds routine returns the bounds for a custom view.

kDETcmdGetCustomViewUserReference

This callback routine returns the reference value that you set in the view list for a custom view.

```
struct DETGetCustomViewUserReferenceBlock {
            DETCallBackBlockPropertyHeader
            short userReference;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdGetCustomViewUserReference
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
←	userReference	short	User reference value

DESCRIPTION

The view list specification for a custom view includes an integer that you can set to any value you wish. The `kDETcmdGetCustomViewUserReference` callback routine returns this value for a specific custom view.

SPECIAL CONSIDERATIONS

You cannot call this routine for a custom view that is not visible: the information page must be open and, if the custom view is in a conditional view, that view must be currently drawn.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETUnableToAccessProperty	–15011	Property could not be found
kDETInfoPageNotOpen	–15012	Information page not open
kDETNoSuchView	–15013	No view found with specified property number
kDETCouldNotFindCustomView	–15017	Could not find custom view

SEE ALSO

The view list specifier for a custom view is described in "View Lists" beginning on page 5-123.

For more information about how to implement custom views, see "Custom Views and Custom Menus" beginning on page 5-192.

kDETcmdGetCustomViewBounds

This callback routine returns the bounds of a custom view.

```
struct DETGetCustomViewBoundsBlock {
            DETCallBackBlockPropertyHeader
            Rect bounds;
};
```

Parameter block

→	reqFunction	DETCallBackFunctions	kDETcmdGetCustomViewBounds
→	target	DETTargetSpecification	Target specifier
→	property	short	Property number
←	bounds	rect	Bounds of the view in local window coordinates

DESCRIPTION

You can use this routine to determine the bounds of a specific custom view so that you don't have to store the bounds for every custom view you define.

SPECIAL CONSIDERATIONS

You cannot call this routine for a custom view that is not visible: the information page must be open and, if the custom view is in a conditional view, that view must be currently drawn.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETUnableToAccessProperty	–15011	Property could not be found
kDETInfoPageNotOpen	–15012	Information page not open
kDETNoSuchView	–15013	No view found with specified property number
kDETCouldNotFindCustomView	–15017	Could not find custom view

SEE ALSO

The view list specifier for a custom view is described in "View Lists" beginning on page 5-123.

For more information about how to implement custom views, see "Custom Views and Custom Menus" beginning on page 5-192.

Sending a Property Command

The `kDETcmdDoPropertyCommand` callback routine sends a property command to a code resource.

kDETcmdDoPropertyCommand

This callback routine sends a property command to the targeted code resource.

```
struct DETDoPropertyCommandBlock {
            DETCallBackBlockPropertyHeader
            long parameter;
};
```

Parameter block

→	`reqFunction`	`DETCallBackFunctions`	`kDETcmdDoPropertyCommand`
→	`target`	`DETTargetSpecification`	Target specifier
→	`property`	`short`	Property number
→	`parameter`	`long`	Parameter of property command

DESCRIPTION

When you call this routine, the Catalogs Extension calls your code resource's property command (`kDETcmdPropertyCommand`) routine. The CE passes the property number and parameter value you specify to your property command. The effect is the same as when the CE initiates a property command.

RESULT CODES

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETUnableToAccessProperty	–15011	Property could not be found

SEE ALSO

The kDETcmdPropertyCommand routine is described on page 5-159.

Summary of AOCE Templates

C Summary

Constants and Data Types

```
/* Current versions of all the different template types */

#define kDETAspectVersion     -976
#define kDETInfoPageVersion   -976
#define kDETKillerVersion     -976
#define kDETForwarderVersion  -976
#define kDETFileTypeVersion   -976

/* Suggested separation for template IDs within a file */
#define kDETIDSep             250

/* Predefined base IDs */
#define kDETFirstID           (1000)
#define kDETSecondID          (1000 + kDETIDSep)
#define kDETThirdID           (1000 + 2 * kDETIDSep)
#define kDETFourthID          (1000 + 3 * kDETIDSep)
#define kDETFifthID           (1000 + 4 * kDETIDSep)

/* Template resource ID offsets */
#define kDETTemplateName               0
#define kDETRecordType                 1
#define kDETKillerName                 1
#define kDETAttributeType              2
#define kDETAttributeValueTag          3
#define kDETAspectCode                 4
#define kDETInfoPageName               4
#define kDETForwarderTemplateNames     4
#define kDETAspectMainBitmap           5
#define kDETInfoPageMainViewAspect     5
#define kDETAspectName                 6
#define kDETInfoPageMenuName           6
#define kDETAspectCategory             7
#define kDETInfoPageMenuEntries        7
```

```
#define kDETAspectExternalCategory     8
#define kDETAspectKind                 9
#define kDETAspectGender               10
#define kDETAspectWhatIs               11
#define kDETAspectAliasKind            12
#define kDETAspectAliasGender          13
#define kDETAspectAliasWhatIs          14
#define kDETAspectBalloons             15
#define kDETAspectNewMenuName          16
#define kDETAspectNewEntryName         17
#define kDETAspectNewValue             18
#define kDETAspectSublistOpenOnNew     19
#define kDETAspectLookup               20
#define kDETAspectDragInString         21
#define kDETAspectDragInVerb           22
#define kDETAspectDragInSummary        23
#define kDETAspectRecordDragIn         24
#define kDETAspectRecordCatDragIn      25
#define kDETAspectAttrDragIn           26
#define kDETAspectAttrDragOut          27
#define kDETAspectViewMenu             28
#define kDETAspectReverseSort          29
#define kDETAspectInfoPageCustomWindow 30

/* Properties */
#define kDETNoProperty                 -1
#define kDETFirstLocalProperty         0
#define kDETLastLocalProperty          (kDETFirstLocalProperty + 249)
#define kDETFirstDevProperty           40
#define kDETFirstConstantProperty      250
#define kDETLastConstantProperty       (kDETFirstConstantProperty + 249)
#define kDETConstantProperty           kDETFirstConstantProperty
#define kDETZeroProperty               (kDETConstantProperty + 0)
#define kDETOneProperty                (kDETConstantProperty + 1)
#define kDETFalseProperty              (kDETConstantProperty + 0)
#define kDETTrueProperty               (kDETConstantProperty + 1)

/* Name and kind properties */
#define kDETPrName                     3050
#define kDETPrKind                     3051

#define kDETPastFirstLookup         26550
#define kDETInfoPageNumber          27050
#define kDETAspectTemplateNumber    26551
```

```
#define kDETInfoPageTemplateNumber  26552
#define kDETOpenSelectedItems       26553 /* open selected sublist items */
#define kDETAddNewItem              26554 /* add new sublist item */
#define kDETRemoveSelectedItems     26555 /* remove selected sublist items */

/* Access masks */
#define kDETDNodeAccessMask         25825 /* the DNode access mask */
#define kDETRecordAccessMask        25826 /* the record access mask */
#define kDETAttributeAccessMask     25827 /* the attribute access mask */
#define kDETPrimaryMaskByBit        25828 /* a set of 16 properties
                                             to access all bits of the
                                             primary mask */

#define kDETPrimarySeeMask          kDETPrimaryMaskByBit
#define kDETPrimaryAddMask          (kDETPrimaryMaskByBit + 1)
#define kDETPrimaryDeleteMask       (kDETPrimaryMaskByBit + 2)
#define kDETPrimaryChangeMask       (kDETPrimaryMaskByBit + 3)
#define kDETPrimaryRenameMask       (kDETPrimaryMaskByBit + 4)
#define kDETPrimaryChangePrivsMask  (kDETPrimaryMaskByBit + 5)
#define kDETPrimaryTopMaskBit       (kDETPrimaryMaskByBit + 15)

/* Property types */
#define kDETPrTypeNumber            -1    /* a number */
#define kDETPrTypeString            -2    /* a string */
#define kDETPrTypeBinary            -3    /* a binary block */

/* Rez-compatible attribute-tag types */
#define typeRString                 'rstr'
#define typePackedDSSpec            'dspc'
#define typeBinary                  'bnry'

/* Constants used in view lists */
#define kDETNoFlags           0
#define kDETEnabled           (1 << 0) /* main view field enabled */

#define kDETHilightIfSelected (1 << 0) /* hilight when entry is selected */

#define kDETNumericOnly       (1 << 3) /* allow digits only */
#define kDETMultiLine         (1 << 4) /* allow multiple lines in view */
#define kDETDynamicSize       (1 << 9) /* don't draw box around text
                                          until user clicks in it,
                                          then auto-size it */
#define kDETAllowNoColons     (1 << 10)/* don't allow colons */
```

```
#define kDETPopupDynamicSize  (1 << 8) /* automatically resize pop-up */

#define kDETScaleToView       (1 << 8) /* scale picture to view bounds */

#define kDETLargeIcon              0
#define kDETSmallIcon              1
#define kDETMiniIcon               2

#define kDETLeft                   0
#define kDETCenter                 1
#define kDETRight                  -1
#define kDETForceLeft              -2

#define kDETUnused                 0
#define kDETBoxTakesContentClicks  (1 << 0)
#define kDETBoxIsRounded           (1 << 1)
#define kDETBoxIsGrayed            (1 << 2)
#define kDETBoxIsInvisible         (1 << 3)

#define kDETApplicationFont         1
#define kDETApplicationFontSize     9
#define kDETAppFontLineHeight      12

#define kDETSystemFont              0
#define kDETSystemFontSize         12
#define kDETSystemFontLineHeight   16

#define kDETDefaultFont             1
#define kDETDefaultFontSize         9
#define kDETDefaultFontLineHeight  12

#define kDETNormal                 0
#define kDETBold                   1
#define kDETItalic                 2
#define kDETUnderline              4
#define kDETOutline                8
#define kDETShadow                 0x10
#define kDETCondense               0x20
#define kDETExtend                 0x40

#define kDETIconStyle              -3 /* normal text style for
                                         regular sublist entries,
```

```
                                        italic text style for aliases */

#define kDETChangeViewCommand       'view'   /* change the view */

/* Default information-page layouts */

/* Default record information-page size */
#define kDETRecordInfoWindHeight    228
#define kDETRecordInfoWindWidth     400

/* Default attribute information-page size */
#define kDETAttributeInfoWindHeight 250
#define kDETAttributeInfoWindWidth  230

/* Page identifying icon */
#define kDETSubpageIconTop          8
#define kDETSubpageIconLeft         8
#define kDETSubpageIconBottom       (kDETSubpageIconTop + 32)
#define kDETSubpageIconRight        (kDETSubpageIconLeft + 32)
#define kDETSubpageIconRect         {kDETSubpageIconTop,\
                                     kDETSubpageIconLeft,\
                                     kDETSubpageIconBottom,\
                                     kDETSubpageIconRight}

/* The following rectangle can be used in a 'deti' with no sublist: */
#define kDETNoSublistRect           {0, 0, 0, 0}

/* Reserved category names */
#define kDETCategoryAllItems     "aoce All Items"     /* everything */
#define kDETCategoryAddressItems "aoce Address Items" /* all addresses */
#define kDETCategoryMisc         "aoce Miscellaneous" /* things that
                                         don't have their own category */

/* Target selectors */
enum DETTargetSelector {
   kDETSelf = 0,              /* the "current" item */
   kDETSelfOtherAspect,       /* another aspect of the current item */
   kDETParent,                /* the parent of the current item */
   kDETSublistItem,           /* the ith item in the sublist */
   kDETSelectedSublistItem,   /* the ith selected item in the sublist */
   kDETDSSpec,                /* DSSpec */
   kDETAspectTemplate,        /* specific aspect template */
   kDETInfoPageTemplate,      /* specific info-page template */
   kDETHighSelector = 0xF000  /* force type to be short */
```

```
};

typedef enum DETTargetSelector DETTargetSelector;

/* Return value for code resources */
#define kDETDidNotHandle 1

/* Valid commandIDs for DETDropQueryBlock and DETDropMeQueryBlock (in
    addition to property numbers) */
#define kDETDoNothing   'xxx0'
#define kDETMove        'move'
#define kDETDrag        'drag'
#define kDETAlias       'alis'

/* Item types */
enum DETItemType {
   kDETHFSType = 0,             /* HFS item type */
   kDETDSType,                  /* catalog service item type */
   kDETMailType,                /* mail (letter) item type */
   kDETMoverType,               /* sounds, fonts, etc., from inside
                                    a suitcase or system file */
   kDETLastItemType = 0xF0000000 /* force itemType to be a long */
};

typedef enum DETItemType DETItemType;

struct DETFSInfo {
   OSType fileType;        /* file type */
   OSType fileCreator;     /* file creator */
   unsigned short fdFlags; /* Finder flags */
   FSSpec fsSpec;          /* FSSpec */
};

typedef struct DETFSInfo DETFSInfo;

struct {
   PackedDSSpecPtr* dsSpec;   /* DSSpec for item */
   short refNum;              /* refnum for returned address */
   AuthIdentity identity;     /* identity for returned address */
} ds;

/* Application-defined routines */
enum DETCallFunctions {

   kDETcmdSimpleCall = 0,
```

```
    kDETcmdInit,
    kDETcmdExit,
    kDETcmdAttributeCreation,
    kDETcmdDynamicForwarders,

    kDETcmdTargetedCall = 1000,
    kDETcmdInstanceInit,
    kDETcmdInstanceExit,
    kDETcmdIdle,
    kDETcmdViewListChanged,
    kDETcmdValidateSave,
    kDETcmdDropQuery,
    kDETcmdDropMeQuery,
    kDETcmdAttributeNew,
    kDETcmdAttributeChange,
    kDETcmdAttributeDelete,
    kDETcmdItemNew,
    kDETcmdOpenSelf,
    kDETcmdDynamicResource,
    kDETcmdShouldSync,
    kDETcmdDoSync,

    kDETcmdPropertyCall = 2000,
    kDETcmdPropertyCommand,
    kDETcmdMaximumTextLength,
    kDETcmdPropertyDirtied,
    kDETcmdPatternIn,
    kDETcmdPatternOut,
    kDETcmdConvertToNumber,
    kDETcmdConvertToRString,
    kDETcmdConvertFromNumber,
    kDETcmdConvertFromRString,
    kDETcmdCustomViewDraw,
    kDETcmdCustomViewMouseDown,
    kDETcmdKeyPress,
    kDETcmdPaste,
    kDETcmdCustomMenuSelected,
    kDETcmdCustomMenuEnabled,

    kDETcmdHighCall = 0xF0000000/* force the type to be long */
};

typedef enum DETCallFunctions DETCallFunctions;
```

```
/* Callback functions */
enum DETCallBackFunctions {

   kDETcmdSimpleCallback = 0,
   kDETcmdBeep,
   kDETcmdBusy,
   kDETcmdChangeCallFors,
   kDETcmdGetCommandSelectionCount,
   kDETcmdGetCommandItemN,
   kDETcmdOpenDSSpec,
   kDETcmdAboutToTalk,
   kDETcmdUnloadTemplates,
   kDETcmdTemplateCounts,

   kDETcmdTargetedCallback = 1000,
   kDETcmdGetDSSpec,
   kDETcmdSublistCount,
   kDETcmdSelectedSublistCount,
   kDETcmdRequestSync,
   kDETcmdBreakAttribute,
   kDETcmdGetTemplateFSSpec,
   kDETcmdGetOpenEdit,
   kDETcmdCloseEdit,

   kDETcmdPropertyCallback = 2000,
   kDETcmdGetPropertyType,
   kDETcmdGetPropertyNumber,
   kDETcmdGetPropertyRString,
   kDETcmdGetPropertyBinarySize,
   kDETcmdGetPropertyBinary,
   kDETcmdGetPropertyChanged,
   kDETcmdGetPropertyEditable,
   kDETcmdSetPropertyType,
   kDETcmdSetPropertyNumber,
   kDETcmdSetPropertyRString,
   kDETcmdSetPropertyBinary,
   kDETcmdSetPropertyChanged,
   kDETcmdSetPropertyEditable,
   kDETcmdDirtyProperty,
   kDETcmdDoPropertyCommand,
   kDETcmdAddMenu,
   kDETcmdRemoveMenu,
   kDETcmdMenuItemRString,
```

```
   kDETcmdSaveProperty,
   kDETcmdGetCustomViewUserReference,
   kDETcmdGetCustomViewBounds,
   kDETcmdGetResource,

   kDETcmdHighCallback = 0xF0000000          /* force type to be long */
};

typedef enum DETCallBackFunctions DETCallBackFunctions;
```

Target Specifier

```
struct DETTargetSpecification
{
      DETTargetSelector selector;   /* target selector */
      RStringPtr aspectName;        /* aspect name */
      long itemNumber;              /* sublist index number */
      PackedDSSpecPtr dsSpec;       /* DSSpec */
};

typedef struct DETTargetSpecification DETTargetSpecification;
```

Forwarder List

```
struct DETForwarderListItem {
   struct DETForwarderListItem** next;/* handle to next item, or nil */
   AttributeTag attributeValueTag;    /* attribute value tag (0 for none) */
   PackedPathName rstrs;              /* forwarder list */
};
```

Call Block Headers

```
#define DETCallBlockHeader \
   DETCallFunctions reqFunction; /* requested function */\
   DETCallBack callBack;         /* pointer to callback routine */\
   long callBackPrivate;         /* private data for the callback routine */\
   long templatePrivate;         /* private data stored in template */

#define DETCallBlockTargetedHeader \
   DETCallFunctions reqFunction; /* requested function */\
   DETCallBack callBack;         /* pointer to callback routine */\
   long callBackPrivate;         /* private data for the callback routine */\
   long templatePrivate;         /* private data stored in template */\
```

```
   long instancePrivate;         /* private data stored in aspect */\
   DETTargetSpecification target;/* the target (originator) of the call */\
   Boolean targetIsMainAspect;   /* true if the target is the main aspect */

#define DETCallBlockPropertyHeader \
   DETCallFunctions reqFunction; /* requested function */\
   DETCallBack callBack;         /* pointer to callback routine */\
   long callBackPrivate;         /* private data for the callback routine */\
   long templatePrivate;         /* private data stored in template */\
   long instancePrivate;         /* private data stored in aspect */\
   DETTargetSpecification target;/* the target (originator) of the call */\
   Boolean targetIsMainAspect;   /* true if the target is the main aspect */\
   short property;               /* the property number the call refers to */

struct DETProtoCallBlock {
   DETCallBlockPropertyHeader
};

typedef struct DETProtoCallBlock DETProtoCallBlock;
```

Call Block Union Structure

```
union DETCallBlock {
   DETProtoCallBlock               protoCall;
   DETInitBlock                    init;
   DETExitBlock                    exit;
   DETInstanceInitBlock            instanceInit;
   DETInstanceExitBlock            instanceExit;
   DETInstanceIdleBlock            instanceIdle;
   DETPropertyCommandBlock         propertyCommand;
   DETMaximumTextLengthBlock       maximumTextLength;
   DETViewListChangedBlock         viewListChanged;
   DETPropertyDirtiedBlock         propertyDirtied;
   DETValidateSaveBlock            validateSave;
   DETDropQueryBlock               dropQuery;
   DETDropMeQueryBlock             dropMeQuery;
   DETAttributeCreationBlock       attributeCreationBlock;
   DETAttributeNewBlock            attributeNew;
   DETAttributeChangeBlock         attributeChange;
   DETAttributeDeleteBlock         attributeDelete;
   DETItemNewBlock                 itemNew;
   DETPatternInBlock               patternIn;
   DETPatternOutBlock              patternOut;
   DETShouldSyncBlock              shouldSync;
```

```
   DETDoSyncBlock                  doSync;
   DETOpenSelfBlock                openSelf;
   DETConvertToNumberBlock         convertToNumber;
   DETConvertToRStringBlock        convertToRString;
   DETConvertFromNumberBlock       convertFromNumber;
   DETConvertFromRStringBlock      convertFromRString;
   DETCustomViewDrawBlock          customViewDraw;
   DETCustomViewMouseDownBlock     customViewMouseDown;
   DETKeyPressBlock                keyPress;
   DETPasteBlock                   paste;
   DETCustomMenuSelectedBlock      customMenuSelected;
   DETCustomMenuEnabledBlock       customMenuEnabled;
   DETDynamicForwardersBlock       dynamicForwarders;
   DETDynamicResourceBlock         dynamicResource;
};

typedef union DETCallBlock DETCallBlock;
typedef DETCallBlock* DETCallBlockPtr;
```

Callback Block Headers

```
#define DETCallBackBlockHeader \
   DETCallBackFunctions reqFunction;   /* requested function */

#define DETCallBackBlockTargetedHeader \
   DETCallBackFunctions reqFunction;   /* requested function */\
   DETTargetSpecification target;      /* the target for the request */

#define DETCallBackBlockPropertyHeader \
   DETCallBackFunctions reqFunction;   /* requested function */\
   DETTargetSpecification target;      /* the target for the request */\
   short property;                     /* the property to apply the
                                          request to */

struct DETProtoCallBackBlock {
   DETCallBackBlockPropertyHeader
};

typedef struct DETProtoCallBackBlock DETProtoCallBackBlock;
```

Callback Block Union Structure

```
union DETCallBackBlock {
   DETProtoCallBackBlock               protoCallBack;
   DETBeepBlock                        beep;
   DETBusyBlock                        busy;
   DETChangeCallForsBlock              changeCallFors;
   DETGetCommandSelectionCountBlock    getCommandSelectionCount;
   DETGetCommandItemNBlock             getCommandItemN;
   DETGetDSSpecBlock                   getDSSpec;
   DETGetTemplateFSSpecBlock           getTemplateFSSpec;
   DETGetOpenEditBlock                 getOpenEdit;
   DETCloseEditBlock                   closeEdit;
   DETGetPropertyTypeBlock             getPropertyType;
   DETGetPropertyNumberBlock           getPropertyNumber;
   DETGetPropertyRStringBlock          getPropertyRString;
   DETGetPropertyBinarySizeBlock       getPropertyBinarySize;
   DETGetPropertyBinaryBlock           getPropertyBinary;
   DETGetPropertyChangedBlock          getPropertyChanged;
   DETGetPropertyEditableBlock         getPropertyEditable;
   DETSetPropertyTypeBlock             setPropertyType;
   DETSetPropertyNumberBlock           setPropertyNumber;
   DETSetPropertyRStringBlock          setPropertyRString;
   DETSetPropertyBinaryBlock           setPropertyBinary;
   DETSetPropertyChangedBlock          setPropertyChanged;
   DETSetPropertyEditableBlock         setPropertyEditable;
   DETDirtyPropertyBlock               dirtyProperty;
   DETDoPropertyCommandBlock           doPropertyCommand;
   DETSublistCountBlock                sublistCount;
   DETSelectedSublistCountBlock        selectedSublistCount;
   DETRequestSyncBlock                 requestSync;
   DETAddMenuBlock                     addMenu;
   DETRemoveMenuBlock                  removeMenu;
   DETMenuItemRStringBlock             menuItemRString;
   DETOpenDSSpecBlock                  openDSSpec;
   DETAboutToTalkBlock                 aboutToTalk;
   DETBreakAttributeBlock              breakAttribute;
   DETSavePropertyBlock                saveProperty;
   DETGetCustomViewUserReferenceBlock  getCustomViewUserReference;
   DETGetCustomViewBoundsBlock         getCustomViewBounds;
   DETGetResourceBlock                 getResource;
   DETTemplateCounts                   templateCounts;
   DETUnloadTemplatesBlock             unloadTemplates;
};
```

```
typedef union DETCallBackBlock DETCallBackBlock;
typedef DETCallBackBlock* DETCallBackBlockPtr;

typedef pascal OSErr (*DETCallBack) (union DETCallBlock* callBlockPtr,
                                     DETCallBackBlockPtr callBackBlockPtr);
```

Call-For Mask

```
/* Call-for list: */

#define kDETCallForOther         1     /* call for events not listed below */
#define kDETCallForIdle          2     /* kDETcmdIdle */
#define kDETCallForCommands      4     /* kDETcmdPropertyCommand,
                                          kDETcmdSelfOpen */
#define kDETCallForViewChanges   8     /* kDETcmdViewListChanged
                                          kDETcmdPropertyDirtied,
                                          kDETcmdMaximumTextLength */
#define kDETCallForDrops         0x10  /* kDETcmdDropQuery,
                                          kDETcmdDropMeQuery */
#define kDETCallForAttributes    0x20  /* kDETcmdAttributeCreation,
                                          kDETcmdAttributeNew,
                                          kDETcmdAttributeChange,
                                          kDETcmdAttributeDelete */
#define kDETCallForValidation    0x40  /* kDETcmdValidateSave */
#define kDETCallForKeyPresses    0x80  /* kDETcmdKeyPress and
                                          kDETcmdPaste */
#define kDETCallForSyncing       0x200 /* kDETcmdShouldSync, kDETcmdDoSync */
#define kDETCallForResources     0x100 /* kDETcmdDynamicResource */
#define kDETCallForEscalation    0x8000/* all calls escalated to the
                                          next level */

#define kDETCallForNothing       0     /* do not call */
#define kDETCallForEverything    0xFFFFFFFF  /* all of the above */

typedef pascal OSErr (*DETCall) (DETCallBlockPtr callBlockPtr);
```

Functions You Can Provide as Part of Your Code Resource

Initializing and Removing Templates

```
struct DETInitBlock {
            DETCallBlockHeader
            long newCallFors;
};

struct DETExitBlock{
            DETCallBlockHeader
};

struct DETInstanceInitBlock {
            DETCallBlockTargetedHeader
};

struct DETItemNewBlock{
            DETCallBlockTargetedHeader
};

struct DETInstanceExitBlock {
            DETCallBlockTargetedHeader
};
```

Dynamic Creation of Resources

```
struct DETDynamicForwardersBlock {
            DETCallBlockHeader
            DETForwarderListHandle forwarders;
};

struct DETDynamicResourceBlock {
            DETCallBlockTargetedHeader
            ResType resourceType;
            short propertyNumber;
            short resourceID;
            Handle theResource;
};
```

Processing Idle-Time Tasks

```
struct DETcmdInstanceIdleBlock {
            DETCallBlockTargetedHeader
};
```

Property and Information Page Routines

```
struct DETOpenSelfBlock {
            DETCallBlockTargetedHeader
            short modifiers;
};

struct DETPropertyCommandBlock {
            DETCallBlockPropertyHeader
            long parameter;
};

struct DETKeyPressBlock {
            DETCallBlockPropertyHeader
            EventRecord *theEvent;
};

struct DETPasteBlock {
            DETCallBlockPropertyHeader
            short modifiers;
};

struct DETMaximumTextLengthBlock {
            DETCallBlockPropertyHeader
            long MaxSize;
};

struct DETViewListChangedBlock {
            DETCallBlockTargetedHeader
};

struct DETPropertyDirtiedBlock {
            DETCallBlockPropertyHeader
};

struct DETValidateSaveBlock {
            DETCallBlockTargetedHeader
            RStringHandle errorString;
};
```

Supporting Drops

```
struct DETDropMeQueryBlock {
            DETCallBlockTargetedHeader
            short modifiers;
```

```
            long commandID;
            AttributeType destinationType
            Boolean copyToHFS;
};

struct DETDropQueryBlock {
            DETCallBlockTargetedHeader
            short modifiers;
            long commandID;
            AttributeType destinationType
            Boolean copyToHFS;
};
```

Attribute-Related Commands

```
struct DETAttributeCreationBlock {
            DETCallBlockHeader
            PackedDSSpecPtr parent;
            short refNum;
            AuthIdentity identity;
            AttributeType attrType;
            AttributeTag attrTag;
            Handle value;
};

struct DETAttributeNewBlock {
            DETCallBlockTargetedHeader
            PackedDSSpecPtr parent;
            short refNum;
            AuthIdentity identity;
            AttributeType attrType;
            AttributeTag attrTag;
            Handle value;
};

struct DETAttributeChangeBlock {
            DETCallBlockTargetedHeader
            PackedDSSpecPtr parent;
            short refNum;
            AuthIdentity identity;
            AttributeType attrType;
            AttributeTag attrTag;
```

```
            AttributeCreationID attrCID;
            Handle value;
};

struct DETAttributeDeleteBlock {
            DETCallBlockTargetedHeader
            PackedDSSpecPtr dsSpec;
            short refNum;
            AuthIdentity identity;
};
```

Processing Custom Lookup-Table Pattern Elements

```
struct DETPatternInBlock {
            DETCallBlockPropertyHeader
            long elementType;
            long extra;
            AttributePtr attribute;
            long dataOffset;
            short bitOffset;
};

struct DETPatternOutBlock {
            DETCallBlockPropertyHeader
            long elementType;
            long extra;
            AttributePtr attribute;
            Handle data;
            long dataOffset;
            short bitOffset;
};
```

Synchronizing Property Values

```
struct DETShouldSyncBlock {
            DETCallBlockTargetedHeader
            Boolean shouldSync;
};

struct DETDoSyncBlock {
            DETCallBlockTargetedHeader;
};
```

Custom Property-Type Conversions

```
struct DETConvertToNumberBlock {
            DETCallBlockPropertyHeader
            long theValue;
};

struct DETConvertToRStringBlock {
            DETCallBlockPropertyHeader
            RStringHandle theValue;
};

struct DETConvertFromNumberBlock {
            DETCallBlockPropertyHeader
            long theValue;
};

struct DETConvertFromRStringBlock {
            DETCallBlockPropertyHeader
            RStringHandle theValue;
};
```

Custom Views and Custom Menus

```
struct DETGetCustomViewDrawBlock {
            DETCallBlockPropertyHeader
};

struct DETCustomViewMouseDownBlock {
            DETCallBlockPropertyHeader
            EventRecord *theEvent;
};

struct DETCustomMenuEnabledBlock {
            DETCallBlockTargetedHeader
            short menuTableParameter
            Boolean enable;
};

struct DETCustomMenuSelectedBlock {
            DETCallBlockTargetedHeader
            short menuTableParameter;
);
```

CE-Provided Functions That Your Code Resource Can Call

Calling CE-Provided Functions

```
CallBackDET(callBlockPtr, callBackBlockPtr);
```

Testing Your Code Resource

```
struct DETBeepBlock {
            DETCallBackBlockHeader
};
```

Changing the Call-For Mask

```
struct DETChangeCallForsBlock {
            DETCallBackBlockTargetedHeader
            long newCallFors;
};
```

Process Control

```
struct DETAboutToTalkBlock {
            DETCallBackBlockHeader
};

struct DETBusyBlock {
            DETCallBackBlockHeader;
};
```

Handling Drags and Drops

```
struct DETGetCommandSelectionCountBlock {
            DETCallBackBlockHeader;
            long count;
};

struct DETGetCommandItemNBlock {
      DETCallBackBlockHeader;
      long itemNumber;
      DETItemType itemType;
      union {
         DETFSInfo** fsInfo;
         struct {
                  PackedDSSpecPtr* dsSpec;
```

```
                short refNum;
                AuthIdentity identity;
          } ds;
          PackedDSSpecPtr* dsSpec;
          LetterSpec** ltrSpec;
       } item;
};
```

Working With Templates

```
struct DETTemplateCounts {
            DETCallBackBlockHeader
            long aspectTemplateCount;
            long infoPageTemplateCount;
};

struct DETGetTemplateFSSpecBlock {
            DETCallBackBlockTargetedHeader
            FSSpec fsSpec;
            short baseID;
            long aspectTemplateNumber;
};

struct DETGetResourceBlock {
            DETCallBackBlockPropertyHeader
            ResType resourceType;
            Handle theResource;
};

struct DETUnloadTemplatesBlock {
   DETCallBackBlockHeader
};
```

Working With Catalog Objects

```
struct DETGetDSSpecBlock {
            DETCallBackBlockTargetedHeader
            PackedDSSpecPtr* dsSpec;
            short refNum;
            AuthIdentity identity;
            Boolean isAlias;
            Boolean isRecordRef;
};
```

```
struct DETOpenDSSpecBlock {
            DETCallBackBlockHeader
            PackedDSSpecPtr dsSpec;
};
```

Edit-Text Routines

```
struct DETGetOpenEditBlock {
            DETCallBackBlockTargetedHeader
            short viewProperty;
};

struct DETCloseEditBlock {
            DETCallBackBlockTargetedHeader
};
```

Getting Information About Properties

```
struct DETGetPropertyTypeBlock {
            DETCallBackBlockPropertyHeader
            short propertyType;
};

struct DETGetPropertyNumberBlock {
            DETCallBackBlockPropertyHeader
            unsigned long propertyValue;
};

struct DETGetPropertyRStringBlock {
            DETCallBackBlockPropertyHeader
            RStringHandle propertyValue;
};

struct DETGetPropertyBinarySizeBlock {
            DETCallBackBlockPropertyHeader
            long propertyBinarySize;
};

struct DETGetPropertyBinaryBlock {
            DETCallBackBlockPropertyHeader
            Handle propertyValue;
};
```

```
struct DETGetPropertyChangedBlock {
            DETCallBackBlockPropertyHeader
            Boolean propertyChanged;
};

struct DETGetPropertyEditableBlock {
            DETCallBackBlockPropertyHeader
            Boolean propertyEditable;
};
```

Setting Value, Type, and Other Features of Properties

```
struct DETBreakAttributeBlock {
            DETCallBackBlockTargetedHeader
            AttributePtr breakAttribute;
            Boolean isChangeable;
};

struct DETSetPropertyTypeBlock {
            DETCallBackBlockPropertyHeader
            short newType;
};

struct DETSetPropertyNumberBlock {
            DETCallBackBlockPropertyHeader
            unsigned long newValue;
};

struct DETSetPropertyRStringBlock {
            DETCallBackBlockPropertyHeader
            RStringPtr newValue;
};

struct DETSetPropertyBinaryBlock {
            DETCallBackBlockPropertyHeader
            Ptr newValue;
            long newValueSize;
};

struct DETSetPropertyChangedBlock {
            DETCallBackBlockPropertyHeader
            Boolean propertyChanged;
};
```

```
struct DETSetPropertyEditableBlock {
            DETCallBackBlockPropertyHeader
            Boolean propertyEditable;
};

struct DETDirtyPropertyBlock {
            DETCallBackBlockPropertyHeader
};

struct DETSavePropertyBlock {
            DETCallBackBlockPropertyHeader
};
```

Working With Sublists

```
struct DETSublistCountBlock {
            DETCallBackBlockTargetedHeader
            long count;
};

struct DETSelectedSublistCountBlock {
            DETCallBackBlockTargetedHeader
            long count;
};

struct DETRequestSyncBlock {
            DETCallBackBlockTargetedHeader
};
```

Working With Pop-Up Menus

```
struct DETAddMenuBlock {
            DETCallBackBlockPropertyHeader
            RString* name;
            long parameter;
            long addAfter;
};

struct DETRemoveMenuBlock {
            DETCallBackBlockPropertyHeader
            long itemToRemove;
};
```

```
struct DETMenuItemRStringBlock {
            DETCallBackBlockPropertyHeader
            long itemParameter;
            RStringHandle rString;
};
```

Custom Views

```
struct DETGetCustomViewUserReferenceBlock {
            DETCallBackBlockPropertyHeader
            short userReference;
};

struct DETGetCustomViewBoundsBlock {
            DETCallBackBlockPropertyHeader
            Rect bounds;
};
```

Sending a Property Command

```
struct DETDoPropertyCommandBlock {
            DETCallBackBlockPropertyHeader
            long parameter;
};
```

Pascal Summary

Constants

```
{Current versions of all the different template types}

CONST
kDETAspectVersion             = -976;
kDETInfoPageVersion           = -976;
kDETKillerVersion             = -976;
kDETForwarderVersion          = -976;
kDETFileTypeVersion           = -976;

{Suggested separation for template IDs within a file}
kDETIDSep                     = 250
```

```
{Predefined base IDs}
kDETFirstID                  = (1000);
kDETSecondID                 = (1000 + kDETIDSep);
kDETThirdID                  = (1000 + 2 * kDETIDSep);
kDETFourthID                 = (1000 + 3 * kDETIDSep);
kDETFifthID                  = (1000 + 4 * kDETIDSep);

{Template resource ID offsets}
kDETTemplateName                      = 0;
kDETRecordType                        = 1;
kDETKillerName                        = 1;
kDETAttributeType                     = 2;
kDETAttributeValueTag                 = 3;
kDETAspectCode                        = 4;
kDETInfoPageName                      = 4;
kDETForwarderTemplateNames            = 4;
kDETAspectMainBitmap                  = 5;
kDETInfoPageMainViewAspect            = 5;
kDETAspectName                        = 6;
kDETInfoPageMenuName                  = 6;
kDETAspectCategory                    = 7;
kDETInfoPageMenuEntries               = 7;
kDETAspectExternalCategory            = 8;
kDETAspectKind                        = 9;
kDETAspectGender                      = 10;
kDETAspectWhatIs                      = 11;
kDETAspectAliasKind                   = 12;
kDETAspectAliasGender                 = 13;
kDETAspectAliasWhatIs                 = 14;
kDETAspectBalloons                    = 15;
kDETAspectNewMenuName                 = 16;
kDETAspectNewEntryName                = 17;
kDETAspectNewValue                    = 18;
kDETAspectSublistOpenOnNew            = 19;
kDETAspectLookup                      = 20;
kDETAspectDragInString                = 21;
kDETAspectDragInVerb                  = 22;
kDETAspectDragInSummary               = 23;
kDETAspectRecordDragIn                = 24;
kDETAspectRecordCatDragIn             = 25;
kDETAspectAttrDragIn                  = 26;
kDETAspectAttrDragOut                 = 27;
```

```
kDETAspectViewMenu                  = 28;
kDETAspectReverseSort               = 29;
kDETAspectInfoPageCustomWindow      = 30;

{Properties};
kDETNoProperty                      -1;
kDETFirstLocalProperty              0;
kDETLastLocalProperty               (kDETFirstLocalProperty + 249);
kDETFirstDevProperty                40;
kDETFirstConstantProperty           250;
kDETLastConstantProperty            (kDETFirstConstantProperty + 249);
kDETConstantProperty                kDETFirstConstantProperty;
kDETZeroProperty                    (kDETConstantProperty + 0);
kDETOneProperty                     (kDETConstantProperty + 1);
kDETFalseProperty                   (kDETConstantProperty + 0);
kDETTrueProperty                    (kDETConstantProperty + 1);

{Name and kind properties}
kDETPrName                          3050;
kDETPrKind                          3051;

kDETPastFirstLookup                 26550;
kDETInfoPageNumber                  27050;
kDETAspectTemplateNumber            26551;
kDETInfoPageTemplateNumber          26552;
kDETOpenSelectedItems               26553;{open selected sublist items}
kDETAddNewItem                      26554;{add new sublist item}
kDETRemoveSelectedItems             26555;{remove selected sublist items}

{Access masks}
kDETDNodeAccessMask                 25825;{the DNode access mask}
kDETRecordAccessMask                25826;{the record access mask}
kDETAttributeAccessMask             25827;{the attribute access mask}
kDETPrimaryMaskByBit                25828;{a set of 16 properties
                                          to access all bits of the
                                          primary mask}

kDETPrimarySeeMask                  kDETPrimaryMaskByBit;
kDETPrimaryAddMask                  (kDETPrimaryMaskByBit + 1);
kDETPrimaryDeleteMask               (kDETPrimaryMaskByBit + 2);
kDETPrimaryChangeMask               (kDETPrimaryMaskByBit + 3);
kDETPrimaryRenameMask               (kDETPrimaryMaskByBit + 4);
kDETPrimaryChangePrivsMask          (kDETPrimaryMaskByBit + 5);
kDETPrimaryTopMaskBit               (kDETPrimaryMaskByBit + 15);
```

```
{Property types}
kDETPrTypeNumber                    -1    {a number}
kDETPrTypeString                    -2    {a string}
kDETPrTypeBinary                    -3    {a binary block}

{Rez-compatible attribute-tag types}
typeRString                         'rstr';
typePackedDSSpec                    'dspc';
typeBinary                          'bnry';

{Constants used in view lists}
kDETNoFlags                   $0000;
kDETEnabled                   $0001;   {main view field enabled}

kDETHilightIfSelected         $0001;   {hilight when entry is selected}

kDETNumericOnly               $0008;   {allow digits only}
kDETMultiLine                 $0010;   {allow multiple lines in view}
kDETDynamicSize               $0200;   {don't draw box around text
                                          until user clicks in it,
                                          then auto-size it}
kDETAllowNoColons             $0400;   {don't allow colons}

kDETPopupDynamicSize          $0100;   {automatically resize pop-up}

kDETScaleToView               $0100;   {scale picture to view bounds}

kDETLargeIcon                       0;
kDETSmallIcon                       1;
kDETMiniIcon                        2;

kDETLeft                            0;
kDETCenter                          1;
kDETRight                           -1;
kDETForceLeft                       -2;

kDETUnused                          0;
kDETBoxTakesContentClicks           $0001;
kDETBoxIsRounded                    $0002;
kDETBoxIsGrayed                     $0004;
kDETBoxIsInvisible                  $0008;

kDETApplicationFont                  1;
kDETApplicationFontSize              9;
```

```
kDETAppFontLineHeight               12;

kDETSystemFont                       0;
kDETSystemFontSize                  12;
kDETSystemFontLineHeight            16;

kDETDefaultFont                      1;
kDETDefaultFontSize                  9;
kDETDefaultFontLineHeight           12;

kDETNormal                          0;
kDETBold                            1;
kDETItalic                          2;
kDETUnderline                       4;
kDETOutline                         8;
kDETShadow                          $10;
kDETCondense                        $20;
kDETExtend                          $40;

kDETIconStyle                       -3;   {normal text style for
                                            regular sublist entries,
                                            italic text style for aliases}

kDETChangeViewCommand               'view';  {change the view}

{Default information page layouts}

{Default record information page size}
kDETRecordInfoWindHeight            228;
kDETRecordInfoWindWidth             400;

{Default attribute information page size}
kDETAttributeInfoWindHeight         250;
kDETAttributeInfoWindWidth          230;

{Page identifying icon}
kDETSubpageIconTop                  8;
kDETSubpageIconLeft                 8;
kDETSubpageIconBottom               (kDETSubpageIconTop + 32);
kDETSubpageIconRight                (kDETSubpageIconLeft + 32);
*( #define kDETSubpageIconRect      (kDETSubpageIconTop,\
                                     kDETSubpageIconLeft,\
                                     kDETSubpageIconBottom,\
                                     kDETSubpageIconRight) *)
```

```
{The following rectangle can be used in a 'deti' with no sublist:}
(* #define kDETNoSublistRect        {0, 0, 0, 0} *)

{Reserved category names}
kDETCategoryAllItems             'aoce All Items';    {everything}
kDETCategoryAddressItems         'aoce Address Items';{all addresses}
kDETCategoryMisc                 'aoce Miscellaneous';{things that
                                         don't have their own category}

{Target selectors}
enum DETTargetSelector {
   kDETSelf = 0;                 {the "current" item}
   kDETSelfOtherAspect = 1;      {another aspect of the current item}
   kDETParent = 2;               {the parent of the current item}
   kDETSublistItem = 3;          {the ith item in the sublist}
   kDETSelectedSublistItem = 4;  {the ith selected item in the sublist}
   kDETDSSpec = 5;               {DSSpec}
   kDETAspectTemplate = 6;       {specific aspect template}
   kDETInfoPageTemplate = 7;     {specific info-page template}
   kDETHighSelector = $F000      {force type to be short}
};

{Return value for code resources}
CONST kDETDidNotHandle  = 1;

{Valid commandIDs for DETDropQueryBlock and DETDropMeQueryBlock (in
    addition to property numbers)}
CONST
   kDETDoNothing           = 'xxx0';
   kDETMove                = 'move';
   kDETDrag                = 'drag';
   kDETAlias               = 'alis';

{Application-defined routines}
CONST
   kDETcmdSimpleCall          = 0;
   kDETcmdInit                = 1;
   kDETcmdExit                = 2;
   kDETcmdAttributeCreation   = 3;
   kDETcmdDynamicForwarders   = 4;

   kDETcmdTargetedCall        = 1000;
   kDETcmdInstanceInit        = 1001;
```

```
   kDETcmdInstanceExit        = 1002;
   kDETcmdIdle                = 1003;
   kDETcmdViewListChanged     = 1004;
   kDETcmdValidateSave        = 1005;
   kDETcmdDropQuery           = 1006;
   kDETcmdDropMeQuery         = 1007;
   kDETcmdAttributeNew        = 1008;
   kDETcmdAttributeChange     = 1009;
   kDETcmdAttributeDelete     = 1010;
   kDETcmdItemNew             = 1011;
   kDETcmdOpenSelf            = 1012;
   kDETcmdDynamicResource     = 1013;
   kDETcmdShouldSync          = 1014;
   kDETcmdDoSync              = 1015;

   kDETcmdPropertyCall        = 2000
   kDETcmdPropertyCommand     = 2001;
   kDETcmdMaximumTextLength   = 2002;
   kDETcmdPropertyDirtied     = 2003 ;
   kDETcmdPatternIn           = 2004;
   kDETcmdPatternOut          = 2005;
   kDETcmdConvertToNumber     = 2006;
   kDETcmdConvertToRString    = 2007;
   kDETcmdConvertFromNumber   = 2008;
   kDETcmdConvertFromRString  = 2009;
   kDETcmdCustomViewDraw      = 2010;
   kDETcmdCustomViewMouseDown = 2011;
   kDETcmdKeyPress            = 2012;
   kDETcmdPaste               = 2013;
   kDETcmdCustomMenuSelected  = 2014;
   kDETcmdCustomMenuEnabled   = 2015;

   kDETcmdHighCall            = $F0000000    {force the type to be long}
};

{Callback functions}
CONST
   kDETcmdSimpleCallback           = 0;
   kDETcmdBeep                     = 1;
   kDETcmdBusy                     = 2;
   kDETcmdChangeCallFors           = 3;
   kDETcmdGetCommandSelectionCount = 4;
   kDETcmdGetCommandItemN          = 5;
   kDETcmdOpenDSSpec               = 6;
```

```
   kDETcmdAboutToTalk               = 7;
   kDETcmdUnloadTemplates           = 8;
   kDETcmdTemplateCounts            = 9;

   kDETcmdTargetedCallback          = 1000;
   kDETcmdGetDSSpec                 = 1001;
   kDETcmdSublistCount              = 1002;
   kDETcmdSelectedSublistCount      = 1003;
   kDETcmdRequestSync               = 1004;
   kDETcmdBreakAttribute            = 1005;
   kDETcmdGetTemplateFSSpec         = 1006;
   kDETcmdGetOpenEdit               = 1007;
   kDETcmdCloseEdit                 = 1008;

   kDETcmdPropertyCallback          = 2000;
   kDETcmdGetPropertyType           = 2001;
   kDETcmdGetPropertyNumber         = 2002;
   kDETcmdGetPropertyRString        = 2003;
   kDETcmdGetPropertyBinarySize     = 2004;
   kDETcmdGetPropertyBinary         = 2005;
   kDETcmdGetPropertyChanged        = 2006;
   kDETcmdGetPropertyEditable       = 2007;
   kDETcmdSetPropertyType           = 2008;
   kDETcmdSetPropertyNumber         = 2009;
   kDETcmdSetPropertyRString        = 2010;
   kDETcmdSetPropertyBinary         = 2011;
   kDETcmdSetPropertyChanged        = 2012;
   kDETcmdSetPropertyEditable       = 2013;
   kDETcmdDirtyProperty             = 2014;
   kDETcmdDoPropertyCommand         = 2015;
   kDETcmdAddMenu                   = 2016;
   kDETcmdRemoveMenu                = 2017;
   kDETcmdMenuItemRString           = 2018;
   kDETcmdSaveProperty              = 2019;
   kDETcmdGetCustomViewUserReference = 2020;
   kDETcmdGetCustomViewBounds       = 2021;
   kDETcmdGetResource               = 2022;

   kDETcmdHighCallback              = $F0000000;   {force type to be LongInt}

CONST
{Values of DETItemType}
kDETHFSType       = 0;          {HFS item type}
kDETDSType        = 1;          {Catalog Service item type}
```

```
kDETMailType      = 2;          {Mail (letter) item type}
kDETMoverType     = 3;          {sounds, fonts, etc., from inside a
                                   suitcase or system file}
kDETLastItemType  = $F0000000;  {force it to be a LongInt}
```

Call-For Mask

```
CONST
   kDETCallForOther        = 1;     {call for events not listed below}
   kDETCallForIdle         = 2;     {kDETcmdIdle}
   kDETCallForCommands     = 4;     {kDETcmdPropertyCommand, kDETcmdSelfOpen}
   kDETCallForViewChanges  = 8;     {kDETcmdViewListChanged,
                                     kDETcmdPropertyDirtied,
                                     kDETcmdMaximumTextLength}
   kDETCallForDrops        = $10;   {kDETcmdDropQuery, kDETcmdDropMeQuery}
   kDETCallForAttributes   = $20;   {kDETcmdAttributeCreation,
                                     kDETcmdAttributeNew,
                                     kDETcmdAttributeChange,
                                     kDETcmdAttributeDelete}
   kDETCallForValidation   = $40;   {kDETcmdValidateSave}
   kDETCallForKeyPresses   = $80;   {kDETcmdKeyPress, kDETcmdPaste}
   kDETCallForResources    = $100;  {kDETcmdDynamicResource}
   kDETCallForSyncing      = $200;  {kDETcmdShouldSync, kDETcmdDoSync}
   kDETCallForEscalation   = $8000; {all calls escalated to the next level}

   kDETCallForNothing      = 0;     {none of the above}
   kDETCallForEverything   = $FFFFFFFF;   {all of the above}
```

Data Types

```
TYPE
   DETTargetSelector = Integer;

   DETCallBackFunctions = LongInt;

   DETCallFunctions = LongInt;

   DETCall = ProcPtr;

   DETItemType = LongInt;

{FSSpec plus additional info}
   DETFSInfo =
   RECORD
```

```
      fileType: OSType;    {File type}
      fileCreator: OSType; {File creator}
      fdFlags: Integer;    {Finder flags}
      fsSpec: FSSpec;      {FSSpec}
   END;

DETFSInfoPtr = ^DETFSInfo;

LetterSpecPtr = ^LetterSpec;

LetterSpecHandle = ^LetterSpecPtr;
```

Target Specifier

```
TYPE
   DETTargetSelector = Integer

   DETTargetSpecification =
   RECORD
      selector: DETTargetSelector;  {target selector}
      aspectName: RStringPtr;       {aspect name}
      itemNumber: LongInt;          {sublist index number}
      dsSpec: PackedDSSpecPtr;      {DSSpec}
   END;
```

Forwarder List

```
TYPE
   DETForwarderListItem = RECORD
      next: ^DETForwarderListPtr;      {handle to next item, or nil}
      attributeValueTag: AttributeTag; {attribute value tag (0 for none)}
      rstrs: PackedPathName;           {forwarder list}
   END;

   DETForwarderListPtr = ^DETForwarderListItem;
   DETForwarderListHandle = ^DETForwarderListPtr;
```

Call Block Headers

```
TYPE
   DETCallBlockHeader =
   RECORD
      reqFunction: DETCallFunctions;{requested function}
      callBack: DETCallBack;        {pointer to callback routine}
```

```
   callBackPrivate: LongInt;     {private data for the callback routine}
   templatePrivate: LongInt;     {private data stored in template}
END;

DETCallBlockTargetedHeader =
RECORD
   reqFunction: DETCallFunctions;{requested function}
   callBack: DETCallBack;        {pointer to callback routine}
   callBackPrivate: LongInt;     {private data for the callback routine}
   templatePrivate: LongInt;     {private data stored in template}
   instancePrivate: LongInt;     {private data stored in aspect}
   target: DETTargetSpecification;{the target (originator) of the call}
   targetIsMainAspect: Boolean;  {TRUE if the target is the main aspect}
END;

DETCallBlockPropertyHeader =
RECORD
   reqFunction: DETCallFunctions;{requested function}
   callBack: DETCallBack;        {pointer to callback routine}
   callBackPrivate: LongInt;     {private data for the callback routine}
   templatePrivate: LongInt;     {private data stored in template}
   instancePrivate: LongInt;     {private data stored in aspect}
   target: DETTargetSpecification;{the target (originator) of the call}
   targetIsMainAspect: Boolean;  {TRUE if the target is the main aspect}
   property: Integer;            {the property number the call refers to}
END;

DETProtoCallBlock = DETCallBlockPropertyHeader;
```

Call Block Case Statement

```
TYPE
   DETCallBlock =
   RECORD
   CASE Integer OF
       1: (protoCall: DETProtoCallBlock);
       2: (init: DETInitBlock);
       3: (exit: DETExitBlock);
       4: (instanceInit: DETInstanceInitBlock);
       5: (instanceExit: DETInstanceExitBlock);
       6: (instanceIdle: DETInstanceIdleBlock);
       7: (propertyCommand: DETPropertyCommandBlock);
       8: (maximumTextLength: DETMaximumTextLengthBlock);
```

```
    9: (viewListChanged: DETViewListChangedBlock);
   10: (propertyDirtied: DETPropertyDirtiedBlock);
   11: (validateSave: DETValidateSaveBlock);
   12: (dropQuery: DETDropQueryBlock);
   13: (dropMeQuery: DETDropMeQueryBlock);
   14: (attributeCreationBlock: DETAttributeCreationBlock);
   15: (attributeNew: DETAttributeNewBlock);
   16: (attributeChange: DETAttributeChangeBlock);
   17: (attributeDelete: DETAttributeDeleteBlock);
   18: (itemNew: DETItemNewBlock);
   19: (patternIn: DETPatternInBlock);
   20: (patternOut: DETPatternOutBlock);
   21: (shouldSync: DETShouldSyncBlock);
   22: (doSync: DETDoSyncBlock);
   23: (openSelf: DETOpenSelfBlock);
   24: (convertToNumber: DETConvertToNumberBlock);
   25: (convertToRString: DETConvertToRStringBlock);
   26: (convertFromNumber: DETConvertFromNumberBlock);
   27: (convertFromRString: DETConvertFromRStringBlock);
   28: (customViewDraw: DETCustomViewDrawBlock);
   29: (customViewMouseDown: DETCustomViewMouseDownBlock);
   30: (keyPress: DETKeyPressBlock);
   31: (paste: DETPasteBlock);
   32: (customMenuSelected: DETCustomMenuSelectedBlock);
   33: (customMenuEnabled: DETCustomMenuEnabledBlock);
   34: (dynamicForwarders: DETDynamicForwardersBlock);
   35: (dynamicResource: DETDynamicResourceBlock);
END;

DETCallBlockPtr = ^DETCallBlock;
```

Callback Block Headers

```
TYPE
   DETCallBackBlockHeader =
   RECORD
      reqFunction: DETCallBackFunctions;  {requested function}
   END;

   DETCallBackBlockTargetedHeader =
   RECORD
      reqFunction: DETCallBackFunctions;  {requested function}
      target: DETTargetSpecification;     {the target for the request}
```

```
   END;

   DETCallBackBlockPropertyHeader =
   RECORD
      reqFunction: DETCallBackFunctions;  {requested function}
      target: DETTargetSpecification;     {the target for the request}
      property: Integer;                  {the property to apply the
                                            request to}
   END;

   DETProtoCallBackBlock = DETCallBackBlockPropertyHeader;
```

Callback Block Case Statement

```
TYPE
   DETCallBackBlock =
   RECORD
   CASE Integer OF
      1: (protoCallBack: DETProtoCallBackBlock);
      2: (beep: DETBeepBlock);
      3: (busy: DETBusyBlock);
      4: (changeCallFors: DETChangeCallForsBlock);
      5: (getCommandSelectionCount: DETGetCommandSelectionCountBlock);
      6: (getCommandItemN: DETGetCommandItemNBlock);
      7: (getDSSpec: DETGetDSSpecBlock);
      8: (getTemplateFSSpec: DETGetTemplateFSSpecBlock);
      9: (getOpenEdit: DETGetOpenEditBlock);
      10: (closeEdit: DETCloseEditBlock);
      11: (getPropertyType: DETGetPropertyTypeBlock);
      12: (getPropertyNumber: DETGetPropertyNumberBlock);
      13: (getPropertyRString: DETGetPropertyRStringBlock);
      14: (getPropertyBinarySize: DETGetPropertyBinarySizeBlock);
      15: (getPropertyBinary: DETGetPropertyBinaryBlock);
      16: (getPropertyChanged: DETGetPropertyChangedBlock);
      17: (getPropertyEditable: DETGetPropertyEditableBlock);
      18: (setPropertyType: DETSetPropertyTypeBlock);
      19: (setPropertyNumber: DETSetPropertyNumberBlock);
      20: (setPropertyRString: DETSetPropertyRStringBlock);
      21: (setPropertyBinary: DETSetPropertyBinaryBlock);
      22: (setPropertyChanged: DETSetPropertyChangedBlock);
      23: (setPropertyEditable: DETSetPropertyEditableBlock);
      24: (dirtyProperty: DETDirtyPropertyBlock);
      25: (doPropertyCommand: DETDoPropertyCommandBlock);
```

```
      26: (sublistCount: DETSublistCountBlock);
      27: (selectedSublistCount: DETSelectedSublistCountBlock);
      28: (requestSync: DETRequestSyncBlock);
      29: (addMenu: DETAddMenuBlock);
      30: (removeMenu: DETRemoveMenuBlock);
      31: (menuItemRString: DETMenuItemRStringBlock);
      32: (openDSSpec: DETOpenDSSpecBlock);
      33: (aboutToTalk: DETAboutToTalkBlock);
      34: (breakAttribute: DETBreakAttributeBlock);
      35: (saveProperty: DETSavePropertyBlock);
      36: (getCustomViewUserReference: DETGetCustomViewUserReferenceBlock);
      37: (getCustomViewBounds: DETGetCustomViewBoundsBlock);
      38: (getResource: DETGetResourceBlock);
      39: (templateCounts: DETTemplateCounts);
      40: (unloadTemplates: DETUnloadTemplatesBlock);
   END;

   DETCallBackBlockPtr = ^DETCallBackBlock;
```

Functions You Can Provide as Part of Your Code Resource

Initializing and Removing Templates

```
TYPE
DETInitBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      newCallFors: LongInt;
   END;

DETExitBlock = DETCallBlockHeader;

DETInstanceInitBlock = DETCallBlockTargetedHeader;

DETItemNewBlock = DETCallBlockTargetedHeader;

DETInstanceExitBlock = DETCallBlockTargetedHeader;
```

Dynamic Creation of Resources

```
TYPE
DETDynamicForwardersBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      forwarders: DETForwarderListHandle;
   END;

DETDynamicResourceBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      resourceType: ResType;
      propertyNumber: Integer;
      resourceID: Integer;
      theResource: Handle;
   END;
```

Processing Idle-Time Tasks

```
TYPE
DETInstanceIdleBlock = DETCallBlockTargetedHeader;
```

Property and Information Page Routines

```
TYPE
DETOpenSelfBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
```

```
      targetIsMainAspect: Boolean;
      modifiers: Integer;
   END;

DETPropertyCommandBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      parameter: LongInt;
   END;

DETKeyPressBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      theEvent: ^EventRecord;
   END;

DETPasteBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      modifiers: Integer;
   END;
```

```
DETMaximumTextLengthBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      maxSize: LongInt;
   END;

DETViewListChangedBlock = DETCallBlockTargetedHeader;

DETPropertyDirtiedBlock = DETCallBlockPropertyHeader;

DETValidateSaveBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      errorString: RStringHandle;
   END;
```

Supporting Drops

```
TYPE
DETDropMeQueryBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      modifiers: Integer;
      commandID: LongInt;
```

```
      destinationType: AttributeType;
      copyToHFS: Boolean;
   END;

DETDropQueryBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      modifiers: Integer;
      commandID: LongInt;
      destinationType: AttributeType;
      copyToHFS: Boolean;
   END;
```

Attribute-Related Commands

```
TYPE
DETAttributeCreationBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      parent: PackedDSSpecPtr;
      refNum: Integer;
      identity: AuthIdentity;
      attrType: AttributeType;
      attrTag: AttributeTag;
      value: Handle;
   END;

DETAttributeNewBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
```

```
      targetIsMainAspect: Boolean;
      parent: PackedDSSpecPtr;
      refNum: Integer;
      identity: AuthIdentity;
      attrType: AttributeType;
      attrTag: AttributeTag;
      value: Handle;
   END;

DETAttributeChangeBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      parent: PackedDSSpecPtr;
      refNum: Integer;
      identity: AuthIdentity;
      attrType: AttributeType;
      attrTag: AttributeTag;
      attrCID: AttributeCreationID;
      value: Handle;
   END;

DETAttributeDeleteBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      dsSpec: PackedDSSpecPtr;
      refNum: Integer;
      identity: AuthIdentity;
   END;
```

Processing Custom Lookup-Table Pattern Elements

```
TYPE
DETPatternInBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      elementType: LongInt;
      extra: LongInt;
      attribute: AttributePtr;
      dataOffset: LongInt;
      bitOffset: Integer;
   END;

DETPatternOutBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      elementType: LongInt;
      extra: LongInt;
      attribute: AttributePtr;
      data: Handle;
      dataOffset: LongInt;
      bitOffset: Integer;
   END;
```

Synchronizing Property Values

```
TYPE
DETShouldSyncBlock =
   RECORD
      reqFunction: DETCallFunctions;
```

```
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      shouldSync: Boolean;
   END;

DETDoSyncBlock = DETCallBlockTargetedHeader;
```

Custom Property-Type Conversions

```
TYPE
DETConvertToNumberBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      theValue: LongInt;
   END;

DETConvertToRStringBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      property: Integer;
      theValue: RStringHandle;
   END;

DETConvertFromNumberBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
```

```
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      theValue: LongInt;
   END;

DETConvertFromRStringBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      theValue: RStringPtr;
   END;
```

Custom Views and Custom Menus

```
TYPE
DETCustomViewDrawBlock = DETCallBlockPropertyHeader;

DETCustomViewMouseDownBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      property: Integer;
      theEvent: ^EventRecord;
   END;

DETCustomMenuEnabledBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
```

```
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      menuTableParameter: Integer;
      enable: Boolean;
   END;

DETCustomMenuSelectedBlock =
   RECORD
      reqFunction: DETCallFunctions;
      callBack: DETCallBack;
      callBackPrivate: LongInt;
      templatePrivate: LongInt;
      instancePrivate: LongInt;
      target: DETTargetSpecification;
      targetIsMainAspect: Boolean;
      menuTableParameter: Integer;
   END;
```

CE-Provided Functions That Your Code Resource Can Call

Calling CE-Provided Functions

```
{There is no Pascal equivalent to the C macro for calling callback routines.
```

Testing Your Code Resource

```
DETBeepBlock = DETCallBackBlockHeader;

DETBusyBlock = DETCallBackBlockHeader;

DETChangeCallForsBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      newCallFors: LongInt;
   END;

DETGetCommandSelectionCountBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      count: LongInt;
   END;
```

```
DETGetCommandItemNBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      itemNumber: LongInt;
      itemType: DETItemType;
      CASE Integer OF
         1: (fsInfo: ^DETFSInfoPtr);
         2: (ds: RECORD
            dsSpec: ^PackedDSSpecPtr;{
            refNum: Integer;
            identity: AuthIdentity;
            END);
         3: (dsSpec: ^PackedDSSpecPtr);
         4: (ltrSpec: LetterSpecHandle);
   END;

DETGetDSSpecBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      dsSpec: ^PackedDSSpecPtr;
      refNum: Integer;
      identity: AuthIdentity;
      isAlias: Boolean;
      isRecordRef: Boolean;
   END;

DETGetTemplateFSSpecBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      fsSpec: FSSpec;
      baseID: Integer;
      aspectTemplateNumber: LongInt;
   END;

DETGetOpenEditBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      viewProperty: Integer;
   END;
```

```
DETCloseEditBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
   END;

DETGetPropertyTypeBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      propertyType: Integer;
   END;

DETGetPropertyNumberBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      propertyValue: LongInt;
   END;

DETGetPropertyRStringBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      propertyValue: RStringHandle;
   END;

DETGetPropertyBinarySizeBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      propertyBinarySize: LongInt;
   END;

DETGetPropertyBinaryBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
```

```
      property: Integer;
      propertyValue: Handle;
   END;

DETGetPropertyChangedBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      propertyChanged: Boolean;
   END;

DETGetPropertyEditableBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      propertyEditable: Boolean;
   END;

DETSetPropertyTypeBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      newType: Integer;
   END;

DETSetPropertyNumberBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      newValue: LongInt;
   END;

DETSetPropertyRStringBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      newValue: RStringPtr;
   END;
```

```
DETSetPropertyBinaryBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      newValue: Ptr;
      newValueSize: LongInt;
   END;

DETSetPropertyChangedBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      propertyChanged: Boolean;
   END;

DETSetPropertyEditableBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      propertyEditable: Boolean;
   END;

DETDirtyPropertyBlock = DETCallBackBlockPropertyHeader;

DETDoPropertyCommandBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      parameter: LongInt;
   END;

DETSublistCountBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      count: LongInt;
   END;

DETSelectedSublistCountBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
```

```
      target: DETTargetSpecification;
      count: LongInt;
   END;

DETRequestSyncBlock =  DETCallBackBlockTargetedHeader;

DETAddMenuBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      name: ^RString;
      parameter: LongInt;
      addAfter: LongInt;
   END;

DETRemoveMenuBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      itemToRemove: LongInt;
   END;

DETMenuItemRStringBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      itemParameter: LongInt;
      rString: RStringHandle;
   END;

DETOpenDSSpecBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      dsSpec: PackedDSSpecPtr;
   END;

DETAboutToTalkBlock = DETCallBackBlockHeader;

DETBreakAttributeBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
```

```
      breakAttribute: AttributePtr;
      isChangeable: Boolean;
   END;

DETSavePropertyBlock = DETCallBackBlockPropertyHeader;

DETGetCustomViewUserReferenceBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      userReference: Integer;
   END;

DETGetCustomViewBoundsBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      bounds: Rect;
   END;

DETGetResourceBlock =
   RECORD
      reqFunction: DETCallBackFunctions;
      target: DETTargetSpecification;
      property: Integer;
      resourceType: ResType;
      theResource: Handle;
   END;

DETTemplateCounts =
   RECORD
      reqFunction: DETCallBackFunctions;
      aspectTemplateCount: LongInt;
      infoPageTemplateCount: LongInt;
   END;

DETUnloadTemplatesBlock = DETCallBackBlockHeader;
```

Result Codes

Result codes in the range of –15000 to –15039 are reserved for AOCE templates.

noErr	0	No error
kDETInvalidTargetAspectName	–15000	Could not find aspect named in target selector
kDETInvalidTargetItemNumber	–15001	Item number in target selector out of range
kDETInvalidTargetFromNonAspect	–15002	Targeted item doesn't have an aspect
kDETInvalidTargetDSSpec	–15003	DSSpec in target selector could not be resolved
kDETUnknownTargetSelector	–15004	Selector type in target selector invalid
kDETInvalidTarget	–15005	Target selector invalid
kDETTargetNotAnAspect	–15006	Specified target object not an aspect
kDETInvalidCommandItemNumber	–15007	Command item number out of range
kDETUnableToGetCommandItemSpec	–15008	Unable to retrieve information about item (possibly out of memory)
kDETRequestedTypeUnavailable	–15009	Item could not be represented in the specified format
kDETInvalidDSSpec	–15010	Could not resolve DSSpec
kDETUnableToAccessProperty	–15011	Property could not be found
kDETInfoPageNotOpen	–15012	Information page not open
kDETNoSuchView	–15013	No view found with specified property number
kDETCouldNotAddMenuItem	–15014	Could not add item to menu
kDETCouldNotRemoveMenuItem	–15015	Could not remove item from dynamic menu
kDETCouldNotFindMenuItem	–15016	Could not find menu item
kDETCouldNotFindCustomView	–15017	Could not find custom view
kDETInvalidReqFunction	–15018	Invalid callback routine selector
kDETInvalidCallBack	–15019	Invalid callback (for reasons other than those above)
kDETPropertyBusy	–15020	Specified property is being edited

CHAPTER 6

Digital Signature Manager

Contents

This chapter describes the Digital Signature Manager, one of the AOCE security services. The Digital Signature Manager is a set of routines that allows you to add electronic signature capabilities to your application.

Read this chapter if you plan to let your users sign documents or other data electronically, so that recipients can be confident of the authenticity of the signature and the integrity of the signed data.

Note that other AOCE components provide limited use of digital signatures; the AOCE Standard Mail Package allows users to add digital signatures to electronic mail and to check the signatures of received mail. The Interprogram Messaging (IPM) Manager provides an application program interface to add digital signatures to IPM messages and to verify such signatures. A user who has the AOCE software installed can use the Finder to add a digital signature to any file. If, however, you want to allow a user to sign a file or verify a signature from within your application, you must use the routines described in this chapter.

This chapter first introduces the concept of digital signatures, including a brief introduction to public-key cryptography. It goes on to explain public-key certificates—necessary documents for creating digital signatures. It then explains how to use the Digital Signature Manager to create a signature for a file or portion of a file, and to verify a signature. It also explains how to get information from a digital signature.

About Digital Signatures

A **digital signature** is an encrypted number that is associated with a particular set of data. It has two purposes. First, it uniquely identifies the individual or entity that signed or authorized the content of the data. Second, it ensures the integrity of the data; the signature contains coded information that can be used to detect any changes made to the data after the creation of the signature.

A digital signature can be applied to an entire set of data or to any portion of it; anything that can be represented as a stream of bytes can be given a digital signature. You can use the functions in this chapter to sign a file, one or more fields in a form, data in memory, or even another digital signature. In terms of security and integrity, an item with a verifiable digital signature is comparable to a paper document that is signed and notarized. In most ways, digital signatures can provide better security than signed paper forms, because a digital signature cannot be forged and because a digitally signed document cannot be altered without the alteration being detected.

The digital signature capability is useful in networked organizations. Users on the network can fill out forms and route them electronically for signature, thus saving time and effort and enhancing security. Even users of computers that are not on a network can sign electronic forms before mailing them or physically delivering them. Digital signatures can also be used with data that is not transmitted at all; a user can assign a digital signature to important data left on a computer or server to ensure that the data is not tampered with. This capability could be used to detect viruses, for example.

Cryptography and Digital Signatures

The digital signature technology used by the AOCE toolbox involves the use of two **keys**, large unique numbers that are computationally applied to data to encrypt or decrypt it.

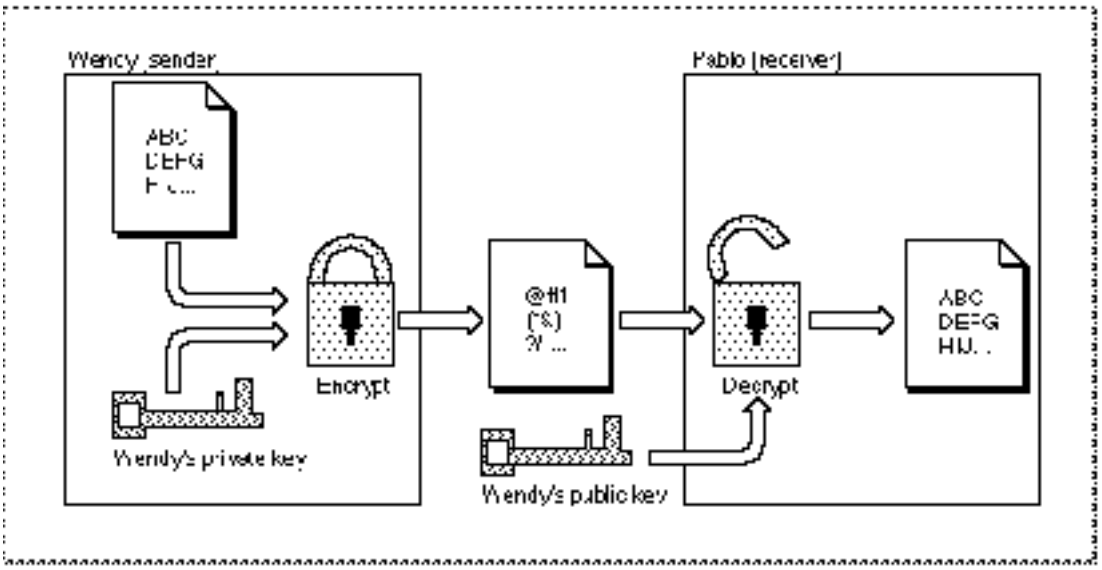
The Digital Signature Manager does not encrypt documents; encryption and decryption are applied to the digital signature only. When you send an electronically signed document, the contents of the document are as public as the channel over which you transmit.

Common cryptographic techniques typically involve a single key, one that both decrypts and encrypts information. Any holder of the key used in the encryption can use it to decrypt the information. Those wishing to exchange information must keep the key a secret among themselves. This type of cryptography is called **secret-key cryptography**.

The AOCE services use another cryptographic technique, called **public-key cryptography**. In this technique, key holders use a pair of keys to encrypt and decrypt information. Each key pair consists of a **private key** and a **public key**. A key holder must keep its private key secret and not share that key with anyone else. At the same time, it may freely publish and exchange its public key without compromising security.

Both the private and public keys can be used to encrypt information and decrypt information. Information encrypted with a private key can be decrypted only with its paired public key. Similarly, information encrypted with a public key can be decrypted only with its paired private key. Figure 6-1 illustrates the concept of public-key encryption.

Figure 6-1 Principles of public-key encryption



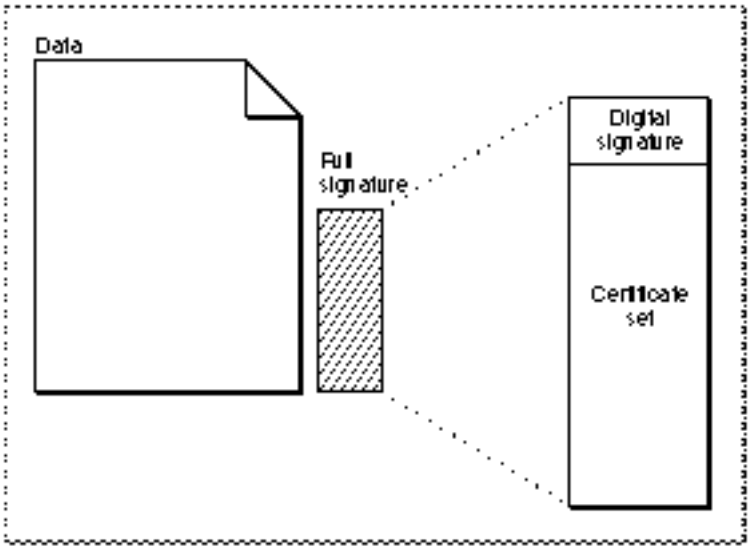
The sender, Wendy, uses her own private key to encrypt an item. The receiver, Pablo, can decrypt the item and read it because he has access to Wendy's public key. Wendy's public key is widely available, so the contents of the item are not hidden to anyone with the key. But because only Wendy's public key can decrypt the item, anyone who successfully decrypts the item knows that it must have come from Wendy.

Components of a Full Signature

As implemented by the Digital Signature Manager, an electronically signed item consists of (1) the item itself, any collection of data; and (2) a full digital signature**,** a stream of bytes that can be used to verify the integrity of the item's data and that uniquely identifies the signer.

A full signature has two components: the digital signature itself and the certificate set of the signer. Figure 6-2 illustrates the components of a full signature. Certificate sets provide the signer's public key, verification of the authenticity of that key, and the identity of the signer. They are described in "The Certificate Set" on page 6-6.

Figure 6-2 The components of a full signature



The Digital Signature

A digital signature is an encrypted digest. A **digest** is a 16-byte number, calculated by the Digital Signature Manager from a set of data, that reflects the content of that data. The digest is like a sophisticated checksum but far more reliable in verifying the integrity of data. It is very nearly impossible for any two sets of data that differ in any way to yield the same digest. The digest, therefore, ensures the integrity of the data; if someone changes even a single bit of a signed item, a recalculation of that item's digest will yield a different number from the digest contained in the signature.

Once the digest is created, it is encrypted. The **encrypted digest (**or **signed digest)** is the digital signature. The Digital Signature Manager encrypts the digest by applying the signer's private key to it. The encrypted digest is called a signed digest because it could have been created only by the signer (the holder of that private key).

Verifying a digital signature requires decrypting the encrypted digest and comparing it to a new digest of the same data. To decrypt the digest, the recipient of the signed data applies the signer's public key to it. Because an item encrypted with an individual's private key can be decrypted only with that same individual's public key, the very act of correctly decrypting a signature proves the identity of the signer.

Verifying a signature also requires making sure that the data has not changed since it was signed. The Digital Signature Manager creates a digest of the data in its present state and compares it with the decrypted digest from the signature. If they match, the signed data is unchanged.

Finally, verifying a signature requires establishing the authenticity of the public key used for the decryption. To allow for that, the Digital Signature Manager affixes a certificate set (described next) to every digital signature it creates.

The Certificate Set

The second part of a full signature—the certificate set—has three purposes: it provides the signer's public key to allow decryption of the signature, it allows verification of the authenticity of that public key, and it provides the identity of the signer.

Suppose, for example, that Mary (an impostor) claims to be Joe. She signs a document with her own private key and sends it off as a document from Joe. If Mary also sends along her own public key as Joe's public key, then the recipient of the document might use Mary's public key, thinking it was Joe's, to decrypt the signature. The decryption would be successful—because Mary's private key had performed the encryption—and the recipient would mistakenly think the message had been signed by Joe.

As a safeguard against deceptions of this kind, each public key in use is registered with a mutually trusted official issuing organization (such as a corporation or government bureau). That agency publishes a **public-key certificate**, which includes not only the public key itself but the name of the owner of the key and the name of the organization that issued the certificate (as well as other information; see "About Public-Key Certificates" beginning on page 6-8). See the AOCE user documentation for information on how a user obtains a public-key certificate.

As a guarantee of authenticity, each public-key certificate is itself digitally signed by the issuer of the certificate; it then becomes a **signed certificate.** No change to the name or the public key in a signed certificate can go undetected.

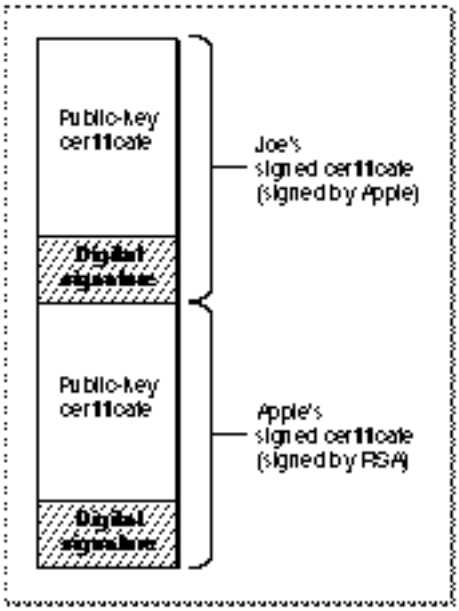
The signature on a certificate must itself be verified before the certificate can be considered authentic. For that reason each issuer also has a public-key certificate, signed by *its* issuer. Verifying the signature on a certificate thus leads to another certificate whose signature must be verified, and so on.

For each digital signature this chain of certificates, or **certificate set**, leads from the signer through all intermediate issuers and up to the prime issuing organization. Verifying a

digital signature requires verifying the signatures on all the certificates in the certificate set associated with that signer. This certification process ensures that every public key in every certificate is authentic, as long as one public key—that of the prime issuer—is trusted.

For example, assume Joe has a certificate issued by Apple Computer, Inc. Joe's certificate includes his name and public key, and it is digitally signed by Apple. Apple's certificate includes Apple's name and public key, and like Joe's it is digitally signed. If, in this example, RSA Data Security, Inc., had issued Apple's certificate, then the certificate set in Joe's signed certificate would consist of two certificates, as shown in Figure 6-3.

Figure 6-3 A certificate set consisting of two signed certificates



RSA has no certificate because there is no authorizing agency to issue it. RSA in this case is the prime issuer, so its public key cannot be verified. It must be trusted for reasons other than verifiability, such as wide public availability. The Digital Signature Manager has access to RSA's public key, so the key is available on every user's Macintosh computer.

In summary, when a recipient verifies the signature, the Digital Signature Manager (1) decrypts the digital signature with the public key provided in the certificate and compares the resulting digest with one it recalculates from the data; and (2) verifies all the digital signatures on the certificate set. If all the verifications are successful, the signer's public key is considered authentic.

Note

Because the number of attributes in a certificate is not limited, a full digital signature can be fairly large—as much as several kilobytes. u

Creating and Verifying Signatures

With the Digital Signature Manager you can let users sign documents, and you can verify the signatures on documents received by users. The Digital Signature Manager also provides routines with which you can get information from the certificate set.

When the user wants to sign a document, you call routines that prompt the user for private-key and certificate information, create a digest of the document, and append the user's certificate set to make the full signature. You then attach or otherwise associate the full signature with the document—in a way appropriate to your application—and it is ready to be sent to its recipient by any normal means.

When the user wants to validate a signed document, you locate the full signature by methods appropriate to your application, and you then call routines that verify the signature by decrypting the encrypted digest, creating a new digest from the document and comparing it with the decrypted one, and verifying the authenticity of the public keys in the certificate set. To process a digital signature created in another application, you must know how the other application created the signature.

You may also wish to record or provide the user with additional information, such as who signed the document and when they signed it. To get that information you can make calls that return information about a specified certificate within the full signature.

Note

Users can sign any file by dragging the file onto their signer file. They can verify the signature in a file by clicking the button in the Get Info window for the file. See "Dealing With Standard Signatures in Files" on page 6-22 for information on how to deal with this possibility. u

About Public-Key Certificates

Public-key certificates are an integral part of the digital signature concept. Only with an authentic public key can a signature be verified with confidence; the set of signed public-key certificates that accompany every signature is used to ensure that authenticity.

A public-key certificate is an electronic document that verifies the identity of a signer. A public-key certificate contains the following information:

- identifying information for the certificate owner—the entity, person, or organization that is authorized to use the certificate
- the public key of the owner of the certificate
- a time period (range of dates) during which the certificate is valid
- identifying information for the organization that issued the certificate
- a serial number (assigned by the issuer)

A public-key certificate is not valid unless it is digitally signed by the issuing organization of that certificate. It then becomes a signed certificate. The signature assures the authenticity of the certificate owner's name and public key.

A certificate can be owned by a person not affiliated with any organization, a person who is a member of an organization, an organizational role (such as vice president or administrator), or an issuer (a certified authority). A **distinguished name** is a set of attributes that fully specify the owner or issuer of a certificate. For example, the distinguished name of a private certificate owner consists of a common name (typically the proper name by which the person is known), a country, a state or province, a locality (such as a city), a zip code, and sometimes a street address.

The 1988 CCITT Recommendation X.520 sets guidelines for the definition and attributes of a distinguished name. The Digital Signature Manager supports a subset of the recommendation. Table 6-1 summarizes the attribute conventions supported by the Digital Signature Manager.

As the table shows, for example, every certification authority (issuing organization) must have either a country name or an organization name—and may have both—but cannot have a common name. Conversely, an individual residential certificate owner must have a common name but cannot have an organizational name or title.

IMPORTANT

When you display a distinguished name, be sure to show the entire set of attributes for that distinguished name. If you show only a portion of the distinguished name, the user might incorrectly identify the owner of the certificate. There may be two identical names, for example, two certificate owners named John Smith. s

Table 6-1 Conventions governing attributes of a distinguished name

Mandatory attributes	Optional attributes	Prohibited attributes
Attributes of a certification authority		
Country *or* organization	Country	Title
	Organization	Common name
	State or province	
	Locality	
	Organizational unit	
	Street address	
	Zip code	

continued

Table 6-1 Conventions governing attributes of a distinguished name (continued)

Mandatory attributes	Optional attributes	Prohibited attributes
Attributes of a residential person		
Country	Street address	Organization
State or province		Organizational unit
Locality		Title
Common name		
Zip code		
Attributes of an organizational person		
Organization	Country	
Common name	State or province	
	Locality	
	Street address	
	Organizational unit	
	Title	
	Zip code	
Attributes of an organizational role		
Organization	Country	Common name
Title	State or province	
	Locality	
	Organizational unit	
	Street address	
	Zip code	

A distinguished name can have one or more attributes of each mandatory or optional type, and the attributes can be arranged in a hierarchy to help indicate their relationships. You can use this hierarchy when you display the distinguished name for a user. Figure 6-4 illustrates a hierarchically arranged distinguished name of an organizational person.

Figure 6-4 Hierarchically arranged distinguished name

Country: USA
Organization: Apple Computer, Inc.
  Organizational unit: Research and Development
    Organizational unit: Collaborative Software
    Common name: Pablo Calamera
    Common name: ID number 123456
    Title: Software Engineer

Using the Digital Signature Manager

In using the Digital Signature Manager, the main tasks your application needs to perform are allowing the user to sign a document and verifying the signature on a document that the user receives.

Other Digital Signature Manager features give you added convenience in extracting certificate information—such as the name of the signer—from full signatures.

Note
Because the Digital Signature Manager is loaded into memory when it is used, in general it is a good idea to keep your calls to the Digital Signature Manager close together so that the memory is used only when it is needed. For example, if you call the `SIGNewContext` function when your application starts up, call several Digital Signature Manager routines sometime later, and call the `SIGDisposeContext` function when your application shuts down, the manager code segment remains in memory the whole time. u

Determining the Version Number of the Digital Signature Manager

To determine what version of the Digital Signature Manager is available, call the `Gestalt` function, using the selector value `gestaltDigitalSignatureVersion`. Upon completion of the call, the `response` parameter contains the version number in its low-order word. For example, a value of 0x0101 indicates version 1.0.1.

Using a Context

The Digital Signature Manager uses a private data structure called a **context** to hold information and the results of calculations while it is processing data. Before you call Digital Signature Manager routines to perform a specific task, you must call the `SIGNewContext` function (see page 6-28) to create a context. This function returns a pointer of type `SIGContextPtr`. You must provide this pointer to each subsequent routine that you call to perform that task. When you first create a context, it can be used for any task; however, once you pass a context to another routine (`SIGSignPrepare`, `SIGVerifyPrepare`, or `SIGDigestPrepare`), it can be used only for that specific task.

For example, to create a signature you first call the `SIGNewContext` function to create a context, then pass that context to the `SIGSignPrepare`, `SIGProcessData`, and `SIGSign` functions (see the following section for details). When you are finished creating the signature, you call the `SIGDisposeContext` function to dispose of the context. Once you have passed the context pointer to the `SIGSignPrepare` function, you cannot use that context to verify a signature or create a digest; you must create a new context for each such operation.

Table 6-2 summarizes the Digital Signature Manager tasks and the functions required to perform each task. The "Optional functions" column lists functions that you can call with the same context you used for the preceding function in the "Required functions" column.

Table 6-2 Digital Signature Manager tasks and functions

Task	Required functions	Optional functions
Creating a signature	`SIGNewContext`	
	`SIGSignPrepare`	`SIGGetSignerInfo`
		`SIGGetCertInfo`
		`SIGGetCertNameAttributes`
		`SIGGetCertIssuerNameAttributes`
	`SIGProcessData`	
	`SIGSign`	`SIGGetSignerInfo`
		`SIGGetCertInfo`
		`SIGGetCertNameAttributes`
		`SIGGetCertIssuerNameAttributes`
	`SIGDisposeContext`	

Table 6-2 Digital Signature Manager tasks and functions (continued)

Task	Required functions	Optional functions
Signing a file	SIGNewContext	
	SIGSignPrepare	SIGGetSignerInfo
		SIGGetCertInfo
		SIGGetCertNameAttributes
		SIGGetCertIssuerNameAttributes
	SIGSignFile	SIGGetSignerInfo
		SIGGetCertInfo
		SIGGetCertNameAttributes
		SIGGetCertIssuerNameAttributes
	SIGDisposeContext	
Checking for a standard signature	SIGFileIsSigned	
Verifying a file	SIGNewContext	
	SIGVerifyPrepare	
	SIGProcessData	
	SIGVerify	SIGShowSigner
		SIGGetSignerInfo
		SIGGetCertInfo
		SIGGetCertNameAttributes
		SIGGetCertIssuerNameAttributes
	SIGDisposeContext	
Verifying a signature	SIGNewContext	
	SIGVerifyFile	SIGShowSigner
		SIGGetSignerInfo
		SIGGetCertInfo
		SIGGetCertNameAttributes
		SIGGetCertIssuerNameAttributes
	SIGDisposeContext	

continued

Table 6-2 Digital Signature Manager tasks and functions (continued)

Task	Required functions	Optional functions
Creating a digest	`SIGNewContext`	
	`SIGDigestPrepare`	
	`SIGProcessData`	
	`SIGDigest`	
	`SIGDisposeContext`	

Creating a Full Signature

When the user wants to sign a document or a portion of a document, you are responsible for knowing the location and extent of the data to be signed and for attaching or associating the full signature with that data once the signature is created. The Digital Signature Manager expects you to provide a pointer to the data, a pointer to a memory block where it is to place the full signature, and a context pointer.

To create a signature, follow these steps:

1. First, call the `SIGNewContext` function to allocate and initialize a context. The function returns a context pointer. If the Digital Signature Manager is not already in memory, the Operating System loads it into memory.
2. Call the `SIGSignPrepare` function, passing it the context pointer. It opens the signer file you specify; if you do not specify one, it opens the default signer file, which is the last signer file used. If there is no default signer file, it prompts the user for a signer-file location. It also prompts the user for the password needed to decrypt the signer's private key. It returns to you the size that the full signature will be.
3. Call the `SIGProcessData` function as many times as necessary to process all of the data to be signed. Either move a pointer through the data each time you call the function, or create a buffer and put blocks of data into it. The `SIGProcessData` function creates a digest of the data to be signed.
4. Create a properly sized memory block to hold the signature, and call the `SIGSign` function. It encrypts the digest, assembles the full signature, and puts it in the memory block you allocated. The `SIGSign` function periodically calls a callback routine that you may provide, so that you can notify the user of the progress of the signing operation or perform other background tasks.
5. If you are finished creating signatures for the current signer, go on to the next step. If you are creating additional signatures on different data sets for the same signer, repeat steps 3 and 4 for each signature in turn. The user will not be prompted for a password or signer-file location as each additional signature is created.
6. When you are finished creating signatures for the current user, call the `SIGDisposeContext` function to release the memory used by the signing routines and to release the Digital Signature Manager from memory. The next time you call `SIGSignPrepare`, the user is prompted once more for a password.

Listing 6-1 shows an example of a function that creates a full signature for a piece of data. This function requires an application-defined function named `DoGetDataToProcess`, which cycles through all the data that is to be signed. At the end of the `SignData` function is a call to another application-defined function named `DoSaveSignature`, which controls how and where to save the signature.

Listing 6-1 A sample signature-creation routine

```
OSErr SignData()
{
   OSErr                error;
   Boolean              moreToSign;
   Size                 signatureSize;
   Size                 dataSize;
   SIGSignaturePtr      signature = NULL;
   SIGContextPtr        context = NULL;
   Ptr                  dataBuffer = NULL;

   do {
      /* Allocate a new context and prepare it for signing. */

      if ((error = SIGNewContext(&context)) != noErr)
         break;

      if ((error = SIGSignPrepare(context, (FSSpecPtr)NULL, "\p",
            &signatureSize)) != noErr)
         break;

      /* Retrieve the data to be signed, in your application-specific way
         and pass it to the toolbox to generate the digest for our
         signature. */

      /* NOTE: DoGetDataToProcess can be the same function for signing and
         verifying. */

      do {
         if (error = DoGetDataToProcess(&dataBuffer, &dataSize, &moreToSign))
            break;

         if (error = SIGProcessData(context, dataBuffer, dataSize))
            break;

      } while (moreToSign);

      if (error != noErr)/* if encountered error above, go all the way out */
         break;
```

```
      /* Allocate a buffer of the size returned from SIGSignPrepare to hold
         the signature and create the signature by passing the buffer to
         SIGSign. */

      signature = (SIGSignaturePtr)NewPtr(signatureSize);

      if (error = MemError())
         break;

      if (error = SIGSign(context, signature, (SIGStatusProcPtr)NULL))
         break;

      /* Save the signature in your application-specific way. */

      error = DoSaveSignature(signature, signatureSize);

   } while (0);

   /* Free the context now, which forces user to reenter the password next
      time the SIGSignPrepare call is made. */

   if (context != NULL) SIGDisposeContext(context);

   if (dataBuffer != NULL) DisposPtr(dataBuffer);

   if (signature != NULL) DisposPtr((Ptr)signature);

   return error;
}
```

Verifying a Full Signature

When the user wants to verify the signature on a document or a portion of a document, you are responsible for knowing the location, processing order, and extent of the data to be verified, and for locating the full signature that applies to that data. The Digital Signature Manager expects you to provide a pointer to the data, a pointer to the full signature, the signature size, and a context pointer.

To verify a signature, follow these steps:

1. First, call the `SIGNewContext` function to allocate and initialize a context. The function returns a context pointer. If the Digital Signature Manager is not already in memory, the Operating System loads it into memory.
2. Call the `SIGVerifyPrepare` function, passing it a pointer to the signature to be verified, the signature size, and the context pointer. The `SIGVerifyPrepare` function periodically calls a callback routine that you may provide, so that you can notify the user of the progress of the operation or perform other background tasks.

The SIGVerifyPrepare function verifies the authenticity and currency of all certificates in the certificate set and returns the kSIGSignerErr result code if any of the certificates have been altered.

3. Call the SIGProcessData function as many times as necessary to process all of your data. Either move a pointer through your data each time you call the function, or create a buffer and put blocks of data into it. The SIGProcessData function creates a digest of the data whose signature is to be verified.
4. Call the SIGVerify function. It completes the digest and compares it with the digest in the signature.
5. Check the result code returned by the SIGVerify function to see if the verification was successful. A result code of noErr means the verification was successful and the signature is valid. A result code of kSIGInvalidCredentialErr means the verification was successful but the signer's credential is either pending or expired. A result code of kSIGVerifyFailedErr means the verification failed.
6. Call the SIGDisposeContext function to release the memory used by the verification routines and to release the Digital Signature Manager from memory. To verify another signature, you must start over from step 1.

After verifying a signature, you may want to get information about it and present that to the user. See "Getting Information From a Signature or Certificate" beginning on page 6-19.

Listing 6-2 shows an example of a function that verifies a signature. At the beginning of this function is a call to DoRetrieveSignature, an application-defined function that loads the signature in from where it is stored. The DoVerifyData function also requires an application-defined function named DoGetDataToProcess to cycle through all the data to be verified.

Listing 6-2 A sample signature-verification routine

```
OSErr DoVerifyData()
{
   OSErr             error;
   Boolean           moreToVerify;
   Size              signatureSize;
   Size              dataSize;
   SIGSignaturePtr   signature = NULL;
   SIGContextPtr     context = NULL;
   Ptr               dataBuffer = NULL;

   do {
      /* Get the signature and its size from wherever your application saved
         it. */
```

```
      if (error = DoRetrieveSignature(&signature, &signatureSize))
         break;

      /* Allocate a new context and prepare it for verifying. */
      if (error = SIGNewContext(&context))
         break;

      if (error = SIGVerifyPrepare(context, signature, signatureSize,
            (SIGStatusProcPtr)NULL))
         break;

      /* Get the data to be verified in your application-specific way, and
         pass it to the toolbox to generate a digest for verification. */

      /* NOTE: DoGetDataToProcess can be the same function for signing and
         verifying. */

      do {
         if (error = DoGetDataToProcess(&dataBuffer, &dataSize,
&moreToVerify))
            break;

         if (error = SIGProcessData(context, dataBuffer, dataSize))
            break;

         } while (moreToVerify);

      if (error)/* if encountered error above, go all the way out */
         break;

      /* Now, perform verification. */
      if (error = SIGVerify(context))
         break;

      /* Finally, display the name of the signer of the data. NOTE: you can
         call SIGShowSigner even if a kSIGInvalidCredentialErr was returned
         from SIGVerify. */

      error = SIGShowSigner(context, "\p");

   } while (0);

   /* Free the context. */

   if (context) SIGDisposeContext(context);

   if (dataBuffer) DisposPtr(dataBuffer);
```

```
    if (signature) DisposPtr((Ptr)signature);

    return error;
}
```

Creating a Simple (Unencrypted) Digest

As a convenience utility, the Digital Signature Manager allows you to create a digest of a document (or any stream of data you manipulate). The digest thus created cannot be encrypted or turned into a signature of the document, but its value as a sophisticated checksum makes it useful for other purposes, such as checking reliability in data transmission. And, like any data, the digest itself can be signed to ensure its integrity.

As one example, assume you are transmitting a massive document in separate blocks across a network. You want to ensure that the blocks are assembled in the right order at the receiving end. You can construct digests of individual blocks as they are sent and, after all the blocks have been sent, concatenate all the digests into a single file and send it. If the recipient has built a file of concatenated digests as the received blocks are reassembled, the concatenated digests should match each other if there has been no transmission or reassembly error. This method avoids the necessity of processing massive amounts of data at once, as would be necessary to create or verify a single signature on the entire document.

Creating a digest is similar to creating a signature. You first call the `SIGNewContext` function, then you call the `SIGDigestPrepare` function. Next you call `SIGProcessData` as many times as necessary to process all of your data. Finally you call `SIGDigest`, which returns the finished digest to you.

To create another digest, call the `SIGProcessData` as many times as necessary, then call the `SIGDigest` function. When you are finished creating digests, call the `SIGDisposeContext` function.

Getting Information From a Signature or Certificate

When you add a signature to a block of data or verify a signature, you are informed only of the success or failure of the operation. Neither you nor the user has direct access to any information in the signature—not even the name of the signer.

If you want to know (or tell the user) who created a signature, when it was signed, who issued the certificate to the signer, whether the signer's certificate has expired, or any other information available from the signature, you can call Digital Signature Manager routines that return that information.

After you successfully verify a signature, you can display a dialog box containing the full distinguished name of the signer by calling the `SIGShowSigner` function (page 6-46).

After you successfully verify a signature, after you call the `SIGSignPrepare` function to initiate the signing process, or after you call the `SIGSign` function to sign a block of data, you can call the `SIGGetSignerInfo` function (page 6-48) to determine when a block of data was signed and how many certificates constitute the certificate set for the

signature. The SIGGetSignerInfo function also tells you whether the entire certificate set is valid and, if not, whether it has expired or has not yet become valid.

You can use the SIGGetCertInfo function (page 6-49) to obtain the beginning and ending dates of a certificate's validity, and the total number of attributes in the distinguished names of the certificate's signer and issuer.

You can use the SIGGetCertNameAttributes function (page 6-51) and the SIGGetCertIssuerNameAttributes function (page 6-52) to obtain the attributes that compose the distinguished names of the certificate's signer and issuer.

To obtain complete information on a newly applied or verified signature, you might follow a procedure something like this:

1. Call the SIGGetSignerInfo function to get the date of the signing and the total number of certificates in the full signature.
2. Call the SIGGetCertInfo function for the first certificate in the signature to get the dates for which the certificate is valid, the serial number of the certificate, and the number of name attributes in the distinguished name of the certificate.
3. Call the SIGGetCertNameAttributes function once for each name attribute in the certificate to get the full distinguished name for each certificate. This function returns the string for each attribute and the type of the attribute. It also specifies whether the attribute is the same level in the name hierarchy as the previous attribute (See "About Public-Key Certificates" beginning on page 6-8 for a description of distinguished names.)
4. Repeat steps 2 and 3 for each additional certificate in the certificate set. The certificates are always in order: the signer's certicate is first, the issuer of the signer's certificate is next, and so forth.
5. You can use the SIGGetCertIssuerNameAttributes function to get the full distinguished name of the prime issuer.

Listing 6-3 is an example of a function that extracts information from a certificate set. The DoDisplayCertificateSet sample function displays the name of the signer of a verified signature and searches through a certificate set, displaying information about the owner of each certificate. The DoDisplayCertificateSet function assumes that the input is a valid context that has gone through a successful call to either the SIGVerify, SIGSignPrepare, or SIGSign functions.

In addition, the DoDisplayCertificateSet function requires the following support functions to actually display the data to the user: DoDisplaySignatureInfo, DoDisplayCertificateInfo, and DoDisplayCertNameAttribute.

Listing 6-3 A sample routine that returns information in a certificate set

```
OSErr DoDisplayCertificateSet(SIGContextPtr context)
{
   unsigned short          attrIndex;
   SIGNameAttributesInfo   attrInfo;
   unsigned short          certIndex;
   SIGCertInfo             certInfo;
   SIGSignerInfo           signerInfo;
   OSErr                   error;

   do {
      /* Get and display general signature information first. */

      if (error = SIGGetSignerInfo(context, &signerInfo))
         break;

      DoDisplaySignatureInfo(&signerInfo);

      /* Traverse entire certificate set and for each certificate, display
         its certificate information. Then traverse the name attribute
         information for that certificate and display the attributes. */

      for (certIndex = kSIGSignerCertIndex; certIndex < signerInfo.certCount;
            certIndex++)
         {
         if (error = SIGGetCertInfo(context, certIndex, &certInfo))
            break;
         DoDisplayCertificateInfo(&certInfo);
         for (attrIndex = 0; attrIndex < certInfo.certAttributeCount;
               attrIndex++)
         {
            if (error = SIGGetCertNameAttributes(context, certIndex,
                  attrIndex, &attrInfo))
               break;

            DoDisplayCertNameAttribute(&attrInfo);
         }
      }

      /* Finally, display the root issuers' name attributes. */
```

```
      /* NOTE: there's no certificate information for the root; it's always
         valid.*/

      for (attrIndex = 0; attrIndex < certInfo.issuerAttributeCount;
            attrIndex++)
      {
         if (error = SIGGetCertIssuerNameAttributes(context, certIndex-1,
               attrIndex, &attrInfo))
            break;

         DoDisplayCertNameAttribute(&attrInfo);
      }
   } while (0);
   return error;
}
```

Dealing With Standard Signatures in Files

On the desktop, a user can add a standard signature to any file by dragging the icon for the file to be signed onto the icon of his or her signer file. When a user signs a file this way, the Digital Signature Manager adds a resource of type `'dsig'` to the resource fork of the file. Whenever you open a file, you should use the `SIGFileIsSigned` function to determine if the file contains such a signature. If a file contains a standard signature, you should not allow the user to alter the file without first displaying a dialog box warning that the file has been signed and that changing the file in any way will invalidate the signature. You should also not make any changes of your own to the file, such as saving a new window position, unless the user has chosen to allow changes that invalidate the signature.

All resources in the file are also signed, except any resources of type `'nods'` (no digital signature). You can store anything that you don't want to be signed in this resource, such as a new window position, and verification will still work.

Note

The '`dsig`' resource is mentioned here for your information only and may change in the future. Therefore, any attempt to manipulate this resource directly could cause incompatibilities with future versions of the Digital Signature Manager. u

You can verify a standard signature by calling the `SIGVerifyFile` function. You can add a standard signature to a file from within your application or replace an existing standard signature in a file by calling the `SIGSignFile` function. A user can also use the Finder to verify the signature in a file signed in this way.

You must call the `SIGNewContext` function before you call either the `SIGSignFile` function or the `SIGVerifyFile` function. To add a standard signature to a file, you must call the `SIGSignPrepare` function and the `SIGSignFile` function. The `SIGSignFile` function processes the data and adds the signature to the file. To verify a standard signature, you call the `SIGVerifyFile` function. You do not have to call the `SIGProcessData` function when you are working with standard signatures in files. You cannot add a standard signature to a file or verify such a signature if the file is in use.

Digital Signature Manager Reference

This section describes the data types and routines provided by the Digital Signature Manager and the interface to a status callback routine that you may provide.

Constants and Data Types

This section describes the constants and data types that are used by the `SIGGetSignerInfo` and `SIGGetCertInfo` functions to return information about signers and certificates. The `SIGDigestData` data type is described with the `SIGDigest` routine on page 6-44.

Signer Information Structure

The `SIGGetSignerInfo` function (page 6-48) uses a signer information structure to return information about a signature. The signer information structure is defined by the `SIGSignerInfo` data type.

```
struct SIGSignerInfo
{
   unsigned long        signingTime;      /* local sign time */
   unsigned long        certCount;        /* # of certificates
                                             in the set */
   unsigned long        certSetStatusTime;/* expiration time*/
   SIGSignatureStatus   signatureStatus;  /* certificate status */
};
```

Field descriptions

`signingTime` The time at which the data was signed. The time is in standard Macintosh format: the number of seconds elapsed since Midnight, January 1, 1904. The time is converted from Greenwich Mean Time (GMT) to the local time of the user's Macintosh. To convert to local time, the AOCE toolbox uses the local system clock and Map control panel on the signer's Macintosh computer. Thus, the time cannot be considered reliable.

`certCount` The number of certificates in the certificate set.

`certSetStatusTime`
If all the certificates in the certificate set are valid, this field holds the expiration time of the certificate that will be the first to expire. If one or more certificates have expired, this field holds the time when the first certificate in the set expired. If none of the certificates have expired but one or more is not yet valid, this field holds the time that the last pending certificate will become valid. The time is given as the number of seconds elapsed since midnight, January 1, 1904.

`signatureStatus`
If all the certificates in the certificate set are valid, this field holds the value `kSIGValid`. If any of the certificates have expired since the data was signed, this field holds the value `kSIGExpired`. If any of the certificates had already expired before the data was signed, this field holds the value `kSIGInvalid`. If none of the certificates have expired but any have not yet become valid, this field holds the value `kSIGPending`.

This field can have any of the following values:

```
enum {
   kSIGValid,     /* all valid */
   kSIGPending,   /* none expired; some pending
                     or unknown */
   kSIGExpired,   /* some expired, unknown, or
                     pending */
   kSIGInvalid    /* some invalid, pending, expired
                     or unknown */
};

typedef unsigned short SIGSignatureStatus;
```

Certificate Information Structure

The `SIGGetCertInfo` function (page 6-49) uses a certificate information structure to return information about a specific certificate in a signature. The certificate information structure is defined by the `SIGCertInfo` data type.

```
struct SIGCertInfo
{
   unsigned long  startDate;           /* validity start date */
   unsigned long  endDate;             /* validity end date */
   SIGCertStatus  certStatus;          /* certificate status */
   unsigned long  certAttributeCount;  /* number of name
                                          attributes in cert*/
   unsigned long  issuerAttributeCount;/* # of name attributes
                                          in cert's issuer */
   Str255         serialNumber;        /* cert serial number */
};
```

Field descriptions

`startDate` The time at which the certificate became (or will become) valid. The time is in standard Macintosh format: the number of seconds elapsed since midnight, January 1, 1904.

`endDate` The expiration time of the certificate in seconds since midnight, January 1, 1904.

`certStatus` The status of the certificate: `kSIGValid`, `kSIGPending`, or `kSIGExpired`.

`certAttributeCount`
The number of attributes in the distinguished name for this certificate (see Table 6-1 on page 6-9). You can use the `SIGGetCertNameAttributes` function (page 6-51) to list the attributes.

`issuerAttributeCount`
The number of attributes in the distinguished name of the issuer of this certificate (see Table 6-1 on page 6-9). You can use the `SIGGetCertIssuerNameAttributes` function (page 6-52) to list the attributes.

`serialNumber` A certificate number assigned by the issuer.

Standard Signature Icon Suite

The Digital Signature Manager provides an icon suite that you use to represent a digital signature in your document. This suite contains all bit depths and sizes.

```
#define kSIGSignatureIconResID          -16797

#define kSIGValidSignatureIconResID     -16799

#define kSIGInvalidSignatureIconResID   -16798
```

Name Attribute Information Structure

The `SIGGetCertNameAttributes` function (page 6-51) and the `SIGGetCertIssuerNameAttributes` function (page 6-52) use a name attribute information structure to return information about a name attribute. The name attribute information structure is defined by the `SIGNameAttributesInfo` data type.

```
struct SIGNameAttributesInfo
{
   Boolean                 onNewLevel;
   SIGNameAttributeType    attributeType;
   ScriptCode              attributeScript;
   Str255                  attribute;
};
```

Field descriptions

`onNewLevel` A Boolean value that indicates whether the name attribute is at the same level of the name hierarchy as the previous value returned.

`attributeType` The type of attribute returned.

`attributeScript` The script code for the name attribute. Script codes are defined by the Script Manager.

`attribute` The name attribute value.

The `attributeType` field can have any of the following values:

```
enum {
            kSIGCountryCode,
            kSIGOrganization,
            kSIGStreetAddress,
            kSIGState,
            kSIGLocality,
            kSIGCommonName,
            kSIGTitle,
```

```
            kSIGOrganizationUnit,
            kSIGPostalCode
};

typedef unsigned short SIGNameAttributeType;
```

You can use the hierarchy information in the `onNewLevel` parameter to arrange the distinguished name for display to the user.

Distinguished names and name hierarchies are described in detail in "About Public-Key Certificates" beginning on page 6-8.

Digital Signature Manager Functions

You can use Digital Signature Manager functions to perform the following tasks: creating a signature (page 6-31), verifying a signature (page 6-38), creating an unencrypted digest (page 6-43), signing a file (page 6-36), and verifying a file (page 6-41). All of these tasks, except signing and verifying a file, require you to process data (page 6-30). You begin each of these operations by creating a new context and end the operation by disposing of the context (page 6-28). After you prepare a context for a signature, create a signature, or verify a signature, you can extract information from the certificate or signature (page 6-45).

Assembly-Language Interface

To call a Digital Signature Manager function from assembly language, you must do the following:

1. Allot space for the function result and all routine parameters (in Pascal calling-convention order) on the stack.
2. In the D0 register, put a long word consisting of the parameter word count for the routine followed by the routine selector. The parameter word count indicates how many words of parameters you are placing on the stack; for example, if the function has two parameters and each is a pointer, the parameter word count for the function is $0004.
3. Call the Digital Signature Manager trap, $AA5D.

Each routine description in the following sections lists the parameter word count and routine selector for that routine.

Creating and Disposing of a Context

The Digital Signature Manager uses a private data structure called a **context** to hold information and the results of calculations while it is processing data. Before you call Digital Signature Manager routines to perform a specific task, you must call the `SIGNewContext` function to create a context and obtain a context pointer. To free the memory used by the context, call the `SIGDisposeContext` function.

You can use a new context for any type of operation; however, once you have called the first task-specific function (`SIGSignPrepare`, `SIGVerifyPrepare`, or `SIGDigestPrepare`), you can use the context only with other functions associated with that task. Table 6-2 on page 6-12 summarizes the Digital Signature Manager tasks and the functions required to perform each task.

SIGNewContext

The `SIGNewContext` function creates a new context and returns a context pointer.

```
pascal OSErr SIGNewContext (SIGContextPtr *context);
```

`context` A pointer to the new context created by this function.

DESCRIPTION

You must pass the context pointer returned by this function to either the `SIGSignPrepare`, `SIGVerifyPrepare`, or `SIGDigestPrepare` function.

SPECIAL CONSIDERATIONS

This function causes the Digital Signature Manager to be loaded into memory if it is not already in memory.

This function may move or purge memory; you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$076C

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter value
`memFullErr`	–108	Not enough room in heap

SEE ALSO

Use the `SIGDisposeContext` function (described next) to dispose of a context.

The `SIGNewContext` function is normally followed by either the `SIGSignPrepare` function (page 6-31), the `SIGVerifyPrepare` function (page 6-38), or the `SIGDigestPrepare` function (page 6-43).

SIGDisposeContext

The `SIGDisposeContext` function frees the memory used by a context.

```
pascal OSErr SIGDisposeContext (SIGContextPtr context));
```

`context` A pointer to the context you wish to dispose of.

DESCRIPTION

You must call the `SIGDisposeContext` function to dispose of a context when you are finished creating a signature, verifying a signature, creating a digest, or extracting information from a signature or certificate.

SPECIAL CONSIDERATIONS

Because this function removes the Digital Signature Manager (as well as the context) from memory, you must call this function even if the previous function returned an error.

This function may move or purge memory; you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$076D

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter value
`kSIGOperationIncompatibleErr`	–1970	Context in use for different type of operation

Processing Data to Generate a Digest

To process data during the creation or verification of a signature, or during the creation of a digest, call the `SIGProcessData` function one or more times.

SIGProcessData

The `SIGProcessData` function processes the data passed to it and revises the digest accordingly.

```
pascal OSErr SIGProcessData (SIGContextPtr context,
                             const void *data, Size dataSize);
```

`context` A pointer to the context that you passed to the `SIGSignPrepare`, `SIGVerifyPrepare`, or `SIGDigestPrepare` function.

`data` A pointer to a buffer containing the data to be processed.

`dataSize` The number of bytes of data to be processed.

DESCRIPTION

Call the `SIGProcessData` function to generate a digest for a set of data. If you have more data than is convenient to process all at once, you can call the function several times, passing it a block of any size each time. Note, however, that it is more efficient to process data in large blocks than in small blocks.

You can place each block of data into a buffer, or you can change the `data` parameter each time to point at the next starting position in your data. You are responsible for keeping track of where the data is and how much of it to process during each call to the `SIGProcessData` function, and for knowing when all the data has been processed.

The data must be processed in the same order during the corresponding sign and verify operations but need not be processed in blocks of the same size. To the `SIGProcessData` function, the data is a continuous byte stream.

SPECIAL CONSIDERATIONS

You can call the `SIGProcessData` function at interrupt time; it does not move or purge memory.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0006	$0774

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter value
kSIGOperationIncompatibleErr	–1970	Context in use for different type of operation
kSIGInternalsErr	–1977	Bad digest, context, or signature

SEE ALSO

The SIGProcessData function is preceded by a call to SIGSignPrepare (page 6-31), SIGVerifyPrepare (page 6-38), or SIGDigestPrepare (page 6-43).

After calling SIGProcessData, you call either SIGSign (page 6-34), SIGVerify (page 6-40), or SIGDigest (page 6-44).

Creating a Signature

To create a full signature, first call the SIGNewContext function (page 6-28) to create a new context, then call the SIGSignPrepare function (described next).

Next, to sign some portion of the data in a file, call the SIGProcessData function (page 6-30) as many times as necessary to process all the data. When you are finished processing the data, call the SIGSign function (page 6-34). To create additional signatures for the same signer, you can call the SIGProcessData and SIGSign functions again, without first creating a new context or calling the SIGSignPrepare function.

If you want to add a standard signature to a file, call the SIGSignFile function (page 6-36) instead of the SIGProcessData and SIGSign functions. To add signatures to additional files, you can call the SIGSignFile function again, without first creating a new context or calling the SIGSignPrepare function.

When you no longer need the context you used for creating the signatures, call the SIGDisposeContext function (page 6-29).

This section describes the SIGSignPrepare, SIGSign, and SIGSignFile functions.

SIGSignPrepare

The SIGSignPrepare function notifies the Digital Signature Manager that you are about to create a signature.The function returns the size that the full signature will be when it is created.

```
pascal OSErr SIGSignPrepare (SIGContextPtr context,
                             const FSSpec *signerFile,
                             ConstStr255Param prompt,
                             Size *signatureSize);
```

`context` A pointer to the context that the Digital Signature Manager will use while creating the signature. Call the `SIGNewContext` function to obtain the context pointer.

`signerFile`
A pointer to a file-specification structure for the user's signer file. If you specify `NULL` for this parameter, the function opens the previously used signer file, or, if there is no record of a previously used signer file, the function displays a Standard File dialog box prompting the user for the location of a signer file.

`prompt` A string to display in the dialog box that prompts the user for a password. Pass a zero-length Pascal-style string to use the default prompt.

`signatureSize`
A pointer to the size of the signature that is to be created. The function returns this parameter.

DESCRIPTION

The `SIGSignPrepare` function displays a password-prompting dialog box into which the user types the private-key password. The function displays a Standard File dialog box prompting the user for a signer file if you do not specify a signer file in the `signerFile` parameter and there is no default signer file.

If you pass `NULL` in the `signerFile` parameter, the first time the user signs something, the function displays a Standard File dialog box prompting the user for the location of the signer file. The Digital Signature Manager then stores an alias to that file in the user's Preferences folder. The next time you specify `NULL` in the `signerFile` parameter, the `SIGSignPrepare` function uses that signer file as the default and does not display the standard file dialog box.

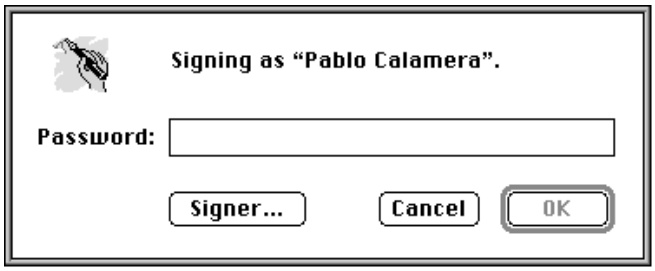
If you already know the location of the user's signer file, you can bypass the Standard File dialog box by passing a pointer to the signer file's file-specification structure in the `signerFile` parameter. You can also use this procedure to override the default signer file.

The `prompt` parameter can contain whatever string you wish displayed in the dialog box to prompt the user for a private-key password. Use the parameter-text designator `^1` for the user's name; the Digital Signature Manager replaces `^1` in your string with the user's common name or title (depending on whether the user is signing as a person or as an organizational role—see Table 6-1 on page 6-9) as it appears in the signer file. If you pass a zero-length string, the function uses the default string.

The password-prompting dialog box also contains a Signer button that allows the user to select a different signer file. Figure 6-5 shows how the dialog box would appear to a user whose common name is Pablo Calamera.

Note
If you specify a signer file to use, the password dialog box does not contain a Signer button allowing users to switch signer files. u

Figure 6-5 The password-prompting dialog box



This function returns the size the signature will be once it is created. Use the result to allocate memory for the signature before calling the `SIGSign` function.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call it at interrupt time.

This function is stack-intensive, requiring approximately 7 KB of memory for its stack.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0008	$076E

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter value
`memFullErr`	–108	Not enough room in heap zone
`userCanceledErr`	–128	User canceled password-prompt dialog box
`kSIGOperationIncompatibleErr`	–1970	Context in use for different type of operation
`kSIGSignerErr`	–1975	Problem with the signer file or signature
`kSIGPasswordErr`	–1976	Password is incorrect
`kSIGInternalsErr`	–1977	Bad digest, context, or signature
`kSIGContextPrepareErr`	–1979	Context already prepared by `SIGVerifyPrepare`, `SIGSignPrepare`, or `SIGDigestPrepare`
`kSIGConversionErr`	–1981	Unable to convert an attribute to Macintosh format
`kSIGSignerNotValidErr`	–1982	Signer file has either expired or is not yet valid

SEE ALSO

Before you call the `SIGSignPrepare` function, you must call the `SIGNewContext` function (page 6-28) to create a new context.

After calling the `SIGSignPrepare` function, you can extract information from the certificate set; see "Getting Information From a Signature or Certificate" beginning on page 6-45.

After you call the `SIGSignPrepare` function, call the `SIGProcessData` function (page 6-30) as many times as necessary to process all the data.

SIGSign

The `SIGSign` function creates a full signature for the data most recently processed by the `SIGProcessData` function, using signer-file information from the most recent call to the `SIGSignPrepare` function.

```
pascal OSErr SIGSign (SIGContextPtr context,
                      SIGSignaturePtr signature,
                      SIGStatusProcPtr statusProc);
```

`context` The context pointer that you passed to the `SIGSignPrepare` function.

`signature` A pointer to a buffer you provide to hold the signature returned by the function. Use the result of the `SIGSignPrepare` function to allocate a buffer of the correct size.

`statusProc`
A pointer to a callback routine you may provide to notify the user of the progress of the signature creation or to perform other background tasks. Specify `NULL` for this parameter if you do not wish to provide a callback routine.

DESCRIPTION

Call this function after having called the `SIGProcessData` function enough times to finish processing the document or data that is to be signed. After creating a signature, `SIGSign` places it in the buffer pointed to by the `signature` parameter.

Because the `SIGSign` function can take a long time to complete, you can provide a pointer to a callback routine to notify the user of the progress of the operation, allow the user to cancel it, and perform background tasks such as spinning the cursor.

To create additional signatures for the same signer, you can call the `SIGProcessData` and `SIGSign` functions again, without first creating a new context or calling the `SIGSignPrepare` function. Call the `SIGDisposeContext` function when you have finished creating signatures with that signer.

Note
You should call the `SIGDisposeContext` function as soon as possible after you finish creating signatures so that the Operating System can free the memory used by the Digital Signature Manager. u

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0006	$076F

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter value
`userCanceledErr`	–128	User canceled signing process
`kSIGOperationIncompatibleErr`	–1970	Context in use for different type of operation
`kSIGSignerErr`	–1975	Problem with the signer file or signature
`kSIGInternalsErr`	–1977	Bad digest, context, or signature

SEE ALSO

You call the `SIGSign` function after calling `SIGSignPrepare` (page 6-31) to initiate the signing process and `SIGProcessData` (page 6-30) to process the data.

You may provide a callback status routine when you call the `SIGSign` function; see "Application-Defined Function" on page 6-54.

After calling the `SIGSign` function, you can extract information from the signature; see "Getting Information From a Signature or Certificate" beginning on page 6-45.

As soon as possible after you finish creating signatures, call the `SIGDisposeContext` (page 6-29) function to dispose of the context and to allow the Operating System to remove the Digital Signature Manager from memory.

SIGSignFile

The `SIGSignFile` function adds a standard signature to a file.

```
pascal OSErr SIGSignFile (SIGContextPtr context,
                    Size signatureSize,
                    const FSSpec *fileSpec,
                    SIGStatusProcPtr statusProc);
```

`context` The context pointer that you passed to the `SIGSignPrepare` function.

`signatureSize`
The size of the signature as returned by the `SIGSignPrepare` function.

`fileSpec` A pointer to the file system specification structure for the file to which you want to add a signature.

`statusProc`
A pointer to a callback routine you may provide to notify the user of the progress of the signature creation or to perform other background tasks. Specify `NULL` for this parameter if you do not wish to provide a callback routine.

DESCRIPTION

The `SIGSignFile` function processes a signature for a complete file and places it in the resource fork of the file as a resource of type `'dsig'`. You must call the `SIGSignPrepare` function before calling the `SIGSignFile` function.

Note
The '`dsig`' resource is mentioned here for your information only. Because it may change in the future, you should not attempt to manipulate this resource directly. Any change could cause incompatibilities with future versions of the Digital Signature Manager. u

A signature you add to a file using this function is identical to one added by the Finder when the user drags the icon for the file onto the icon of their signer file. If the file is already signed, the `SIGSignFile` function creates a new signature and replaces the old one.

All resources in the file are also signed, except any resources of type `'nods'` (no digital signature). You can store anything that you don't want to be signed in this resource, such as a new window position, and verification will still work.

Because the `SIGSignFile` function can take a long time to complete, you can provide a pointer to a callback routine to perform background tasks such as spinning the cursor and to allow the user to cancel the operation.

To sign additional files for the same signer, you can call the `SIGSignFile` function again, without first creating a new context or calling the `SIGSignPrepare` function.

Call the `SIGDisposeContext` function when you are finished signing files for that signer.

Note

You should call the `SIGDisposeContext` function as soon as possible after you finish creating signatures so that the Operating System can free the memory used by the Digital Signature Manager. u

IMPORTANT

The `SIGSignFile` function will not work on a file that is open. s

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0008	$09C5

RESULT CODES

`noErr`	0	No error
`dirFulErr`	–33	Directory full
`dskFulErr`	–34	Disk full
`nsvErr`	–35	No such volume
`ioErr`	–36	I/O error
`bdNamErr`	–37	Bad name error
`tmfoErr`	–42	Too many files open
`fnfErr`	–43	File not found
`fBsyErr`	–47	File is busy
`opWrErr`	–49	File already open with write permission
`paramErr`	–50	Illegal parameter value
`wrPermErr`	–61	File not available
`memFullErr`	–108	Not enough room in heap zone
`dirNFErr`	–120	Directory not found
`userCanceledErr`	–128	User canceled signing process
`addResFailed`	–194	Adding resource failed
`rmvResFailed`	–196	Removing resource failed
`kSIGOperationIncompatibleErr`	–1970	Context in use for different type of operation
`kSIGInternalsErr`	–1977	Bad digest, context, or signature
`afpAccessDenied`	–5000	Disk full

SEE ALSO

You call the `SIGSignFile` function after initiating the signing process with the `SIGSignPrepare` function (page 6-31).

You may provide a callback status routine when you call the SIGSign function; see “Application-Defined Function” on page 6-54.

You can use the SIGFileIsSigned function (page 6-45) to determine if a file already contains a standard signature.

As soon as possible after you finish creating signatures, call the SIGDisposeContext function (page 6-29) to dispose of the context and to allow the Operating System to remove the Digital Signature Manager from memory.

Verifying a Signature

When you use the Digital Signature Manager to verify a signature, it checks the validity of the certificate set, creates a digest of the data whose signature you wish to verify, and compares that digest to the digest in the signature.

To verify a signature of some portion of data in a file, first call the SIGNewContext function (page 6-28), then call the SIGVerifyPrepare function (described next). Next, call the SIGProcessData function (page 6-30) as many times as necessary to prepare a digest of the data. When you have finished processing the data, call the SIGVerify function (page 6-40) to compare the digest you prepared with the one in the signature.

To verify a standard signature in a file (that is, one added by the Finder or by the SIGSignFile function), first call the SIGNewContext function to create a new context, then call the SIGVerifyFile function (page 6-41).

When you are finished with the context you used for verifying the signature, call the SIGDisposeContext function (page 6-29).

This section describes the SIGVerifyPrepare function, the SIGVerify function, and the SIGVerifyFile function.

SIGVerifyPrepare

The SIGVerifyPrepare function notifies the Digital Signature Manager that you have a signature to be verified and initializes the verification process.

```
pascal OSErr SIGVerifyPrepare (SIGContextPtr context,
                               SIGSignaturePtr signature,
                               Size signatureSize,
                               SIGStatusProcPtr statusProc);
```

`context` A pointer to the context that the Digital Signature Manager will use while verifying the signature. Call the SIGNewContext function to obtain the context pointer.

`signature` A pointer to the full signature that is to be verified.

`signatureSize` The size of the signature that is to be verified.

`statusProc`

A pointer to a callback routine you may provide to notify the user of the progress of the verification operation or perform other background tasks. Specify `NULL` for this parameter if you do not wish to provide a callback routine.

DESCRIPTION

You must provide the `SIGVerifyPrepare` function with a pointer to the signature to be verified and the size of the signature. You may release the memory used by the signature after the `SIGVerifyPrepare` function has completed.

Because the `SIGVerifyPrepare` function verifies each certificate in the certificate set and reads in the digest, it can take a long time to complete. You can provide a pointer to a callback routine to perform background tasks such as spinning the cursor and to allow the user to cancel the operation.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call it at interrupt time.

This function is stack-intensive, requiring approximately 7 KB of memory for its stack.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0008	$0770

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter value
`memFullErr`	–108	Not enough room in heap zone
`kSIGSignerErr`	–1975	Problem with the signer file or signature
`kSIGInternalsErr`	–1977	Bad digest, context, or signature
`kSIGContextPrepareErr`	–1979	Context already prepared by `SIGVerifyPrepare`, `SIGSignPrepare`, or `SIGDigestPrepare`
`kSIGNoDigestErr`	–1980	No digest in the signature

SEE ALSO

Before you call the `SIGVerifyPrepare` function, you must call the `SIGNewContext` function (page 6-28) to create a new context.

You may provide a callback status routine when you call the `SIGVerifyPrepare` function; see “Application-Defined Function” on page 6-54.

After you call the `SIGVerifyPrepare` function, call the `SIGProcessData` function (page 6-30) as many times as necessary to process all the data whose signature you wish to verify.

SIGVerify

The `SIGVerify` function tests the validity of the specified signature. To do so, it compares the digest in the signature with the digest you prepared by calling the `SIGProcessData` function. It also checks the validity of the credentials in the signature's certificate set.

```
pascal OSErr SIGVerify (SIGContextPtr context);
```

`context` The context pointer that you passed to the `SIGVerifyPrepare` function.

DESCRIPTION

Call this function after having called the `SIGProcessData` function enough times to finish processing data whose signature is to be verified. Note that you must process the data in the same sequence that it was processed when the signature was created.

Check the result code from this function to see if the signature verification was successful.

Note
You should call the `SIGDisposeContext` function as soon as possible after you finish verifying a signature so that the Operating System can free the memory used by the Digital Signature Manager. u

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$0771

RESULT CODES

`noErr`	0	No error
`ParamErr`	–50	Illegal parameter value
`kSIGOperationIncompatibleErr`	–1970	Context in use for different type of operation
`kSIGVerifyFailedErr`	–1972	Verification failed
`kSIGInvalidCredentialErr`	–1973	Verified OK but credential pending or expired
`kSIGInternalsErr`	–1977	Bad digest, context, or signature

SEE ALSO

You call the `SIGVerify` function after calling `SIGVerifyPrepare` (page 6-38) to initiate the signing process and `SIGProcessData` (page 6-30) to process the data.

After a successful verification, you can extract information from the signature; see "Getting Information From a Signature or Certificate" beginning on page 6-45.

As soon as possible after calling the `SIGVerify` function, call the `SIGDisposeContext` (page 6-29) function to dispose of the context and allow the Operating System to remove the Digital Signature Manager from memory.

SIGVerifyFile

The `SIGVerifyFile` function verifies a standard signature in a file.

```
pascal OSErr SIGVerifyFile (SIGContextPtr context,
                            const FSSpec *fileSpec,
                            SIGStatusProcPtr statusProc);
```

`context` A pointer to the context that the Digital Signature Manager will use while verifying the signature. Call the `SIGNewContext` function to obtain the context pointer.

`fileSpec` A pointer to the file system specification structure for the file whose signature you want to verify.

`statusProc`
A pointer to a callback routine you may provide to notify the user of the progress of the verification operation or perform other background tasks. Specify `NULL` for this parameter if you do not wish to provide a callback routine.

DESCRIPTION

If a file contains a standard signature, you can use the `SIGVerifyFile` function to verify it.

Because the `SIGVerifyFile` function verifies each certificate in the certificate set and reads in the digest, it can take a long time to complete. You can provide a pointer to a callback routine to perform background tasks such as spinning the cursor and to allow the user to cancel the operation.

Note
You should call the `SIGDisposeContext` function as soon as possible after you finish verifying a signature so that the Operating System can free the memory used by the Digital Signature Manager. u

IMPORTANT

You cannot use the `SIGVerifyFile` function on a file that is in use by another application. s

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0006	$09C6

RESULT CODES

`noErr`	0	No error
`dirFulErr`	–33	Directory full
`dskFulErr`	–34	Disk full
`nsvErr`	–35	No such volume
`ioErr`	–36	I/O error
`bdNamErr`	–37	Bad name error
`tmfoErr`	–42	Too many files open
`fnfErr`	–43	File not found
`fBsyErr`	–47	File is busy
`opWrErr`	–49	File already open with write permission
`paramErr`	–50	Illegal parameter value
`permErr`	–54	Permissions error on file open
`memFullErr`	–108	Not enough room in heap zone
`dirNFErr`	–120	Directory not found
`kSIGSignerErr`	–1975	Problem with the signer file or signature
`kSIGInternalsErr`	–1977	Bad digest, context, or signature
`kSIGContextPrepareErr`	–1979	Context already prepared by `SIGVerifyPrepare`, `SIGSignPrepare`, or `SIGDigestPrepare`
`kSIGNoDigestErr`	–1980	No digest in the signature
`kSIGNoSignature`	–1983	Standard file signature not found

SEE ALSO

Before you call the `SIGVerifyFile` function, you must call the `SIGNewContext` function (page 6-28) to create a new context.

You may provide a callback status routine when you call the `SIGVerifyFile` function; see "Application-Defined Function" on page 6-54.

You can call the `SIGFileIsSigned` function (page 6-45) to determine if a file contains a standard signature.

Creating a Digest

You can create an unencrypted digest of a document without creating a digital signature. To create a digest, first call the `SIGNewContext` function (page 6-28) to create a new context, then call the `SIGDigestPrepare` function (described next). Next, call the `SIGProcessData` function (page 6-30) as many times as necessary to process all the data. When you have finished processing the data, call the `SIGDigest` function (page 6-44). To create additional digests, you can call the `SIGProcessData` and `SIGDigest` functions again, without first creating a new context or calling the `SIGDigestPrepare` function. When you no longer need the context you used for creating the digests, call the `SIGDisposeContext` function (page 6-29).

This section describes the `SIGDigestPrepare` function and the `SIGDigest` function.

SIGDigestPrepare

The `SIGDigestPrepare` function notifies the Digital Signature Manager that you are about to create a digest.

```
pascal OSErr SIGDigestPrepare (SIGContextPtr context);
```

`context` A pointer to the context that the Digital Signature Manager will use while creating the digest. Call the `SIGNewContext` function to obtain the context pointer.

DESCRIPTION

The `SIGDigestPrepare` function notifies the Digital Signature Manager that the context is to be used to create a digest.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$0772

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter value
memFullErr	–108	Not enough room in heap zone
kSIGInternalsErr	–1977	Bad digest, context, or signature
kSIGContextPrepareErr	–1979	Context already prepared by SIGVerifyPrepare, SIGSignPrepare, or SIGDigestPrepare

SEE ALSO

Before you call the SIGDigestPrepare function, you must call the SIGNewContext function (page 6-28) to create a new context.

After you call the SIGDigestPrepare function, call the SIGProcessData function (page 6-30) as many times as necessary to process all the data.

SIGDigest

The SIGDigest function returns a pointer to a digest of the data most recently processed by the SIGProcessData function.

```
pascal OSErr SIGDigest (SIGContextPtr context,
                        SIGDigestData digest);
```

context	The context pointer that you passed to the SIGDigestPrepare function.
digest	A SIGDigestData array that you provide to hold the result of this function.

DESCRIPTION

You can call the SIGProcessData function and the SIGDigest function as many times as you wish to prepare digests of data without calling the SIGDigestPrepare function again or creating a new context.

You must allocate a SIGDigestData structure to hold the digest before calling this function.

```
#define kSIGDigestSize 16

typedef Byte SIGDigestData[kSIGDigestSize], *SIGDigestDataPtr;
```

Note
You should call the SIGDisposeContext function as soon as possible after you finish making digests so that the Operating System can free the memory used by the Digital Signature Manager. u

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0773

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter value
`kSIGOperationIncompatibleErr`	–1970	Context in use for different type of operation
`kSIGInternalsErr`	–1977	Bad digest, context, or signature

SEE ALSO

You call the `SIGDigest` function calling `SIGDigestPrepare` (page 6-43) to initiate the digest process and `SIGProcessData` (page 6-30) to process the data.

As soon as possible after you finish preparing digests, call the `SIGDisposeContext` (page 6-29) function to dispose of the context and to allow the Operating System to remove the Digital Signature Manager from memory.

Getting Information From a Signature or Certificate

The first routine in this section, `SIGFileIsSigned`, indicates whether a file includes a standard signature. Use the other routines in this section to get information about the date, size, or contents of a full signature and its components.

SIGFileIsSigned

The `SIGFileIsSigned` function indicates whether a file contains a standard signature.

```
pascal OSErr SIGFileIsSigned(const FSSpec *fileSpec);
```

`fileSpec` A pointer to the file system specification structure for the file that you want to check for a signature.

DESCRIPTION

A file that has been signed by the finder or by the `SIGSignFile` function contains a digital signature in the form of a resource of type `'dsig'`. The `SIGFileIsSigned` function checks a file for this resource and returns a result code of `noErr` if it finds one. If the function finds no such resource, it returns the result code `kSIGNoSignature`.

Note
The 'dsig' resource is mentioned here for your information only. Because it may change in the future, you should not attempt to manipulate this resource directly. Any change could cause incompatibilities with future versions of the Digital Signature Manager. u

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0002	$09C4

RESULT CODES

noErr	0	File is signed
kSIGNoSignature	–1983	Standard file signature not found

SEE ALSO

You can call the SIGVerifyFile function (page 6-41) to verify a standard signature in a file.

You can add a standard signature to a file by calling the SIGSignFile function (page 6-36).

SIGShowSigner

The SIGShowSigner function displays the entire distinguished name of the signer of a block of data. You can call this function only after successfully verifying a signature.

```
pascal OSErr SIGShowSigner(SIGContextPtr context,
                           ConstStr255Param prompt);
```

context The context pointer you used the last time you called the SIGVerify or SIGVerifyFile function.

prompt The prompt you want to appear in the dialog box displayed by the SIGShowSigner function. If you specify a zero-length Pascal string for this parameter, the function displays a default string.

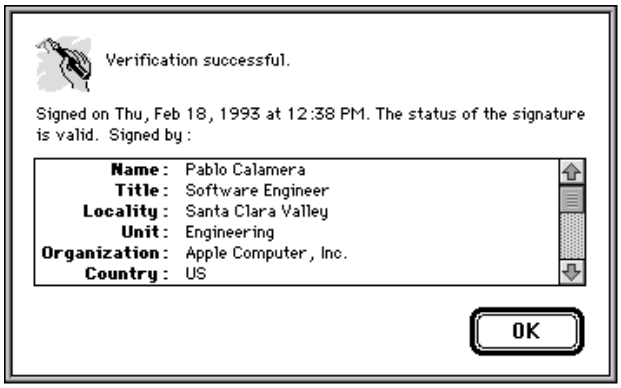
DESCRIPTION

After you call the `SIGVerify` function and it returns either the `noErr` or the `kSIGInvalidCredentialErr` result code, you can call the `SIGShowSigner` function to display a modal dialog box with the full distinguished name of the signer. Figure 6-6 shows an example of this dialog box.

Note

The time displayed is the local time determined by the user's local system clock and Map control panel. u

Figure 6-6 Show-signer dialog box



SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0775

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter value
`memFullErr`	–108	Not enough room in heap zone
`kSIGOperationIncompatibleErr`	–1970	Context in use for different type of operation
`kSIGSignerErr`	–1975	Problem with the signer file or signature
`kSIGInternalsErr`	–1977	Bad digest, context, or signature
`kSIGConversionErr`	–1981	Unable to convert to Macintosh format

SEE ALSO

You cannot call the `SIGShowSigner` function until after you have called the `SIGVerify` function (page 6-40) or the `SIGVerifyFile` function (page 6-41).

Distinguished names are defined in Table 6-1 on page 6-9.

SIGGetSignerInfo

The `SIGGetSignerInfo` function returns information about a signer.

```
pascal OSErr SIGGetSignerInfo (SIGContextPtr context,
                               SIGSignerInfo *signerInfo);
```

`context` The context pointer you used the last time you called the `SIGVerify`, `SIGVerifyFile`, `SIGSignPrepare`, or `SIGSign` function.

`signerInfo`
A pointer to a signer information structure returning information about the signer. You must allocate this structure.

DESCRIPTION

The `SIGGetSignerInfo` function returns information about the signer whose context pointer you provide to the function. You can call the `SIGGetSignerInfo` function after you call the `SIGSignPrepare` function, the `SIGSign` function, the `SIGVerify` function, or the `SIGVerifyFile` function.

You allocate a signer information structure, and the function fills it in. The signer information structure tells you the time (and date) that the data was signed, the number of certificates in the certificate set, and the status of the certificate set. (Note that if you call the `SIGGetSignerInfo` function immediately after calling the `SIGSignPrepare` function, the time of signing is meaningless because the data has not yet been signed.) If all the certificates are valid, the structure lists the earliest expiration date for any certificate in the set. If one or more certificates have expired, the structure lists the expiration date of the one that expired first. If none of the certificates have expired but one or more are not yet valid, the structure lists the date at which the last certificate to become valid will do so.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0004	$0776

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter value
kSIGCertificateQueryDenied	–1971	Can't query certificates with this context
kSIGSignerErr	–1975	Problem with the signer file or signature
kSIGInternalsErr	–1977	Bad digest, context, or signature

SEE ALSO

You can call the SIGGetSignerInfo function after you call the SIGSignPrepare function (page 6-31), the SIGSign function (page 6-34), the SIGVerify function (page 6-40), or the SIGVerifyFile function (page 6-41).

The signer information structure is described on page 6-23.

You can call the SIGShowSigner function (page 6-46) to display a modal dialog box showing the distinguished name of the signer of a verified signature.

You can call SIGGetCertInfo function (described next) to get more information about any certificate in the certificate set, including that of the signer.

SIGGetCertInfo

The SIGGetCertInfo function returns information about a specific certificate in a certificate set.

```
pascal OSErr SIGGetCertInfo (SIGContextPtr context,
                             unsigned long certIndex,
                             SIGCertInfo *certInfo);
```

context	The context pointer you used the last time you called the SIGVerify, SIGVerifyFile, SIGSignPrepare, or SIGSign function.
certIndex	The index number of the certificate about which you want information. The certificates are always in order: the signer's certicate has index number 0, the issuer of the signer's certificate has index number 1, and so forth. You can use the SIGGetSignerInfo function to determine the total number of certificates in the certificate set.
certInfo	A pointer to a certificate information structure returning information about the certificate. You must allocate this structure.

DESCRIPTION

The `SIGGetCertInfo` function returns information about one certificate in the certificate set of the signer whose context pointer you provide to the function. You allocate a certificate information structure and specify the index number of the certificate about which you want information, and the function fills in the structure. The certificate information structure tells you the beginning and ending dates for the validity period of the certificate, the status of the certificate (pending, expired, or valid), the number of attributes in the distinguished name of the signer of the certificate, the number of attributes in the distinguished name of the issuer of the certificate, and the serial number of the certificate.

The serial number and issuer name together uniquely identify a certificate. This information may be of use to a user who needs to contact the issuing organization (for example, to ensure a certificate has not been revoked).

The certificate of the signer of the data always has index number 0. You can use the following constant for this number:

```
#define kSIGSignerCertIndex 0
```

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0006	$0777

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Illegal parameter value
`kSIGCertificateQueryDenied`	–1971	Can’t query certificates with this context
`kSIGIndexErr`	–1974	Index value is outside allowable range
`kSIGInternalsErr`	–1977	Bad digest, context, or signature

SEE ALSO

You can call the `SIGGetCertInfo` function after you call the `SIGSignPrepare` function (page 6-31), the `SIGSign` function (page 6-34), the `SIGVerify` function (page 6-40), or the `SIGVerifyFile` function (page 6-41).

Call the `SIGGetSignerInfo` function (page 6-48) to determine the total number of certificates in the certificate set.

The certificate information structure is described on page 6-25.

You can use the SIGGetCertNameAttributes function (described next) to obtain the contents of each attribute in the distinguished name of the signer of the certificate.

You can use the SIGGetCertIssuerNameAttributes function (page 6-52) to obtain the contents of each attribute in the distinguished name of the issuer of the certificate.

The attributes that compose a distinguished name are shown in Table 6-1 on page 6-9.

SIGGetCertNameAttributes

The SIGGetCertNameAttributes function returns information about a specific attribute of a distinguished name in a specific certificate of a signature.

```
pascal OSErr SIGGetCertNameAttributes (SIGContextPtr context,
                            unsigned long certIndex,
                            unsigned long attributeIndex,
                            SIGNameAttributesInfo *attributeInfo);
```

`context` The context pointer you used the last time you called the SIGVerify, SIGVerifyFile, SIGSignPrepare, or SIGSign function.

`certIndex` The index number of the certificate about which you want information. The certificates are always in order: the signer's certicate has index number 0, the issuer of the signer's certificate has index number 1, and so forth. You can use the SIGGetSignerInfo function to determine the total number of certificates in the certificate set.

`attributeIndex`
The index number of the name attribute about which you want information. The SIGGetCertInfo function returns the total number of attributes in a certificate.

`attributeInfo`
A pointer to a SIGNameAttributesInfo structure.

DESCRIPTION

After you use the SIGGetCertInfo function to determine the total number of attributes in the distinguished name of a certificate, you can use the SIGGetCertNameAttributes function to obtain the attribute strings.

The SIGNameAttributesInfo structure returns information about the hierarchical level of the attribute, the type of name attribute, and the script code of the attribute, as well as returning the attribute string. You can use the hierarchy information to arrange the distinguished name for display to the user.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0008	$0778

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter value
kSIGCertificateQueryDenied	–1971	Can't query certificates with this context
kSIGIndexErr	–1974	Index value is outside allowable range
kSIGInternalsErr	–1977	Bad digest, context, or signature
kSIGConversionErr	–1981	Unable to convert an attribute to Macintosh format

SEE ALSO

You can call the `SIGGetNameAttributes` function after you call the `SIGSignPrepare` function (page 6-31), the `SIGSign` function (page 6-34), the `SIGVerify` function (page 6-40), or the `SIGVerifyFile` function (page 6-41).

Call the `SIGGetCertInfo` function (page 6-49) to determine the total number of attributes in the distinguished name. You can use the `SIGGetSignerInfo` function (page 6-48) to determine the total number of certificates in the certificate set.

The `SIGNameAttributesInfo` structure is described on page 6-26.

You can use the `SIGGetCertIssuerNameAttributes` function (described next) to obtain the contents of each attribute in the distinguished name of the issuer of the certificate.

Distinguished names and name hierarchies are described in detail in "About Public-Key Certificates" beginning on page 6-8.

SIGGetCertIssuerNameAttributes

The `SIGGetCertIssuerNameAttributes` function returns information about a specific attribute of the distinguished name of the issuer of a specific certificate of a signature.

```
pascal OSErr SIGGetCertIssuerNameAttributes
                            (SIGContextPtr context,
                            unsigned long certIndex,
                            unsigned long attributeIndex,
                            SIGNameAttributesInfo *attributeInfo);
```

`context` The context pointer you used the last time you called the `SIGVerify`, `SIGVerifyFile`, `SIGSignPrepare`, or `SIGSign` function.

`certIndex` The index number of the certificate about for whose issuer you want information. The certificates are always in order: the signer's certicate has index number 0, the issuer of the signer's certificate has index number 1, and so forth. You can use the `SIGGetSignerInfo` function to determine the total number of certificates in the certificate set.

`attributeIndex`
The index number of the name attribute about which you want information. The `SIGGetCertInfo` function returns the total number of attributes in the issuer of a certificate.

`attributeInfo`
A pointer to a `SIGNameAttributesInfo` structure.

DESCRIPTION

After you use the `SIGGetCertInfo` function to determine the total number of attributes in the distinguished name of the issuer of a certificate, you can use the `SIGGetCertIssuerNameAttributes` function to obtain the attribute strings.

The `SIGNameAttributesInfo` structure returns information about the hierarchical level of the attribute, the type of name attribute, and the script code of the attribute, as well as returning the attribute string. You can use the hierarchy information to arrange the distinguished name for display to the user.

This function is useful if you want information about the issuer of a certificate. If you are using the `SIGCertInfo` and `SIGCertNameAttributes` functions to obtain information about all the certificates in a certificate set, then you must use the `SIGGetCertIssuerNameAttributes` function to determine the prime issuer.

SPECIAL CONSIDERATIONS

This function may move or purge memory; you should not call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Parameter count	Routine selector
$0008	$0779

RESULT CODES

noErr	0	No error
paramErr	–50	Illegal parameter value
kSIGCertificateQueryDenied	–1971	Can't query certificates with this context
kSIGIndexErr	–1974	Index value is outside allowable range
kSIGInternalsErr	–1977	Bad digest, context, or signature
kSIGConversionErr	–1981	Unable to convert an attribute to Macintosh format

SEE ALSO

You can call the `SIGGetNameAttributes` function after you call the `SIGSignPrepare` function (page 6-31), the `SIGSign` function (page 6-34), the `SIGVerify` function (page 6-40), or the `SIGVerifyFile` function (page 6-41).

Call the `SIGGetCertInfo` function (page 6-49) to determine the total number of attributes in the distinguished name. You can use the `SIGGetSignerInfo` function (page 6-48) to determine the total number of certificates in the certificate set.

The `SIGNameAttributesInfo` structure is described on page 6-26.

You can use the `SIGGetCertNameAttributes` function (page 6-51) to obtain the contents of each attribute in the distinguished name of the signer of the certificate.

Distinguished names and name hierarchies are described in detail in "About Public-Key Certificates" beginning on page 6-8.

Application-Defined Function

The `SIGSign`, `SIGSignFile`, `SIGVerifyPrepare`, and `SIGVerifyFile` functions all take a `statusProc` parameter, which is a pointer to a callback routine. You may provide this routine to notify the user of the progress of the signing or verification process. Your routine may perform typical "busy-notification" actions, such as spinning the cursor, or it may offer the user the opportunity to cancel the operation.

MyStatusCallBack

Your status callback function can perform background tasks during the signing and verification processes.

```
pascal Boolean MyStatusCallBack (void);
```

DESCRIPTION

To provide a status callback function, pass a pointer (of type `SIGStatusProcPtr`) to your function in the `statusProc` parameter of the `SIGSign`, `SIGSignFile`, `SIGVerifyPrepare`, and `SIGVerifyFile` functions. If you return `false` as your function result, the Digital Signature Manager halts the signing or verifying operation.

This interface is available because the signing and verifying operations can take a relatively long time to complete. Your status callback function should provide some sort of feedback to the user, such as a spinning cursor or a dialog box, that indicates that the process is proceeding. This function should poll for Command-period keystrokes and return `false` if it detects one. Your status callback function can also perform any other background tasks you wish.

In addition to this routine, you may wish to have other progress-notification routines that are not callback routines. For example, you may wish to have a routine that keeps the user posted of progress between calls to `SIGProcessData`**.**

Note

It is impossible to determine ahead of time how many times your callback routine will be executed. u

SPECIAL CONSIDERATIONS

If you return `false` to halt the signing or verifying operation, the state of the context is undefined.

On entry, this routine restores the A5 register to the value it had when the routine was first called.

SEE ALSO

The `SIGSign` function is described on page 6-34.

The `SIGSignFile` function is described on page 6-36.

The `SIGVerifyPrepare` function is described on page 6-38.

The `SIGVerifyFile` function is described on page 6-41.

The `SIGProcessData` function is described on page 6-30.

Summary of the Digital Signature Manager

C Summary

Constants and Data Types

```
#define kSIGDigestSize           16

#define kSIGSignerCertIndex      0

#define kSIGSignatureIconResID          –16797

#define kSIGValidSignatureIconResID     –16799

#define kSIGInvalidSignatureIconResID   –16798

/* Name attribute types returned from SIGGetCertNameAttributes or
SIGGetCertIssuerNameAttributes */

typedef enum
{
   kSIGCountryCode,
   kSIGOrganization,
   kSIGStreetAddress,
   kSIGState,
   kSIGLocality,
   kSIGCommonName,
   kSIGTitle,
   kSIGOrganizationUnit,
   kSIGPostalCode
} ;

typedef unsigned short SIGNameAttributeType;

/* Signature status codes returned in SIGCertInfo or SIGSignerInfo */

typedef enum {
   kSIGValid,        /* all valid */
   kSIGPending,      /* none expired; some pending or unknown */
   kSIGExpired,      /* some expired, unknown, or pending */
```

```
   kSIGInvalid       /* some invalid, pending, expired, or unknown */
};

typedef unsigned short SIGCertStatus;
typedef unsigned short SIGSignatureStatus;

#define gestaltDigitalSignatureVersion 'dsig'

typedef Byte SIGDigestData[kSIGDigestSize], *SIGDigestDataPtr;

struct SIGSignerInfo
{
   unsigned long        signingTime;      /* time of signing */
   unsigned long        certCount;        /* number of certs in cert set */
   unsigned long        certSetStatusTime;/* expiration time */
   SIGSignatureStatus   signatureStatus;  /* certificate status */
};
typedef struct SIGSignerInfo SIGSignerInfo;
typedef SIGSignerInfo *SIGSignerInfoPtr;

struct SIGCertInfo
{
   unsigned long  startDate;           /* cert start validity date */
   unsigned long  endDate;             /* cert end validity date */
   SIGCertStatus  certStatus;          /* certificate status*/
   unsigned long  certAttributeCount;  /* number of name attributes in cert*/
   unsigned long  issuerAttributeCount;/* number of name attributes in
                                          cert's issuer */
   Str255         serialNumber;        /* cert serial number */
};
typedef struct SIGCertInfo SIGCertInfo;
typedef SIGCertInfo *SIGCertInfoPtr;

typedef Ptr SIGContextPtr;
typedef Ptr SIGSignaturePtr;

struct SIGNameAttributesInfo
{
   Boolean                 onNewLevel;
   SIGNameAttributeType    attributeType;
   ScriptCode              attributeScript;
   Str255                  attribute;
};
```

```
typedef struct SIGNameAttributesInfo SIGNameAttributesInfo;
typedef SIGNameAttributesInfo *SIGNameAttributesInfoPtr;
```

Digital Signature Manager Functions

Creating and Disposing of a Context

```
pascal OSErr SIGNewContext  (SIGContextPtr *context);
pascal OSErr SIGDisposeContext
                            (SIGContextPtr context));
```

Processing Data to Generate a Digest

```
pascal OSErr SIGProcessData
                            (SIGContextPtr context,
                             const void *data,
                             Size dataSize);
```

Creating a Signature

```
pascal OSErr SIGSignPrepare
                            (SIGContextPtr context,
                             const FSSpec *signerFile,
                             ConstStr255Param prompt,
                             Size *signatureSize);
pascal OSErr SIGSign        (SIGContextPtr context,
                             SIGSignaturePtr signature,
                             SIGStatusProcPtr statusProc);
pascal OSErr SIGSignFile    (SIGContextPtr context,
                             Size signatureSize,
                             const FSSpec *fileSpec,
                             SIGStatusProcPtr statusProc)
```

Verifying a Signature

```
pascal OSErr SIGVerifyPrepare
                            (SIGContextPtr context,
                             SIGSignaturePtr signature,
                             Size signatureSize,
                             SIGStatusProcPtr statusProc);
pascal OSErr SIGVerify      (SIGContextPtr context);
```

```
pascal OSErr SIGVerifyFile  (SIGContextPtr context,
                             const FSSpec *fileSpec,
                             SIGStatusProcPtr statusProc)
```

Creating a Digest

```
pascal OSErr SIGDigestPrepare
                            (SIGContextPtr context);
pascal OSErr SIGDigest      (SIGContextPtr context,
                             SIGDigestData digest);
```

Getting Information From a Signature or Certificate

```
pascal OSErr SIGFileIsSigned
                            (const FSSpec *fileSpec);
pascal OSErr SIGShowSigner  (SIGContextPtr context,
                             ConstStr255Param prompt);
pascal OSErr SIGGetSignerInfo
                            (SIGContextPtr context,
                             SIGSignerInfo *signerInfo);
pascal OSErr SIGGetCertInfo (SIGContextPtr context,
                             unsigned long certIndex,
                             SIGCertInfo *certInfo);
pascal OSErr SIGGetCertNameAttributes
                            (SIGContextPtr context,
                             unsigned long certIndex,
                             unsigned long attributeIndex,
                             SIGNameAttributesInfo *attributeInfo);
pascal OSErr SIGGetCertIssuerNameAttributes
                            (SIGContextPtr context,
                             unsigned long certIndex,
                             unsigned long attributeIndex,
                             SIGNameAttributesInfo *attributeInfo);
```

Application-Defined Function

```
pascal Boolean MyStatusCallBack
                            (void);
```

Pascal Summary

Constants and Data Types

```
CONST

{ Number of bytes needed for a digest record when using SIGDigest }

kSIGDigestSize                  = 16;

kSIGSignerCertIndex             = 0;

kSIGSignatureIconResID          = –16197

kSIGValidSignatureIconResID     = –16799

kSIGInvalidSignatureIconResID   = –16798

{ values of SIGNameAttributeType }

   kSIGCountryCode      = 0;
   kSIGOrganization     = 1;
   kSIGStreetAddress    = 2;
   kSIGState            = 3;
   kSIGLocality         = 4;
   kSIGCommonName       = 5;
   kSIGTitle            = 6;
   kSIGOrganizationUnit = 7;
   kSIGPostalCode       = 8;

{ values for SIGCertStatus or SIGSignatureStatus }

   kSIGValid      = 0;     { possible for either a SIGCertStatus or
                                 SIGSignatureStatus }
   kSIGPending    = 1;     { possible for either a SIGCertStatus or
                             SIGSignatureStatus }
   kSIGExpired    = 2;     { possible for either a SIGCertStatus or
                             SIGSignatureStatus }
   kSIGInvalid    = 3;     { possible only for a SIGSignatureStatus }

{ Gestalt selector code - returns toolbox version in low-order word }

   gestaltDigitalSignatureVersion = 'dsig';
```

```
TYPE

SIGNameAttributeType = INTEGER;

SIGCertStatus = INTEGER;

SIGSignatureStatus = INTEGER;

SIGDigestData = PACKED ARRAY[1..kSIGDigestSize] OF Byte;
SIGDigestDataPtr = ^SIGDigestData;

SIGSignerInfo = RECORD

   signingTime:      LONGINT;          { time of signing }
   certCount:        LONGINT;          { number of certificates in cert set }
   certSetStatusTime:LONGINT;          { expiration time }
   signatureStatus:  SIGSignatureStatus;{ status of the certificate }
END;

SIGSignerInfoPtr = ^SIGSignerInfo;

SIGCertInfo = RECORD
   startDate: LONGINT;           { cert start validity date }
   endDate: LONGINT;             { cert end validity date }
   certStatus: SIGCertStatus;    { signature status}
   certAttributeCount: LONGINT;  { number of name attributes in this cert }
   issuerAttributeCount: LONGINT;{ # of name attributes in certs issuer }
   serialNumber: Str255;         { cert serial number }
END;

SIGCertInfoPtr = ^SIGCertInfo;

SIGContextPtr     = Ptr;

SIGSignaturePtr   = Ptr;

SIGStatusProcPtr  = ProcPtr;  { FUNCTION SIGStatusProcPtr(): BOOLEAN;}

SIGNameAttributesInfo = RECORD
   onNewLevel: BOOLEAN;
   attributeType: SIGNameAttributeType;
   attributeScript: ScriptCode;
   attribute: Str255;
END;

SIGNameAttributesInfoPtr = ^SIGNameAttributesInfo;
```

Digital Signature Manager Functions

Creating and Disposing of a Context

```
FUNCTION SIGNewContext      (VAR context: SIGContextPtr): OSErr;
FUNCTION SIGDisposeContext  (context: SIGContextPtr): OSErr;
```

Processing Data to Generate a Digest

```
FUNCTION SIGProcessData     (context: SIGContextPtr; data: UNIV Ptr;
                             dataSize: Size): OSErr;
```

Creating a Signature

```
FUNCTION SIGSignPrepare     (context: SIGContextPtr; signerFile: FSSpecPtr;
                             prompt: StringPtr; VAR signatureSize: Size):
                             OSErr;
FUNCTION SIGSign            (context: SIGContextPtr; signature:
                             SIGSignaturePtr;statusProc: SIGStatusProcPtr):
                             OSErr;
FUNCTION SIGSignFile        (context: SIGContextPtr; signatureSize: Size;
                             fileSpec: FSSpec;statusProc:
                             SIGStatusProcPtr): OSErr;
```

Verifying a Signature

```
FUNCTION SIGVerifyPrepare   (context: SIGContextPtr; signature:
                             SIGSignaturePtr; signatureSize: Size;
                             statusProc: SIGStatusProcPtr): OSErr;
FUNCTION SIGVerify          (context: SIGContextPtr): OSErr;
FUNCTION SIGVerifyFile      (context: SIGContextPtr; fileSpec: FSSpec;
                             statusProc: SIGStatusProcPtr): OSErr;
```

Creating a Digest

```
FUNCTION SIGDigestPrepare   (context: SIGContextPtr): OSErr;
FUNCTION SIGDigest          (context: SIGContextPtr; digest:
                             SIGDigestData): OSErr;
```

Getting Information From a Signature or Certificate

```
FUNCTION SIGFileIsSigned    (fileSpec: FSSpec): OSErr;
FUNCTION SIGShowSigner      (context: SIGContextPtr; prompt: StringPtr):
                             OSErr;
FUNCTION SIGGetSignerInfo   (context: SIGContextPtr;
                             VAR signerInfo: SIGSignerInfo): OSErr;
```

```
FUNCTION SIGGetCertInfo     (context: SIGContextPtr; certIndex: LONGINT;
                             VAR certInfo: SIGCertInfo): OSErr;
FUNCTION SIGGetCertNameAttributes
                            (context: SIGContextPtr; certIndex: LONGINT;
                             attributeIndex: LONGINT; VAR attributeInfo:
                             SIGNameAttributesInfo): OSErr;
FUNCTION SIGGetCertIssuerNameAttributes
                            (context: SIGContextPtr; certIndex: LONGINT;
                             attributeIndex: LONGINT; VAR attributeInfo:
                             SIGNameAttributesInfo): OSErr;
```

Application-Defined Function

```
FUNCTION MyStatusCallBack (): BOOLEAN;
```

Assembly-Language Summary

Trap Macros Requiring Routine Selectors

$AA5D

Selector	Count	Routine
$076C	2	SIGNewContext
$076D	2	SIGDisposeContext
$076E	8	SIGSignPrepare
$076F	6	SIGSign
$0770	8	SIGVerifyPrepare
$0771	2	SIGVerify
$0772	2	SIGDigestPrepare
$0773	4	SigDigest
$0774	6	SIGProcessData
$0775	4	SIGShowSigner
$0776	4	SIGGetSignerInfo
$0777	6	SIGGetCertInfo
$0778	8	SIGGetCertNameAttributes
$0779	8	SIGGetCertIssuerNameAttributes
$09C4	2	SIGFileIsSigned
$09C5	8	SIGSignFile
$09C6	6	SIGVerifyFile

Result Codes

In addition to standard Macintosh Operating System errors such as `memFullErr` and `paramErr`, the Digital Signature Manager returns the result codes listed in this section.

Result codes in the range of –1970 to –1999 are reserved for the Digital Signature Manager.

`kSIGOperationIncompatibleErr`	–1970	Context in use for different type of operation
`kSIGCertificateQueryDenied`	–1971	Can't query certificates with this context
`kSIGVerifyFailedErr`	–1972	Verification failed
`kSIGInvalidCredentialErr`	–1973	Verified OK but credential either pending or expired
`kSIGIndexErr`	–1974	Index given is outside the range of allowable values
`kSIGSignerErr`	–1975	Problem with the signer file or signature
`kSIGPasswordErr`	–1976	Password is incorrect
`kSIGInternalsErr`	–1977	Bad digest, context, or signature
`kSIGToolboxNotPresentErr`	–1978	For servers; not returned by the toolbox
`kSIGContextPrepareErr`	–1979	Context either corrupted or already prepared with `SIGVerifyPrepare`, `SIGSignPrepare`, or `SIGDigestPrepare`
`kSIGNoDigestErr`	–1980	No digest in the signature
`kSIGConversionErr`	–1981	Unable to convert an attribute to Macintosh format
`kSIGSignerNotValidErr`	–1982	Signer file has either expired or is not yet valid
`kSIGNoSignature`	–1983	Standard file signature not found

CHAPTER 7

Interprogram Messaging Manager

Contents

This chapter describes the AOCE Interprogram Messaging (IPM) Manager. The IPM Manager provides a low-level interface to the AOCE store-and-forward messaging service.

You can use the IPM Manager to send a message from one AOCE-aware application to another. There are no restrictions on the contents of AOCE interprogram messages. However, if you want to send or read messages intended to be read by people, you should use the Standard Mail Package instead of the IPM Manager. Such messages are referred to as *letters*. The Standard Mail Package provides a high-level interface to the AOCE store-and-forward messaging service specifically to support letters. It is described in the chapter "Standard Mail Package" in this book.

This chapter assumes that you are familiar with AOCE catalog concepts, including catalog records, attribute types, and attribute values, as described in the chapter "Catalog Manager" in this book.

This chapter provides an introduction to AOCE interprogram messages and the IPM Manager and then discusses how you can use the IPM Manager to

- create and send a message to one or more recipients
- manage the queues in which the IPM Manager places messages
- list and read the messages that you receive

About the IPM Manager

The Apple Open Collaboration Environment provides a store-and-forward messaging service that can deliver a message from one application to another regardless of whether the applications are simultaneously connected to a network, or, in fact, regardless of whether they are connected to a network at all. In addition to general application-to-application messages, the Apple Open Collaboration Environment defines a special category of messages, called *letters*, that are intended to be read by people. The sending and receiving of letters by AOCE-aware applications is referred to as the *AOCE mail service*. The IPM Manager provides a low-level interface to AOCE messaging services. The Standard Mail Package is a client of the IPM Manager that provides a high-level interface to AOCE mail services.

The IPM Manager application interface is the same no matter what transport medium is being used to carry the message. Apple Computer, Inc., provides interfaces between the IPM Manager and an AppleTalk network with and without a mail and messaging server. Apple also provides the Direct Dialup mail and messaging service access module (MSAM), which allows the IPM Manager to use a modem to send messages over telephone lines. Other developers can provide MSAMs that allow the IPM Manager to use other transport media and messaging services, such as Ethernet networks or fax modems.

The IPM Manager maintains output and input queues on the local hard disk to store messages waiting to be forwarded or to be read. The IPM Manager can use the output queue, for example, to store a message until the telephone-connection MSAM can

establish a modem-to-modem connection. Any number of applications can use the same queue. You can ask for a list of messages filtered by creator, so you need not sort through all of the messages intended for other applications. However, if you have a need to do so, you can also create any number of input queues for the use of your application.

When you send a message, you must specify the addresses of one or more recipients. If a recipient or group of recipients has an associated record in an AOCE catalog, you can specify the record ID and the attribute containing the address, and the IPM Manager looks up the address in the catalog. Alternatively, you can specify the type of connection and provide specific information about the address of the recipient, such as the telephone number and modem information or the AppleTalk network address.

You can use the IPM Manager to

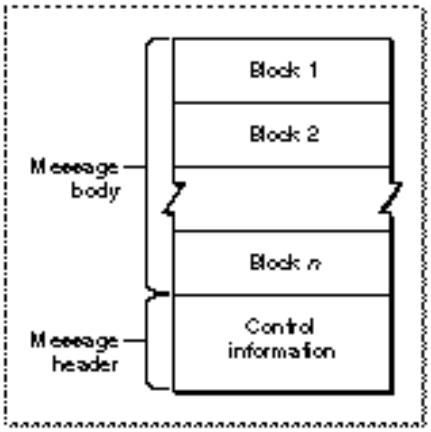
- create a new message
- add blocks to a message
- write data to a message block
- nest a message within a message
- address a message
- send a message or save it to a disk file
- create input queues
- open input queues
- obtain a list of received messages
- filter received-message lists by such attributes as priority, message type, or script code
- read message-header information
- read message blocks
- delete messages from an input queue
- close input queues

About AOCE Interprogram Messages

The AOCE store-and-forward messaging service implemented by the IPM Manager uses messages that consist of a header plus any number of message blocks. The header contains addressing information, a table of contents of the message blocks, other information of interest to the receiving application (such as the message type and priority), and information used solely by the IPM Manager. Each message block can be of any length less than 2^{32} bytes and can contain any type of data. Apple Computer has defined a few message types and message block types, such as the standard-letter-content block type used by the Standard Mail Package. You can define any message block types you wish.

Figure 7-1 illustrates the basic structure of a message. Note that the message header is actually located at the end of the message, after all the message blocks.

Figure 7-1 Structure of an AOCE message



When a block contains a message, the message inside the block is called a **nested message.** A message can contain any number of nested messages, and any nested message can contain other nested messages. The structure of a nested message is exactly the same as the structure of a message. Figure 7-2 illustrates a message containing a nested message.

Note
If you are using the IPM Manager to send letters to the Standard Mail Package, you should avoid sending any nested letters that contain standard content. If the Standard Mail Package receives a letter that contains a nested letter, it ignores any content (standard interchange format or image format) within the nested letter. It displays the header and nesting information of the nested letter as a forwarded mailer. u

Figure 7-2 An AOCE message containing a nested message

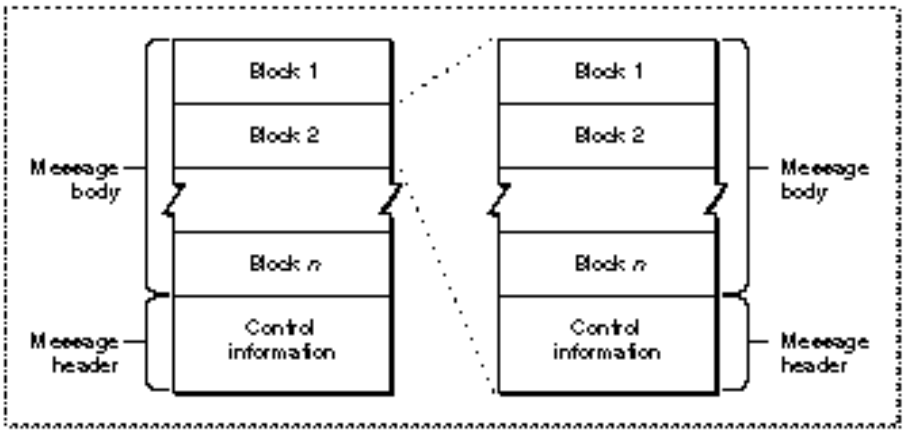
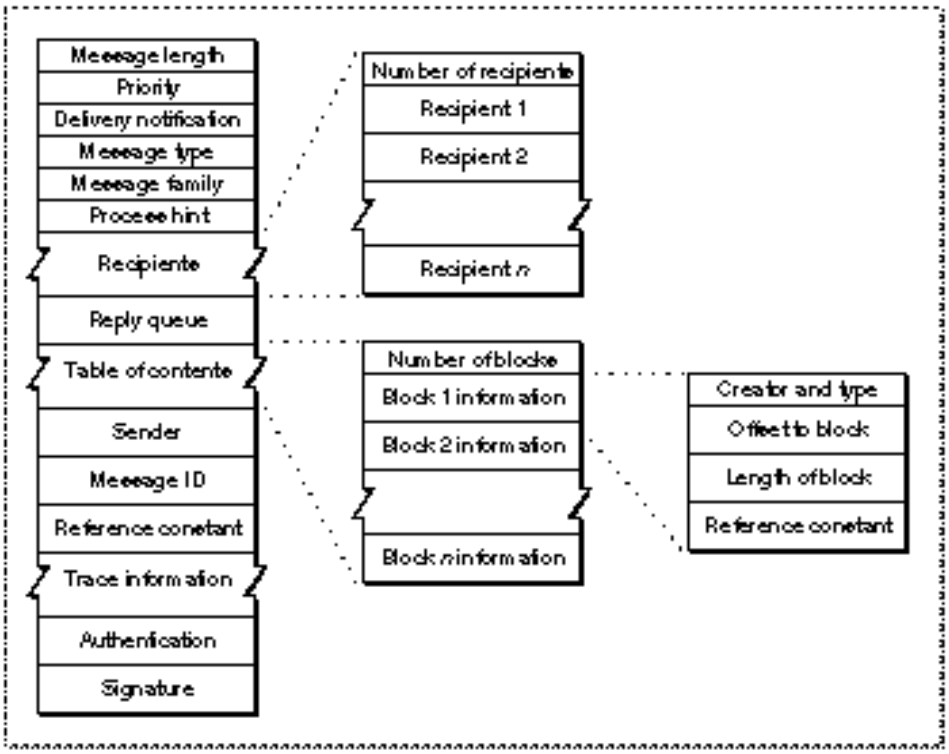


Figure 7-3 illustrates the contents of a message header. Note that Figure 7-3 does not show the size or true sequence of fields in the message header. You must use IPM Manager routines to read and write message-header information.

Figure 7-3 Contents of an AOCE message header



Although all of the public message-header fields are described in detail in the reference section of this chapter, several fields of general interest are briefly described here.

The sender of a message assigns a priority (low, normal, or high) to it. The IPM Manager does not read the contents of the priority field; it is up to the receiving application to determine how to handle messages of different priorities.

When you send a message, you can request delivery and nondelivery reports. The delivery notification field in the message header tells the IPM Manager what kinds of reports you want to receive. Reports are AOCE messages and can include the original message as a nested message if you request that option. Report messages are described in “Report Messages” on page 7-9.

The message type consists of a creator field and a type field. The sending application assigns the message type, and the receiving application uses it to help determine how to interpret the contents of the message. Apple Computer has defined some standard message types for report messages and letters. You can define other message types for whatever purpose you wish.

The message family is a class of messages. Apple Computer has defined some standard message family types for mail and reserves all message family types consisting entirely of lowercase characters. You can define your own message family types, but Apple Computer does not register or otherwise control developer-defined message family types.

The process hint is a character string that you can use for any purpose, such as discriminating among subtypes of messages of the same type or internal routing of messages.

When you send a message, you must specify location information for each recipient. You can specify the record ID of a user record if the recipient's address is stored in an AOCE catalog, or you can specify the actual delivery address of the recipient.

The reply queue is the address to which the IPM Manager should return delivery and nondelivery reports and to which reply messages should be sent.

The table of contents specifies the type and location of each block. The block type includes a creator field and a type field. Apple Computer has defined some standard block types for such things as nested messages and standard-letter-content blocks. You can define other block types for your own use.

In the case of an authenticated message, the sender field is filled in by the IPM Manager and identifies the authenticated originator of the message. In the case of an unauthenticated message, such as a message sent over a serverless network or over a dialup connection, the originator of the message fills in the sender field. In this case, the field should give some indication of who originated the message, but the IPM Manager can not ensure its accuracy or usefulness.

The reference constant is a numeric reference value that the creator of the message provides for the message. You might use this field, for example, to indicate that the message includes blocks of a certain type so that the receiving application can allocate the memory resources it will need to read the message. The table of contents (TOC information) for each block also contains a reference constant that you can use for any purpose you wish.

The IPM Manager sets the authentication information field to indicate whether the message was sent over a secure, authenticated connection. In the case of a message that passes through more than one store-and-forward server, the IPM Manager sets this field to `true` only if the identities of the original sender and of every server in the routing chain were authenticated. The authentication field does not reflect the authentication status of the communication link that the addressee uses to read the message from the last server's message queue. The chapter "Authentication Manager" in this book describes the authentication process in detail.

If the sending application adds a digital signature to a message, the IPM Manager adds a signature block to the message and sets the signature field of the message header to `true`.

Message Queues

The IPM Manager delivers a message to a **message queue,** which is maintained by the IPM Manager on the recipient's disk or by a server on the disk of the server computer. Any application can create message queues. Before you can list the messages in a message queue or read a message in a queue, you must open the queue.

Each queue can be opened any number of times, by any number of applications. Each time an application opens a queue the IPM Manager assigns a queue reference number. Each time you list the messages in the queue, open a message, read information from a message, close a message, or delete a message, you must specify a queue reference number.

When you list the messages in a queue, you can specify a filter that limits the messages included in the list. For example, you can filter a queue list for messages with a specific creator to limit it to messages sent by your own application. You can also filter queue lists by message priority or process hint (an application-defined value). When you open a queue (and so obtain a queue reference number), you can specify a default queue filter to be associated with that queue reference number. You can change the default queue filter at any time.

If you open a queue three times to get three queue reference numbers, it appears as though you have three queues, especially if you specify a different queue filter each time you open the queue. Note, however, that these three "queues" are all actually views of the same physical queue and so may list some or all of the same messages. To distinguish between the queue on disk and the apparent queues you get when you open the queue, this book refers to the **physical queue** on disk and to **virtual queues** associated with that physical queue. Each queue reference number identifies one virtual queue. A physical queue can have any number of associated virtual queues. When you close a virtual queue, the IPM Manager automatically closes all the messages that were opened through that virtual queue.

You can use a virtual queue to open and close messages regardless of whether the same messages are already open through another virtual queue. However, when you *delete* a message, it is deleted from the physical queue and so from all the virtual queues associated with that physical queue. (The IPM Manager prevents you from deleting a message as long as it is open through any virtual queue.)

The primary reason the IPM Manager provides virtual queues is to allow more than one application to use the same physical queue simultaneously. However, you can also use virtual queues to help organize your bookkeeping. You can use multiple virtual queues as a convenient way to group messages, especially if your message groups are based on message type or creator, script code, priority, or process hint.

For example, an application for stockbrokers might receive two types of IPM messages: notices about stock prices and orders sent by clients. Such an application might maintain two virtual queues to make it easier to list, open, and close the two message types independently.

In much the same way that virtual queues link together messages that you might want to list, open, or close together, each virtual queue is associated with a **queue context.** You

must open at least one queue context before you can open a queue, and each time you open a queue you must specify to which context the virtual queue is to belong. When you close a queue context, the IPM Manager automatically closes all of the queues associated with that context. If you are using several virtual queues to organize messages, you might want to use more than one queue context to add another hierarchical level to the organization.

To extend the previous illustration, for example, suppose the stockbrokers' application has separate virtual queues for low-, normal-, and high-priority buy-or-sell orders, and links these three queues together by assigning them all to the same context. Then the application could close all the high-priority orders by closing one virtual queue, or it could close all of the orders of all priorities by closing the queue context to which they belong.

Addresses

When you send an AOCE message, you must specify the address to which the message is to be delivered. The address can specify an entity (such as a person), an exact location (such as a queue on a specific AppleTalk node), or a group (which must be resolved into individual addresses).

An IPM message can contain two types of addresses: direct addresses and indirect addresses. A direct address specifies the exact location and queue name to which you want the message sent. An indirect address specifies the person or group to which you want the message sent and relies on IPM to determine the actual location and queue name of each addressee. AOCE addressing is described in two sections: "Direct Addressing," beginning on page 7-11, and "Indirect Addressing," beginning on page 7-14.

Report Messages

When you send a message, you can request that the IPM Manager return recipient report messages. You have several options for report messages. You can request that the IPM Manager

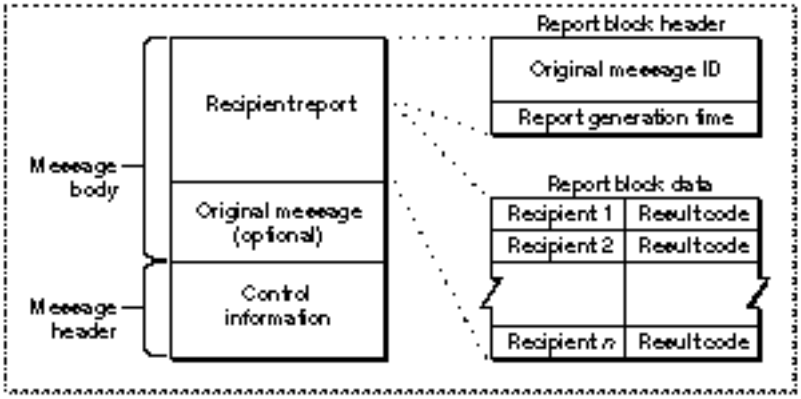
- return report messages when the message is delivered
- return report messages when the message cannot be delivered
- return both delivery and nondelivery reports
- include the original message in the report message
- include the original message only in nondelivery reports
- send a separate report message for each recipient, sending each one as soon as its delivery status is known for that recipient
- wait until the delivery status of the message is known for all recipients and then send a single summary report

Figure 7-4 illustrates the contents of a report message. Note that Figure 7-4 does not show the size or true sequence of fields in the report message. You must use IPM Manager functions to read report message information.

The report message contains a recipient report block, which includes a header and report data. The header, an `IPMReportBlockHeader` structure, includes the message ID of the original message and the time that the IPM Manager generated the report. The report data, an `OCERecipientReport` structure, indicates the outcome of the delivery to each recipient to which the report applies.

Because reports are messages, they are delivered to queues just as all messages are. Report messages are always delivered to the reply queue specified in the original message. If no reply queue was specified in the original message, then the IPM Manager does not issue report messages. When you send a message, you have the option of specifying whether you want the IPM Manager to issue delivery and nondelivery reports.

Figure 7-4 An IPM report message



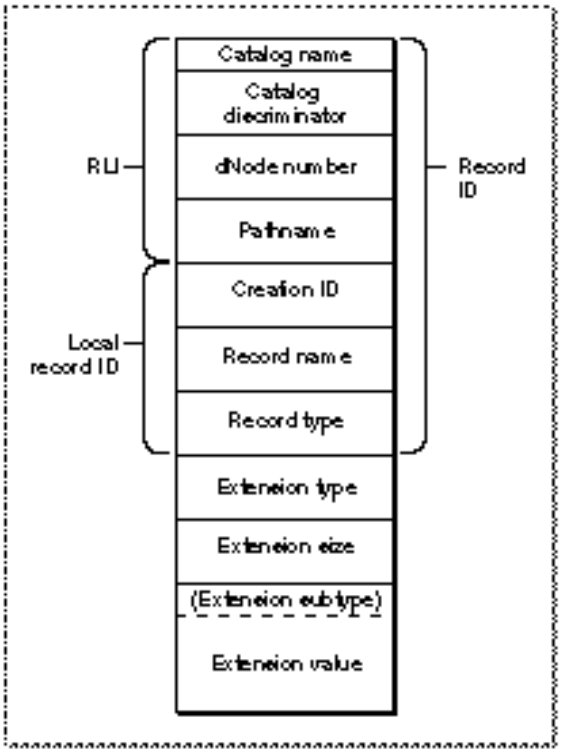
For more information on how to read a report, see the descriptions of the `IPMReportBlockHeader` structure on page 7-33 and the `OCERecipientReport` structure on page 7-33.

Addressing IPM Messages

The IPM Manager uses a single data type, the `OCERecipient` structure, to specify any type of address. Figure 7-5 shows the components of an `OCERecipient` structure. The `OCERecipient` structure has three parts: record location information (RLI), a local record identifier, and an extension. The record location information and local record identifier make up a record ID. Which of these parts are used in a specific address depends on the type of address, as described in the following sections. The `OCERecipient` structure is defined on page 7-24. For more information on the

RecordID, LocalRecordID, and RLI structures, see the chapter "AOCE Utilities" in this book.

Figure 7-5 Contents of an OCERecipient structure



Direct Addressing

In direct addressing, the OCERecipient structure specifies the location of the recipient and the queue to which you want the message sent. (All AOCE messages are delivered to a specific queue at a specific location.) This information is contained in the extension part of the OCERecipient structure.

Apple Computer, Inc., has defined address formats for its built-in transport media, which are described in the following sections. Personal and Server MSAMs allow the transport address space to be extended, and each transport medium has a unique set of addresses. Generally, the record location information (RLI) in the RecordID field is used for routing, the name and type are used for display, and the extension contains the native transport address as a displayable RString. The list of accessible RLIs is available via the DirGetExtendedDirectoriesInfo function, which is defined by the Catalog Manager.

The AOCE software defines two types of direct addresses: the AppleTalk type and the telephone type, described in the following two sections.

AppleTalk Direct Addressing

You can use AppleTalk direct addressing to specify the location on an AppleTalk internet to which a message should be delivered. You can use the Name Binding Protocol (NBP) AppleTalk routines to obtain the addresses of entities on an AppleTalk network. For more information on NBP and AppleTalk networking, see *Inside Macintosh: Networking.*

The IPM Manager recognizes an AppleTalk direct address by the value `'alan'` in the `extensionType` field of the `OCERecipient` structure. In this case, the extension portion of the `OCERecipient` structure contains the entire address; the IPM Manager ignores the record ID portion of the structure. The `extensionValue` field of the `OCERecipient` structure is defined as follows:

```
Str32    objectName
Str32    typeName
Str32    zoneName
Str32    queueName
```

All four of the fields are required, and all are packed. The first three fields are in the exact format used by the NBP `EntityName` structure. As is usual for AppleTalk, you can specify a zero-length string or the wildcard character * to indicate the local zone.

You must fill in the `queueName` field with the name of the specific queue to which the message is to be delivered. The messaging applications on both the sending and receiving computers have to open input queues and must somehow exchange queue names. You have to determine the protocol for achieving this yourself. The easiest way to know the recipient's queue name is for your application to use the same queue name always. If you need to send messages to multiple queues or have other reasons to allow more than one possible queue name, you have to implement your own process for determining which queues are available and what their names are.

Telephone Direct Addressing

You can use telephone direct addressing to specify an address for use by PowerTalk Direct Dialup. You must specify a telephone number and the recipient queue name. The IPM Manager delivers the message to the queue with the specified name on the node that is connected to a modem at the specified telephone number. To receive and route the message correctly, the receiving computer must have Direct Dialup installed and the modem set to answer the telephone.

Note

The telephone direct addressing type of `OCERecipient` structure described here is created by the Direct Dialup template when the user adds a Direct Dialup mail address to an information card. You can use the information in this section to create your own Direct Dialup addresses to use with messaging applications. The IPM Manager provides no facilities for using communications software other than Direct Dialup to send messages over telephone lines. u

The IPM Manager recognizes a telephone direct address by the value `'aphn'` in the `extensionType` field of the `OCERecipient` structure.

You should use the string “Direct Dialup” for the catalog name field of the `OCERecipient` structure. This is the name of the personal catalog used by the Direct Dialup software for setup information. The Direct Dialup catalog contains access numbers for local calls (such as 9, used to obtain an outside line in some telephone systems), long distance calls (such as 8 to obtain a long-distance outside line), and international calls (when calling from the United States, this is generally 011, the international access code) and can specify a credit card number to be used. The IPM Manager ignores the other fields in the record ID portion of the `OCERecipient` structure.

When you use telephone direct addressing, the `extensionType` field must contain the value `'aphn'` and the `extensionValue` field is defined as follows:

```
RString     phoneNumber /* telephone number */
RString     modemType   /* reserved */
Str32       queueName   /* recipient's queue name */
```

All three fields are required. The `phoneNumber` and `modemType` fields must be padded to an even number of bytes, and all fields must be packed.

The `phoneNumber` field is composed of several subfields. Each subfield must be packed and padded to an even number of bytes.

```
short    subType;
RString  countryCode;
RString  areaCode;
RString  phone;
RString  postFix;
RString  nonHandyDialString;
```

Field descriptions

`subType` A byte that specifies whether the Direct Dialup software should use the information in the Direct Dialup setup catalog when it forms the dialing string. If you specify the value `kOCEUseHandyDial` for this field, the Direct Dialup software uses the Direct Dialup catalog to obtain special access numbers and optionally a charge card number. If you specify `kOCEDontUseHandyDial` for this field, the Direct Dialup software uses only the exact dialing string you specify in the `nonHandyDialString` field and ignores the other subfields.

`countryCode` The ASCII value of the country code needed to dial an international number. For example, the country code for the United Kingdom is ASCII 44. For long-distance calls from and within North America, use the long distance prefix, ASCII 1.

`areaCode` The ASCII value of the US area code or, for international calls, the city code.

`phone` The telephone number including any other special modem control characters you may need. For example, you could include the "," character as one of the characters in the `phone` string to cause the modem to pause briefly before dialing the rest of a number.

`postFix` Reserved. You must specify an `RString` structure of zero length and an empty data string (`""`).

`nonHandyDialString` The dialing string used by the Direct Dialup software when you set the `subType` field to `kOCEDontUseHandyDial`. When this is the case all of the other fields of the extension value are ignored when the dialing string is formed. If the `subType` field has a value of `kOCEUseHandyDial`, then Direct Dialup ignores this field.

The `modemType` field of the extension value is reserved and must be set to an empty `RString`; that is, an `RString` structure with a length of `0` and an empty data string (`""`).

You must fill in the `queueName` field of the extension value with the name of the specific queue to which the message is to be delivered. The messaging applications on both the sending and receiving computers have to open input queues and must somehow exchange queue names. You have to determine the protocol for achieving this yourself. The easiest way to know the recipient's queue name is for your application to always use the same queue name. If you need to send messages to multiple queues or have other reasons to allow more than one possible queue name, you have to implement your own process for determining which queues are available and what their names are.

Indirect Addressing

You can use indirect addressing when you want to specify the entity to which a message should go, instead of the exact location and queue name to which the message should be delivered. In indirect addressing you specify a record—and optionally a specific attribute within the record—that contains the location and queue information that the IPM Manager needs to deliver the message. In mail applications, for example, the user typically selects a user record from a catalog or information card as the addressee. The IPM Manager then looks up the address of the recipient in that user record.

To use indirect addressing, fill in the record ID portion of the `OCERecipient` structure with the record ID of the record containing the address and set the `extensionType` field to the value '`entn`'. Extensions of type '`entn`' include a subtype field, which can have the following values:

```
enum {
   kOCEAddrXtn= 'addr', /* reserved */
   kOCEQnamXtn= 'qnam', /* queue-name form */
   kOCEAttrXtn= 'attr', /* attribute-type form */
   kOCESpAtXtn= 'spat'  /* reserved */
};
```

To specify an indirect address, you must use the attribute-type (`'attr'`) subtype. The queue-name subtype of an `OCERecipient` structure is used for attribute values (see “Queue-Name Format for Attribute Values” on page 7-16). The other two subtypes are reserved for use by the IPM Manager.

Both the attribute-type and queue-name subtypes require the record ID portion of the `OCERecipient` structure to contain a valid reference to a record.

The fields that are required in the record ID portion of the `OCERecipient` structure are as follows:

- If the creation ID value is sufficient to identify the record in the catalog then the `recordName` and `recordType` fields are not required and can be `nil`.
- If the creation ID is not sufficient to specify the record or is `null`, then the `recordName` and `recordType` fields are required.
- If you include both the creation ID and the record name and type, they must specify the same record.

You can use the Catalog Manager functions to create and modify records and record attribute values. See the chapter “Catalog Manager” in this book for more information. For information on record IDs, attributes, attribute values, and the creation ID, see the chapter “AOCE Utilities” in this book.

Attribute-Type Indirect Addressing

You use attribute-type indirect addressing when you want to specify the entity that is to receive a message rather than the specific location and queue to which a message is to be delivered. The IPM Manager obtains the location and queue name to which the message is to be delivered from an attribute in the record you specify. If you are specifying a standard AOCE user record or group record into which the system administrator placed messaging addresses, then the IPM Manager creates the attribute containing the address, and you do not have to be concerned with the format of the attribute value. If, however, you want to create your own record or attribute type and place addresses in it yourself, then you need to be familiar with address formats for attributes, discussed in the following section, “Queue-Name Format for Attribute Values.”

The simplest form of an attribute-type `OCERecipient` structure has an extension type of `'entn'`, an extension size of 0, and no extension value. In this case, the IPM Manager uses the preferred messaging queue as specified in the default messaging attribute in the record. The preferred messaging queue is created and designated by the catalog administrator.

To specify an attribute type, use an extension type of `'entn'` and a subtype of `'attr'`. The extension value is defined as follows:

```
OSType         'attr'
AttributeType  attributeName
```

The `AttributeType` structure is defined as follows:

```
struct AttributeType {
   RStringHeader
   Byte body[kAttributeTypeMaxBytes];
};
```

The `attributeName` field must be packed and padded to an even number of bytes. The `AttributeType` structure is equivalent to an `RString` structure that has a length of `kAttributeTypeMaxBytes` bytes. For more information on the `AttributeType` and `RString` structures, see the chapter "AOCE Utilities" in this book.

Setting the subtype to `'attr'` and the `body` field of the `AttributeType` structure to the value `kPrefMsgQAttrTypeBody` has the same effect as leaving out the extension value entirely: the IPM Manager uses the preferred messaging queue in the record as the address to which to deliver the message.

If you specify another attribute type, then the IPM Manager looks for the address in that attribute type. If there is more than one attribute value in the record with the attribute type you specify, the IPM Manager chooses one of the values. The method that the IPM Manager uses to decide which attribute value to use is private. Therefore, you should use a multivalued attribute type to hold an indirect address only when you do not care at which address a recipient receives the message.

Queue-Name Format for Attribute Values

If you want to define your own record type or attribute type to hold addresses for indirect addressing, you must format the attribute value as an `OCERecipient` structure. You use the queue name form of the `OCERecipient` structure for the attribute value. The recipient must have an account on an AOCE messaging server, such as a PowerShare server. The queue name form specifies the messaging server and queue name to which to deliver the message.

In the queue name form of the `OCERecipient` structure, the `extensionType` field has a value of '`entn`', the extension subtype field has a value of '`qnam`', and the extension data is a queue name string. The `extensionValue` portion of the `OCERecipient` structure is defined as follows:

```
OSType          'qnam'
Str32           queueName
```

The record ID portion of the `OCERecipient` structure specifies the catalog and record ID of the catalog record that contains information about the messaging server. (When the system administrator installs a messaging server, the setup software creates a catalog record containing information about the messaging server.)

As with other AOCE addressing formats that require the name of a queue, you must implement your own method for obtaining the queue name because the AOCE toolbox does not provide you with a mechanism for doing so.

Here is one possible procedure for indirect addressing using queue name attribute values:

1. Create your own new record type, or create a new attribute for an existing record type.
2. Log on to the messaging server as an administrator and create a queue with the name you want to use. You use the `IPMCreateQueue` function (page 7-69) for this purpose.
3. Put the name and location of the queue you just created into the new attribute in a queue-name-format `OCERecipient` structure.
4. Once you have created the queue and you have placed the queue name and location information into an attribute, then both ends of your connection can obtain the queue name from the record. Both the recipient and the sender of the message must know before the message is sent which record and attribute in the catalog contains the queue name.

Using the IPM Manager

This section describes how to create messages, create and manage message queues, and read messages.

Determining Whether the Collaboration Toolbox is Available

Before calling any of the Interprogram Messaging Manager functions, you 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" of *Inside Macintosh: Operating System Utilities*.

If you want to be informed when the Interprogram Messaging Manager starts up or shuts down, you can install an entry in the AppleTalk Transition Queue (ATQ). Then the AppleTalk LAP 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 of *Inside Macintosh: Networking*.

Determining the Version of the Collaboration Toolbox

To determine the version of the Collaboration Toolbox 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 you are using the Collaboration toolbox on a computer that has a PowerShare server, the function returns

the version number of the server in the high-order word of the `response` parameter. If the Collaboration Toolbox or server is not present and available, the `Gestalt` function returns 0 for the relevant version number. You can use the constant `gestaltOCETB` for AOCE Collaboration Toolbox version 1.0.

Note that the version number of the Collaboration toolbox is not necessarily the same as that returned by the `IPMReadHeader` function (page 7-89) for the IPM Manager. The `IPMReadHeader` function returns a version number in the `version` field of the `IPMFixedHdrInfo` structure (page 7-38).

Error Handling

If the ASDSP connection between the Collaboration toolbox and the server shuts down for any reason, the next IPM Manager function you call that requires communications with the server fails with the result code `kOCEConnectionClosed`. To correct this condition, call the `IPMCloseQueue` function (page 7-76) to close the messaging queue and then call the `IPMOpenQueue` function (page 7-72) to reopen the queue.

If either end of the IPM connection crashes during message transmission, the IPM Manager might send a duplicate copy of a message that was already successfully delivered. Although such an occurrence is very rare, your application should be capable of handling the receipt of duplicate messages.

Creating a Message

A message is created in three steps:

1. Initiate the message-creation process.
2. Add information to the message.
3. End the process.

Initiating the Message-Creation Process

Before you start to create a message, you must decide whether you intend to send the message, save it to disk, or nest it in another message. These processes are independent of one another. If you want to both send a message and save the same message to disk, for example, you must create the message twice.

- To begin the process of creating a new message to be sent to a recipient, call the `IPMNewMsg` function (page 7-43).
- To start a new message to be saved to disk, call the `IPMNewHFSMsg` function (page 7-47).
- To start a new nested message, call the `IPMNewNestedMsgBlock` function (page 7-56).

You provide each of these functions with information for the message header and an authentication identity of the creator of the message. You can specify the reply message queue and one recipient message queue at this time, or you can add this address

information later, as described in the following section. Each of the new-message functions returns a message reference number that you must use when you call other functions to build the message.

Adding Information to the Message

Once you have started the message, you can add information to the message. You can call the `IPMAddRecipient` (page 7-50) and `IPMAddReplyQueue` (page 7-52) functions at any time during the message-creation process to add recipients and a reply queue to the message header. To add a new message block, you first call either the `IPMNewBlock` function (page 7-53) to start a new message block, or the `IPMNewNestedMsgBlock` function to start a new nested-message block. You then call the `IPMWriteMsg` function (page 7-61) to add data to a message block. You can also use the `IPMNestMsg` function (page 7-59) to add an existing message as a message block. You can't modify such a nested message. You can add as many message blocks and nested messages as you wish to a message.

Note

Although the IPM Manager allows you to add any number of nested-message blocks at the same nesting level in a message, the messaging service access module (MSAM) interface supports only one nested-message block at a given nesting level. Therefore, if you want your message to be compatible with MSAMs, you must not add more than one nested-message block at a given level of nesting. You can, however, nest a message within another nested message to as many nesting levels as disk and memory resources allow. u

The `IPMWriteMsg` function adds data at a specific offset in a message. You can specify an offset from the start of the currently open message block, from the start of the message, or from the end of the last byte written. A message block can be any length. Each time you call the `IPMNewBlock` function or the `IPMNewNestedMsgBlock` function, the IPM Manager closes the current message block and starts a new message block, putting the offset to the beginning of the new block into the message header. Therefore, once you start a new message block, you cannot extend the length of any message blocks you added earlier. You can write over the data in a block you wrote earlier, but you can't extend the block.

If you call the `IPMNewNestedMsgBlock` function to add a nested-message block to a message, each subsequent call to the `IPMNewBlock` or `IPMNewNestedMsgBlock` functions adds another block to the nested message, not a new block to the enclosing message. Once you have started a nested message, you must call the `IPMEndMsg` function (page 7-65) to complete the nested message before you can add any more information to the enclosing message. After you call the `IPMEndMsg` function to end the nested message, you cannot add any recipients or blocks to the nested message.

Ending a Message

When you are finished adding address information, blocks, and nested messages to your message, you call the `IPMEndMsg` function. This function sends the message, saves it to disk, or ends a nested message, depending on which function you used to start the message. You can also choose to add a digital signature to the message at this time and you can request delivery and nondelivery reports.

Creating and Managing Message Queues

The IPM Manager provides functions to perform the following tasks:

- create a new physical queue
- open a queue context
- open a physical queue to establish a virtual queue
- change the default message filter for a virtual queue
- enumerate the messages in a queue
- close a queue context
- close a virtual queue
- delete a physical queue

Creating and Opening a Queue

Before another client of IPM can send messages to your application or process, you must establish the input messaging queue to which the messages will be sent and from which you can read them. You can use the default messaging queue created by the PowerShare system administrator for the user as described in “Attribute-Type Indirect Addressing” on page 7-15.

The administrator of a PowerShare messaging server can create any number of queues on the server computer. Each such queue has a creator (the administrator who created the queue) and an owner, assigned by the administrator. The owner can open a queue and the administrator can delete a queue. An administrator typically creates a queue for each user who has an account on the server.

However, if you want to create and maintain your own messaging queues, you must use the functions described in “Managing Message Queues” on page 7-68.

To establish a messaging queue, follow these steps:

1. Call the `IPMCreateQueue` function to create a new physical queue. When you call the `IPMCreateQueue` function (page 7-69), the IPM Manager sets up a new physical input queue with the name and address you specify. Other users of IPM can send messages to that queue (assuming they know its name and address) at any time.

Note

You must be authenticated as the administrator to add a queue to a PowerShare server. u

2. Call the `IPMOpenContext` function to create a new queue context. A queue context links together virtual queues so that, by closing the context, you can simultaneously close all of the queues associated with that context. You use the `IPMOpenContext` function (page 7-70) to create a new context and the `IPMCloseContext` function (page 7-77) to close one. The `IPMOpenContext` function returns a context reference number that you use when you call the `IPMOpenQueue` function to open a new virtual queue.
3. Call the `IPMOpenQueue` function to establish a new virtual queue. Whereas the `IPMCreateQueue` function creates a physical message queue, the `IPMOpenQueue` function (page 7-72) opens the physical queue to establish a virtual queue (see "Message Queues" on page 7-8 for a discussion of physical and virtual message queues). You cannot read messages from a queue until you open it. When you call the `IPMOpenQueue` function, you must specify the queue context to which the new virtual queue will belong. You can call the `IPMOpenQueue` function any number of times to establish distinct virtual queues associated with the same physical input queue. Each time you call this function, the IPM Manager returns a unique queue reference number.

Specifying a Queue Filter and Enumerating a Queue

When you call the `IPMOpenQueue` function to establish a virtual queue, you can specify a default message filter for that virtual queue. You can filter messages by priority, message type, or other attributes, as described in "Filter Structures" on page 7-34.

For example, you can open an input queue three times to create three virtual queues, each with its own filter: one that passes only high-priority messages, one that passes only messages specifically intended for your application, and one that passes all messages in the physical input queue. You can use the `IPMChangeQueueFilter` function (page 7-74) to change the default message filter for a specific virtual queue.

When you call the `IPMEnumerateQueue` function (page 7-80), you specify a queue reference number and you can specify a queue filter. The IPM Manager uses the message filter to determine which messages in the physical queue to list. If you do not provide a message filter with the `IPMEnumerateQueue` function, the function uses the default filter for that virtual queue.

Closing a Queue

You can close an individual virtual queue or you can close a queue context to simultaneously close all of the virtual queues associated with that context. When you open a message, you specify the reference number for an open virtual queue. This virtual queue must belong to the physical queue that actually contains the message and its filter must pass the specific message you wish to open. When you call the `IPMCloseQueue` function (page 7-76) to close a virtual queue, the IPM Manager closes all of the messages opened using that virtual queue's reference number and removes the virtual queue from its context. When you call the `IPMCloseContext` function (page 7-77) to close a context, the IPM Manager closes all of the messages opened for all the virtual queues associated with that context before it closes the virtual queues and removes the context.

Call the `IPMDeleteQueue` function (page 7-78) to delete a physical queue that you own. Before you delete a physical queue, you must close all of the virtual queues that belong to that physical queue.

Reading Messages

To read a message, follow these steps:

1. Enumerate the queue or determine the location of the message on disk. Use the `IPMEnumerateQueue` function (page 7-80) to list the messages in a virtual queue; that is, the messages that meet the filter criteria for the queue. If you wish, you can specify a filter that is in effect only for a single execution of the function; otherwise, the function uses the current filter for the virtual queue. In addition to the sequence number of each message, the `IPMEnumerateQueue` function provides information about the message such as the message length and priority.

   A queue can contain any number of messages. The IPM Manager assigns a sequence number to each message when it adds the message to the physical queue. The IPM Manager uses a monotonically increasing series of sequence numbers and does not reuse a sequence number when a message is deleted from the queue. Therefore, when you request a list of all the messages in the queue, some sequence numbers might be missing, but the message with the highest sequence number is always the last one added to the queue.

   Use File Manager or Standard File Package routines to locate a message on disk. The File Manager and Standard File Package are described in *Inside Macintosh: Files*.

2. Open the message. Use the `IPMOpenMsg` function (page 7-82) to open a message in an input queue or the `IPMOpenHFSMsg` function (page 7-84) to open a message that has been saved in a file on disk. These functions return a message reference number that you must provide to the various message-reading functions.

   If a message contains a nested-message block, you can use the `IPMOpenBlockAsMsg` function (page 7-86) to open that block as a message. You must open the containing message and determine the offsets of the nested-message block before you can open a nested message. You use the `IPMGetBlkIndex` function (page 7-96) to get the index numbers and block types of the blocks in a message.

3. Read the message header. The IPM Manager reads certain fields of the headers of messages in an input queue and saves this information in local memory. You can use the `IPMGetMsgInfo` function (page 7-87) to read this information. The `IPMGetMsgInfo` function returns the same information about a message as that returned by the `IPMEnumerateQueue` function. To get more information about a message or to read header information from a message on disk or a nested message, use the `IPMReadHeader` function (page 7-89).

   The creator of a message adds one or more recipients to the message header. Some or all of these recipients might be group addresses or references to catalog records that the IPM Manager must resolve before delivering the message. The `IPMReadRecipient` function (page 7-92) returns only the original list of recipients.

4. Call the `IPMGetBlkIndex` function (page 7-96) to get the index numbers and block types of the blocks in the message. If you are interested only in blocks of a certain type, such as nested-message blocks, you can use this function to list only those blocks.

5. Use the `IPMReadMsg` function (page 7-98) to read any message block other than a nested-message block.

   Call the `IPMOpenBlockAsMsg` function to open a nested-message block as a message and then use the other functions in this section to read it as you would read any other message. Before you use this function, you must open the containing message (which can also be a nested message) and you must know the index number of the nested-message block within the containing message. A nested message has a creator type of `kIPMSignature` and a block type of `kIPMEnclosedMsgType`.

   If the message includes a digital-signature block, you can use the `IPMVerifySignature` function (page 7-102) to verify the signature.

6. When you have finished reading the message, call the `IPMCloseMsg` function (page 7-104) to close the message and release the memory the IPM Manager reserved for the message when you opened it. Closing a message does not automatically close any nested messages that you have opened with the `IPMOpenBlockAsMsg` function; you must call the `IPMCloseMsg` function once for every nested message you open. You can also close messages by closing the message queue or the queue context to which that message belongs.

IPM Manager Reference

This section describes the data types and routines provided by the IPM Manager.

Data Types

The IPM Manager routines use the data types described in this section. Included are structures for message addressing, message and block types, delivery notification, filter structures, message information structures, header information structures, sender structures, and interprogram messaging parameter blocks.

Message Addressing Structures

You must use the `OCERecipient` structure to specify a message address. This section also shows some structures you can use for extensions to `OCERecipient` structures. See "Addressing IPM Messages," beginning on page 7-10 for more information about addressing.

OCERecipient

The `OCERecipient` structure is defined as a `DSSpec` data type.

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

typedef struct DSSpec DSSpec;
typedef DSSpec OCERecipient;
```

The `OCERecipient` structure can specify a specific attribute in a specific record in a catalog from which the IPM Manager reads the recipient address, or it can hold the actual queue address. The various forms of the `OCERecipient` structure are described in "Addressing IPM Messages," beginning on page 7-10.

All of the components of the DSSpec data type are defined in the chapter "OCE Utilities" in this book. Figure 7-5 on page 7-11 illustrates the contents of an OCERecipient structure. Note that this figure does not show the true size or location of the fields in an OCERecipient structure, and that the actual structure contains packed fields. You must use the utility routines provided by the IPM Manager to create and read these structures. The utility routines are described in "Utility Functions," beginning on page 7-107.

OCEPackedRecipient

The IPM Manager often uses a packed form of the OCERecipient structure, defined by the OCEPackedRecipient data type.

```
# define OCEPackedRecipientHeader\
   unsigned short    dataLength;

struct ProtoOCEPackedRecipient{
   OCEPackedRecipientHeader;
};

typedef struct ProtoOCEPackedRecipient ProtoOCEPackedRecipient;

# define kOCEPackedRecipientMAXBYTES\
         (4096 - sizeof(ProtoOCEPackedRecipient))

struct OCEPackedRecipient {
   OCEPackedRecipientHeader
   Byte              data[kOCEPackedRecipientMaxBytes];
};

typedef struct OCEPackedRecipient OCEPackedRecipient;
```

The dataLength field at the beginning of the structure specifies the length of the data field that follows. The data field of the OCEPackedRecipient structure contains an OCERecipient structure in packed format. Use the utility routines provided by the IPM Manager to pack and unpack OCERecipient structures.

IPMEntityNameExtension

You can use the following data type when creating an extension to an `OCERecipient` structure:

```
struct IPMEntityNameExtension {
   OSType subExtensionType;
   union {
      IPMEntnSpecificAttributeExtension specificAttribute;
      IPMEntnAttributeExtension        attribute;
      IPMEntnQueueExtension            queue;
   } u;
};
```

The specific attribute type is reserved for use by the IPM Manager.

IPMEntnAttributeExtension

The attribute type is defined by the `IPMEntnAttributeExtension` structure.

```
struct IPMEntnAttributeExtension {     /* kOCEAttrXtn */
   AttributeType attributeName;
};
```

IPMEntnQueueExtension

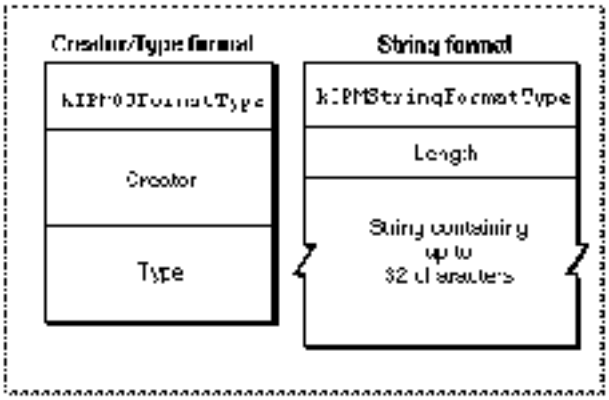
The queue type is defined by the `IPMEntnQueueExtension` data structure.

```
struct IPMEntnQueueExtension {
   Str32 queueName;
};
```

Message and Block Types

Each IPM message has an associated message type. Each block in a message has a block type. A message type can have either of two formats: the creator / type format contains a creator field and a type field; the string format contains a length field and a string. A block type always has the creator / type format. As illustrated in Figure 7-6, the first field in a message type structure is a 2-byte tag that specifies the format of the structure. The block type structure does not include this tag.

Figure 7-6 The two forms of the message type structure



IMPORTANT

Apple Computer, Inc., reserves all message type values and all block type values that consist entirely of lowercase letters. s

OCECreatorType

The block type and the creator/type portion of a message type are defined by the `OCECreatorType` data type.

```
struct OCECreatorType {
   OSType   msgCreator;
   OSType   msgType;
};
```

Field descriptions

`msgCreator` The creator of the message or block. You can specify any four-character value in this field; usually it is the signature of your application. For example, a message or block created by the IPM Manager has a creator type of `kIPMSignature`.

`msgType` The type of the message or block. For example, an enclosed message block has a block type of `kIPMEnclosedMsgType`. You can define your own four-character block types to serve your own purposes. Apple Computer, Inc., reserves all block types consisting entirely of lowercase letters.

IPMMsgType

The message type structure is defined by the `IPMMsgType` data type.

```
/* values of IPMMsgFormat */
enum {
   kIPMOSFormatType = 1,
   kIPMStringFormatType = 2
};

typedef Str32 IPMStringMsgType;

struct IPMMsgType {
   IPMMsgFormat         format;     /* IPMMsgFormat */
   union{
      OCECreatorType    msgOSType;
      IPMStringMsgType  msgStrType;
   }theType;
};

typedef struct IPMMsgType IPMMsgType;
```

IPMBlockType

The block type structure is defined by the `IPMBlockType` data type.

```
typedef OCECreatorType IPMBlockType;
```

Delivery Notification

The IPM Manager uses a delivery notification flag byte in the message header to determine when to generate recipient report messages and whether to include the original message in any report messages that are returned by the recipients. Report messages include a header (the `IPMReportBlockHeader` structure on page 7-33) and an array of delivery results (the `OCERecipientReport` structure on page 7-33). Report messages are described in “Report Messages” on page 7-9.

Nondelivery Codes

The nondelivery result codes that can be returned by an MSAM or the IPM Manager in a recipient report message are shown here. A personal MSAM can define its own result codes in addition to the ones listed here. (If a server MSAM returns a nonstandard result code, the IPM Manager is unable to convert it to a string meaningful to the user.)

```
enum {
   kIPMNoSuchRecipient                =   0x0001,
   kIPMRecipientMalformed             =   0x0002,
   kIPMRecipientAmbiguous             =   0x0003,
   kIPMRecipientAccessDenied          =   0x0004,
   kIPMGroupExpansionProblem          =   0x0005,
   kIPMMsgUnreadable                  =   0x0006,
   kIPMMsgExpired                     =   0x0007,
   kIPMMsgNoTranslatableContent       =   0x0008,
   kIPMRecipientReqStdCont            =   0x0009,
   kIPMRecipientReqSnapShot           =   0x000A,
   kIPMNoTransferDiskFull             =   0x000B,
   kIPMNoTransferMsgRejectedbyDest    =   0x000C,
   kIPMNoTransferMsgTooLarge          =   0x000D
}
```

Constant descriptions

`kIPMNoSuchRecipient`
: The IPM Manager or MSAM has determined that the specified recipient does not exist. For example, the recipient might have no record in the catalog (and therefore no account on the mail server) or have no account on the MSAM's mail or messaging system.

`kIPMRecipientMalformed`
: The recipient address in the message was not formatted correctly. The problem can be any of the following: The name and record creation ID don't match; both the dNode number and pathname are specified in the record location information (RLI) structure; a dialup address is missing a phone number; an NBP address is missing a zone name; the RLI for a catalog is missing a discriminator; the extension value of the `OCERecipient` structure is not properly formed (as determined by the MSAM interpreting the address).

`kIPMRecipientAmbiguous`
: The IPM Manager or MSAM has been unable to resolve, look up, or find the specified recipient. The recipient may exist but has been unavailable (for example, it has an AppleTalk address but has not been logged on to AppleTalk), or there may be duplicate addresses and the IPM Manager or MSAM cannot determine which to use.

`kIPMRecipientAccessDenied`
: In the process of attempting to deliver the message to the specified recipient, access to some critical information was prevented. The address may be valid and the recipient might exist, but the agent responsible for delivering the message doesn't have access to the recipient's record.

`kIPMGroupExpansionProblem`
: The IPM Manager or MSAM was unable to expand a group address fully. Some of the recipients in the group might have received the message.

`kIPMMsgUnreadable`
: The MSAM was unable to read (and thus to translate) a message (the message might be corrupted or the content missing), and therefore the message was never delivered to the specified recipient.

`kIPMMsgExpired`
: The IPM Manager was unable to confirm delivery of this message before the specified expiration time (currently set at 5 days for PowerShare servers, Direct AppleTalk, and server MSAMs). The server makes several attempts to deliver a message before the message delivery time expires. This result code does not necessarily mean that all the attempts at delivery failed— it means that the server has not been able to determine the success or failure of any of the previous attempts to deliver the message and will make no further attempts.

`kIPMMsgNoTranslatableContent`
: The message is missing a piece of information that is considered critical for its delivery. For example, the message might be missing a subject or a type of content required by the MSAM.

`kIPMRecipientReqStdCont`
: The MSAM cannot deliver messages that don't contain a standard-interchange-format block, and such a block was not present.

`kIPMRecipientReqSnapShot`
: The MSAM required the message to contain a standard image format block (or *snapshot*) in order to deliver it, and such a block was not present.

`kIPMNoTransferDiskFull`
: The recipient could not receive the message because there was insufficient room on the disk to hold it. The recipient might be a user's computer in the case of Direct AppleTalk or a server in the case of an MSAM. If a PowerShare disk is full, the IPM Manager periodically makes new attempts to send the message.

`kIPMNoTransferMsgRejectedbyDest`
: The destination system refused delivery without specifying a reason.

`kIPMNoTransferMsgTooLarge`
: The destination system has a limit to the size of message it accepts, and this message exceeded that limit.

IPMNotificationType

The IPM delivery notification setting is specified by the `IPMNotificationType` data type.

```
typedef Byte IPMNotificationType;
```

The bits in the notification byte are defined as follows:

```
enum {
   kIPMDeliveryNotificationBit      = 0,
   kIPMNonDeliveryNotificationBit   = 1,
   kIPMEncloseOriginalBit           = 2,
   kIPMSummaryReportBit             = 3,
   kIPMOriginalOnlyOnErrorBit       = 4
};
```

You can use a combination of the following values to set the flags in the `IPMNotificationType` data type:

```
enum {
   kIPMNoNotificationMask           = 0x00,
   kIPMDeliveryNotificationMask     = 1<<kIPMDeliveryNotificationBit,
   kIPMNonDeliveryNotificationMask  = 1<<kIPMNonDeliveryNotificationBit,
   kIPMDontEncloseOriginalMask      = 0x00,
   kIPMEncloseOriginalMask          = 1<<kIPMEncloseOriginalBit,
   kIPMImmediateReportMask          = 0x00,
   kIPMSummaryReportMask            = 1<<kIPMSummaryReportBit,
   kIPMOriginalOnlyOnErrorMask      = 1<<kIPMOriginalOnlyOnErrorBit,
   kIPMEncloseOriginalOnErrorMask   =
                        (kIPMOriginalOnlyOnErrorMask|kIPMEncloseOriginalMask)
};
```

Constant descriptions

`kIPMNoNotificationMask`
: Do not deliver any report messages. This setting is overridden when combined with any setting that requests reports.

`kIPMDeliveryNotificationMask`
: Generate a report message when the message arrives at the recipient queue.

`kIPMNonDeliveryNotificationMask`
: Generate a report message if the IPM Manager cannot deliver the message to a recipient.

`kIPMDontEncloseOriginalMask`
: Don't enclose the original message in the report message. This is the default setting for this feature; this setting is overridden by the `kIPMEncloseOriginalMask` setting.

`kIPMEncloseOriginalMask`
: Enclose the original message in a report message. This value must be combined with the `kIPMSummaryReportMask` value.

`kIPMImmediateReportMask`
: Generate a report message for each recipient as soon as there is any information to report. This is the default setting for this feature; this setting is overridden by the `kIPMSummaryReportMask` setting.

`kIPMSummaryReportMask`
: Return a single report message for all recipients.

`kIPMOriginalOnlyOnErrorMask`
: Return the original message only in nondelivery reports. For this setting to have an effect, it must be combined with the `kIPMEncloseOriginalMask` value. The `kIPMEncloseOriginalOnErrorMask` value provides this combination.

`kIPMEncloseOriginalOnErrorMask`
: A combination of the `kIPMEncloseOriginalMask` and `kIPMOriginalOnlyOnErrorMask` values, resulting in the original message being included only in nondelivery reports.

The bit `kIPMSummaryReportBit` in the `IPMNotificationType` byte determines whether the report messages that the sending application receives contain information about a single recipient or all of the recipients of the message. If the bit `kIPMSummaryReportBit` is not set, the IPM Manager returns a report message about each recipient as soon as it is generated. If that bit is set, the IPM Manager creates a single report message that summarizes the requested delivery notification for all of the recipients.

IPMMsgID

The message ID is a unique identifier of the message you sent. The message ID is returned by the `IPMEndMsg` function (page 7-65).

```
struct IPMMsgID {
   unsigned long id[4];
};
```

IPMReportBlockHeader

A recipient report message (message creator `kIPMSignature`, message type `kIPMReportInfo`) includes a report block (which also has a creator of `kIPMSignature` and a type of `kIPMReportInfo`**).** The report block starts with a header, followed by the report data (see Figure 7-4 on page 7-10). The report block header is defined by the `IPMReportBlockHeader` data type.

```
struct IPMReportBlockHeader {
   IPMMsgID    msgID;         /* message ID of the original */
   UTCTime     creationTime;  /* creation time of the report */
};
```

Field descriptions

`msgID` The message ID of the message you sent originally. The recipient report message carries information about this message. The message ID is returned by the `IPMEndMsg` function (page 7-65).

`UTCTime` The time at which the report was generated. The `UTCTime` data type is an unsigned long containing Greenwich Mean Time in seconds since 00:00 hours, January 1, 1904.

OCERecipientReport

A recipient report message (message creator `kIPMSignature`, message type `kIPMReportInfo`) includes a report block (which also has a creator of `kIPMSignature` and a type of `kIPMReportInfo`). The report block starts with a header, followed by the report data (see Figure 7-4 on page 7-10). The report data consists of an array of recipients and delivery results defined by the `OCERecipientReport` data type.

```
struct OCERecipientReport {
   unsigned short rcptIndex;  /* index of recipient in
                                    original message */
   OSErr          result;     /* result of sending letter to
                                    this recipient */
};
```

Field descriptions

`rcptIndex` The index number of the recipient in the header of the original message. In the case of group addresses, the delivery report tells you only that the group address was expanded; you don't receive information on delivery to individual members of a group.

`result` The result of the attempt to deliver the message to this recipient. The standard values returned in this field are shown on page 7-29; in addition, each personal MSAM can define its own result codes.

To calculate the number of recipients in a report, divide the size of the block (minus the header size) by the size of an `OCERecipientReport` structure.

```
numRecipients = (pmPB.readMsgPB.actualCount
 - sizeof (IPMReportBlockHeader)) / sizeof (OCERecipientReport);
```

Filter Structures

When you open a message queue or enumerate the messages in the queue, you can apply a filter to the queue so that the IPM Manager lists only the messages that match your filter criteria.

The IPM Manager defines a queue filter as an array of single filters. It performs an OR operation on all of the single filters you specify for a queue filter. For example, if you set one single filter in the filter array to pass high-priority messages of type `'high'` and another single filter to pass low-priority messages of type `'low '`, the queue filter passes messages of both descriptions. The OR operation is performed on the entire single filters, not on the individual fields in the single filters; thus the filter in this example would not pass a low-priority message of type `'high'`.

This section provides the data structures that define single filters and queue filters.

IPMSingleFilter

The `IPMSingleFilter` data type describes the contents of a single filter. You must pack and word-align each field of the structure before you pass it to an IPM routine.

```
struct IPMSingleFilter{
   IPMPriority    priority;
   Byte           padByte;
   OSType         family;  /* family to which this msg belongs */
   ScriptCode     script;  /* language identifier */
   IPMProcHint    hint;
   IPMMsgType     msgType;
};
```

Field descriptions

`priority` The priority of the message. You can set the priority to any of the following values:

`kIPMAnyPriority`
`kIPMNormalPriority`
`kIPMLowPriority`
`kIPMHighPriority`

If you set the filter priority to `kIPMAnyPriority`, the queue does not filter messages according to their priority settings.

`family`	The message family to which the message belongs. You can use the wildcard value `kIPMFamilyWildCard` for all families.
`script`	Reserved.
`hint`	A process hint value. A process hint is a string of up to 32 characters, defined by the creator of the message.
`msgType`	A message type. The message type is assigned by the creator of the message. You can use the wildcard value `kIPMTypeWildCard` for either or both fields of the `IPMMsgType` structure to pass messages with any creator or any type. The `IPMMsgType` data type is defined on page 7-28.

The IPM Manager defines the following message family types:

```
#define kIPMFamilyUnspecified 0              /* any message */

#define kIPMFamilyWildCard    0x3F3F3F3FL    /* '????' */
```

In addition, the AOCE MSAM interface defines the following message family types:

```
#define kMailFamily        'mail'   /* "mail" msgs: content, header, etc */

#define kMailFamilyFile    'file'   /* "direct display" msgs */
```

In addition to the types shown here, Apple Computer reserves for its own use any message family type consisting entirely of lowercase letters.

IPMFilter

A full queue filter is a packed array of single filters. The contents of a filter are shown by the `IPMFilter` data type.

```
struct IPMFilter{
   unsigned short    count;
   IPMSingleFilter   sFilters[1];
};
```

Field descriptions

`count`	The number of single filters in this queue filter.
`sFilters`	An array of single filters.

Message Information Structure

When you call the `IPMEnumerateQueue` function (page 7-80) or the `IPMGetMsgInfo` function (page 7-87), the function returns the information about the message in an message information structure.

IPMMsgInfo

The message information structure is defined by the `IPMMsgInfo` data type.

```
struct IPMMsgInfo{
   IPMSeqNum         sequenceNum;
   unsigned long     userData;
   unsigned short    respIndex;
   Byte              padByte;
   IPMPriority       priority;
   unsigned long     msgSize;
   unsigned short    originalRcptCount;
   unsigned short    reserved;
   UTCTime           creationTime;
   IPMMsgID          msgID;
   OSType            family;  /* family of this msg */
   IPMProcHint       procHint;/* packed and even-length padded */
   IPMMsgType        msgType; /* packed and even-length padded */
};
```

The `IPMEnumerateQueue` function lets you specify whether the returned `IPMMsgInfo` structure includes the `procHint` or `msgType` fields. Because these fields are of variable length, the offset to the `msgType` field depends on the presence and length of the `procHint` field.

Field descriptions

`sequenceNum`	A sequence number that uniquely identifies a particular message in the queue.
`userData`	Reserved.
`respIndex`	Reserved.
`priority`	The priority setting of the message. This field can be set to `kIPMNormalPriority`, `kIPMLowPriority`, or `kIPMHighPriority`.
`msgSize`	The length of the entire message.

`originalRcptCount`
: The number of recipients that the sending application originally specified for the message. This value may differ from the actual number of recipients if the message was sent to one or more groups.

`reserved`
: Reserved.

`creationTime`
: The date and time that the message was created. The `UTCTime` data type is an unsigned long containing Greenwich Mean Time in seconds since 00:00 hours, January 1, 1904.

`msgID`
: A unique identifier of the message. The message ID is returned by the `IPMEndMsg` function (page 7-65).

`family`
: The message family to which the message belongs. Possible values for this field are shown on page 7-35.

`procHint`
: An optional field of varied length. If this field is present, it contains the process hint for the message, which is a Pascal-type string of up to 32 characters, defined by the creator of the message. The information in the field is packed. If the field contains an odd number of bytes (including the length byte), the IPM Manager adds a pad byte following the field. Therefore, the maximum length of this field (including the length byte and the pad byte) is 34 bytes.

`msgType`
: An optional parameter that contains the message type of the message. The `IPMMsgType` data type is defined on page 7-28. Like the `procHint` field, the `msgType` field is packed and padded if necessary to contain an even number of bytes.

Header Information Structures

The `IPMReadHeader` function (page 7-89) uses the data structures in this section to return information from a message header.

IPMTOC

When you specify the value `kIPMTOC` for the `fieldSelector` field in the parameter block used by the `IPMReadHeader` function, the function returns an array of TOC information structures—one for each block in the message. The TOC information structure is defined by the `IPMTOC` data type.

```
struct   IPMTOC
{
   IPMBlockType      blockType;
   long              blockOffset;
   unsigned long     blockSize;
   unsigned long     blockRefCon;
};
```

Field descriptions

`blockType`	The creator and type of the block.
`blockOffset`	The offset from the start of the message to the start of the block.
`blockSize`	The size, in bytes, of the block.
`blockRefCon`	The block's reference constant. The application that creates the message specifies this value when it adds the block to the message. The meaning of this reference constant is defined by the application that creates the message.

IPMFixedHdrInfo

When you specify the value `kIPMFixedInfo` for the `fieldSelector` field of the parameter block used by the `IPMReadHeader` function, the function returns information about the message header in a fixed header information structure. The fixed header information structure is defined by the `IPMFixedHdrInfo` data type.

```
struct IPMFixedHdrInfo {
   unsigned short       version;          /* IPM Manager version */
   Boolean              authenticated;    /* was message authenticated? */
   Boolean              signatureEnclosed;/* digital signature enclosed? */
   unsigned long        msgSize;          /* size of message */
   IPMNotificationType  notification;     /* notification type requested */
   IPMPriority          priority;         /* message priority */
   unsigned short       blockCount;       /* number of blocks */
   unsigned short       originalRcptCount;/* original number of recipients */
   unsigned long        refCon;           /* application-defined data */
   unsigned short       reserved;         /* reserved */
   UTCTime              creationTime;     /* message creation time */
   IPMMsgID             msgID;            /* message ID */
   OSType               family;           /* family of this msg */
};
```

Field descriptions

`version`	The version number of the IPM Manager that created the message. This is not necessarily the same version number as that returned by the `Gestalt` function for the Collaboration toolbox (see page 7-17).
`authenticated`	A Boolean value that indicates whether the message was authenticated. In the case of a message that passes through more than one store-and-forward server, the IPM Manager sets this field to `true` only if the identities of the original sender and of every server in the routing chain were authenticated.

`signatureEnclosed`
: A Boolean value indicating whether the message includes a digital signature. If this field is set to `true`, the message includes a block with a creator of `kIPMSignature` and a type of `kIPMDigitalSignature` containing a digital signature. You can use the `IPMVerifySignature` function (page 7-102) to verify the digital signature.

`msgSize`
: The length, in bytes, of the message.

`notification`
: The delivery notification requested by the application that sent the message. See "Delivery Notification," beginning on page 7-28, for more information about this value.

`priority`
: The priority setting of the message. Values for this field can be `kIPMNormalPriority`, `kIPMLowPriority`, or `kIPMHighPriority`.

`blockCount`
: The number of blocks in the message. You can use the `IPMGetBlkIndex` function (page 7-96) to list the creator, type, and position of each block in the message.

`originalRcptCount`
: The number of recipients in the recipient list that the sending application originally specified for the message. Because the IPM Manager might have expanded groups in the original recipient list, the number of recipients in the current recipient list might be different from this.

`refCon`
: A numeric reference value that the sending application provides for the message when it calls the `IPMNewMsg` function (page 7-43), the `IPMNewHFSMsg` function (page 7-47), or the `IPMNewNestedMsgBlock` function (page 7-56).

`reserved`
: Reserved.

`creationTime`
: The date and time that the message was created. The `UTCTime` data type is an unsigned long containing Greenwich Mean Time in seconds since 00:00 hours, January 1, 1904.

`msgID`
: A unique identifier of the message. The message ID is returned by the `IPMEndMsg` function (page 7-65).

`family`
: The family the message belongs to.

Sender Structure

When you create a new message or read a message header, the name of the originator of the message is held in a sender structure, described in this section. In the case of an application-to-application message, the sender would be an application name. In the case of a message or letter sent by a user, the sender might be the user's name or a record ID that identifies the user record for the sender.

IPMSender

The sender structure contains either the sender's name in `RString` format or a catalog record ID that identifies the user record for the sender of the message. The sender structure is defined by the `IPMSender` data type.

```
struct IPMSender {
   IPMSenderTag         sendTag;
   union {
      RString           rString;
      PackedRecordID    rid;
   } theSender;
};

enum {
   kIPMSenderRStringTag,
   kIPMSenderRecordIDTag
};
typedef unsigned short IPMSenderTag;
```

Interprogram Messaging Parameter Block Header

All IPM Manager function declarations include a pointer to a parameter block. Each parameter block begins with the following fields:

```
#define IPMParamHeader     \
   Ptr      qLink;         \
   long     reservedH1;    \
   long     reservedH2;    \
   ProcPtr  ioCompletion;  \
   OSErr    ioResult;      \
   long     saveA5;        \
   short    reqCode;
```

Field descriptions

`qLink` Reserved.

`reservedH1` Reserved.

`reservedH2` Reserved.

`ioCompletion` A pointer to a completion routine that you provide. If you provide a pointer to a completion routine in this field, the function calls your completion routine when it completes execution. Completion routines are described in"Application-Defined Functions," beginning on page 7-114. Specify `nil` for this parameter if you do not want to supply a completion routine.

`ioResult`	The function result. If you call the function asynchronously, it sets this field to 1 to indicate that the request was queued successfully. The function sets this field to the function result when it completes execution.
`saveA5`	Reserved.
`reqCode`	Reserved.

The individual routine descriptions at the end of this reference contain information about any additional parameters that are specific to the routine.

Asynchronous or Synchronous Operations

You can call the IPM Manager routines either synchronously or asynchronously. If you call the function asynchronously, it returns control to you immediately and completes execution incrementally as it is given time by the system. If you call it synchronously, it completes execution before returning control to you.

IMPORTANT

You must specify asynchronous operation when you call any IPM function at interrupt time. Because a function might not complete successfully, calling it synchronously might cause the computer to hang. s

Completion Routines and Polling Options

When you call an IPM function asynchronously, you can specify a completion routine. The IPM Manager calls your completion routine when the function completes execution. If you write you completion routine in Pascal or C, it must take a single argument, which is a pointer to the parameter block.

For example, to declare a completion routine in Pascal, you could use the following statement:

```
PROCEDURE MyCompletionRoutine (paramBlk: Ptr);
```

To declare a completion routine in C, you could use the following statement:

```
pascal void MyCompletionRoutine (Ptr paramBlk);
```

If you write your completion routine in assembly language, you can find a pointer to the parameter block in the A0 register and the function result in the D0 register.

The IPM Manager saves the value of your A5 register at the time you call an IPM function and restores the A5 value before calling your completion routine.

If you do not provide a completion routine, you can poll the `ioResult` field of the parameter block. The IPM Manager sets the value of the `ioResult` field to 1 when you first call a function asynchronously, indicating that the function was successfully queued. When the function completes execution, the IPM Manager changes the `ioResult` value to the actual function result.

IPM Manager Functions

This section describes all of the functions provided by the IPM Manager except for those specifically for use by MSAMs; see the chapter "Messaging Service Access Modules" in *Inside Macintosh: AOCE Service Access Modules* for descriptions of MSAM functions.

In the functions described here, you must completely specify any data structure that you provide to a function unless the description states otherwise.

All of the functions take a pointer to an `IPMParamBlock` 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 IPM API have the following form:

```
pascal OSErr function (IPMParamBlockPtr paramBlock,
                       Boolean async);
```

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

```
pascal OSErr function (IPMParamBlockPtr paramBlock);
```

The function returns its result code in the `ioResult` field of the parameter block. When you call a function synchronously, it 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 your completion routine if you specified one. 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 its value has changed from 1 to some other value. Although the IPM Manager does not return positive error codes, system utilities may return positive error codes and these may be passed through. s

Calling an IPM Function From Assembly Language

You can call a function from assembly language. Listing 7-1 illustrates one way to do this for a function that takes both the parameter block pointer and the `async` flag as parameters. (If a function can be called only synchronously or only asynchronously, the assembly code would not manipulate the `async` value.)

Listing 7-1 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
   MOVEQ    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
```

Note
The functions described in the section "Utility Functions," beginning on page 7-107 use a different assembly-language calling convention, described on page 7-107. u

Creating a New Message

This section describes the functions that you use to create a new message and either send it or save it to disk. See "Creating a Message," beginning on page 7-18 for information about the sequence in which you use these functions to create a message.

IPMNewMsg

The `IPMNewMsg` function starts the process of creating a new message to be sent to a recipient.

```
pascal OSErr IPMNewMsg(IPMParamBlockPtr paramBlock,
                       Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	ioCompletion	ProcPtr	Pointer to a completion routine.
←	ioResult	OSErr	Result of the function.
→	recipient	OCERecipient*	Pointer to the recipient's queue address.
→	replyQueue	OCERecipient*	Pointer to the queue address for message replies.
→	procHint	StringPtr	Pointer to character string for your use.
→	msgType	IPMMsgType*	Pointer to the message type.
→	refCon	unsigned long	Reserved for your use.
←	newMsgRef	IPMMsgRef	Message reference number.
→	identity	AuthIdentity	Authentication identity.
→	sender	IPMSender*	Pointer to the sender's name.

See "Interprogram Messaging Parameter Block Header" on page 7-40 for descriptions of the ioCompletion and ioResult fields.

Field descriptions

`recipient` A pointer to an `OCERecipient` structure that either specifies a destination message queue or that identifies a catalog record from which the IPM Manager can obtain the destination queue address.

Set this field to `nil` if you intend to use the `IPMAddRecipient` function to add all the recipient addresses later.

`replyQueue` A pointer to an `OCERecipient` structure that specifies the queue in which you receive your incoming messages. The `OCERecipient` structure can specify the reply queue directly, or can specify a record in a catalog that contains the reply queue information.

If you specify `nil` for this field and a local identity for the `identity` field, the IPM Manager uses the PowerTalk Setup catalog to fill in the reply queue field in the message header at the time the message is sent.

You can also set this field to `nil` if you intend to use the `IPMAddReplyQueue` function to specify the reply queue later.

`procHint` A pointer to a process hint, which is a string of up to 32 characters, reserved for your use. The IPM Manager puts this string into the message header. You can use this field, for example, to provide information that helps your recipients determine how to process the message.

`msgType` A pointer to an `IPMMsgType` structure, which specifies the type of message that you are creating. The IPM Manager and other AOCE components do not read the message type; it is for the use of applications only. Note, however, that the Finder might display the contents of the message header's message-type field if the user displays the Info dialog box for the message while the message is in the Out Tray.

refCon	An unsigned 4-byte number, reserved for your use. The IPM Manager puts this value into the message header. You can use this field, for example, to indicate that the message has content of some particular type.
newMsgRef	A reference number returned by the function. You must use this number when you call other functions to complete the process of creating the message.
identity	The authentication identity of the creator of the message. If you provide a nonzero value for this field, the IPM Manager uses this information to fill in the sender field in the message header.
sender	A pointer to an IPMSender structure, which identifies the sender of the message. If you specify 0 or a local identity for the identity field, you should provide a meaningful value for the sender. For an application-to-application message, you might use the application name as the sender. To specify an individual as a sender, you can put the user's name in the IPMSender structure or you can provide the record ID of the sender's user record. If you specify a specific identity for the identity field, the function ignores the sender field.

DESCRIPTION

You must call the IPMNewMsg function to begin the process of creating a new message that is to be sent to a recipient. (Use the IPMNewHFSMsg function to start a message to be saved on disk.) The IPM Manager uses information that you provide in the parameter block of IPMNewMsg to fill in fields of the message header of the new message.

The IPM Manager uses the information you provide in the recipient field to determine where to send the message and returns any delivery or nondelivery reports to the queue that you specify in the replyQueue field. If you do not know the recipient at the time you call IPMNewMsg function, or if you have more than one recipient, you can use the IPMAddRecipient function to provide the recipients. If you do not know the reply queue at the time you call the IPMNewMsg function, you can use the IPMAddReplyQueue function to add the reply queue later.

If the recipient or replyQueue fields specify a record in a PowerShare catalog, the IPM Manager looks up the catalog records at the time it sends the message.

Note

Because the PowerShare server acts as a trusted agent when resolving addresses in catalogs, the sender of the message need not have the access privileges necessary to read these addresses. u

The IPM Manager uses any specific identity you provide in the identity field to fill in the sender field in the message header. If the IPM Manager and each intervening store-and-forward server can authenticate the message, the recipient can then rely on the sender field to indicate the authenticated originator of the message. If you specify 0 or a local identity for the identity field, then you should provide a meaningful value for the sender field, such as the name of the originator of the message.

Note
If you specify either a local identity or 0 for the `identity` field, the IPM Manager stores the message on the local computer until transmitting it. If you provide a specific identity, the IPM Manager creates the message on the computer containing the PowerShare server to which that identity provides access. u

You can use the `SDPPromptForIdentity` function to obtain an identity for the originator of the message. This function allows the user to decide whether to provide a local identity, a specific identity, or no identity (guest access). The `SDPPromptForIdentity` function returns to your application the identity plus a value that tells you which kind of identity it is. To obtain a local identity without displaying a dialog box, use the `AuthGetLocalIdentity` function.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0402

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMInvalidMsgType`	–15091	Message type is invalid
`kIPMInvalidProcHint`	–15092	Process hint is invalid
`kIPMMsgTypeReserved`	–15095	Message type reserved for system use
`kIPMNestedMsgOpened`	–15097	Nested message opened; cannot do operation
`kIPMA1HdrCorrupt`	–15098	Message is corrupt; may not be message
`kIPMCorruptDataStructures`	–15099	Message is corrupt
`kIPMInvalidSender`	–15103	Sender is invalid
`kIPMStreamErr`	–15108	Error on stream
`kIPMPortClosed`	–15109	Stream closed

SEE ALSO

See “Message Addressing Structures” on page 7-24 for more detailed information about the format and contents of an `OCERecipient` structure.

The `IPMSender` structure is described on page 7-39.

You can use the `IPMAddRecipient` function (page 7-50) to add recipient addresses to a message.

You can use the `IPMAddReplyQueue` function (page 7-52) to specify the reply queue.

See “Message and Block Types” on page 7-26 for more information about the `IPMMsgType` structure.

The `RString` structure and record IDs are described in the chapter “Introduction to the Apple Open Collaboration Environment” in this book.

You can use the `SDPPromptForIdentity` function to obtain an identity. That function is described in the chapter “Standard Catalog Package” in this book. You can use the `AuthGetLocalIdentity` function to obtain a local identity. See the chapter “Authentication Manager” in this book for a description of the `AuthGetLocalIdentity` function.

Use the `IPMNewHFSMsg` function, described next, to start a message to be saved on disk.

IPMNewHFSMsg

The `IPMNewHFSMsg` function starts the process of creating a new message to be saved as an HFS file on disk.

```
pascal OSErr IPMNewHFSMsg(IPMParamBlockPtr paramBlock,
                          Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async`
A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result of the function.
→	`hfsPath`	`FSSpec*`	Specifier of the file in which to save the message.
→	`recipient`	`OCERecipient*`	Pointer to the recipient’s queue address.
→	`replyQueue`	`OCERecipient*`	Pointer to the queue address for message replies.
→	`procHint`	`StringPtr`	Pointer to a character string for your use.
→	`msgType`	`IPMMsgType*`	Pointer to the message type.
→	`refCon`	`unsigned long`	Reserved for your use.
←	`newMsgRef`	`IPMMsgRef`	Message reference number.
→	`identity`	`AuthIdentity`	Authentication identity.
→	`sender`	`IPMSender*`	Pointer to the sender’s name.

See “Interprogram Messaging Parameter Block Header” on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`hfsPath` A pointer to the file system specification structure that describes the file in which you wish to save the message.

`recipient` A pointer to an `OCERecipient` structure that either specifies a destination message queue or that identifies a catalog record from which the IPM Manager can obtain the destination queue address.

Set this field to `nil` if you intend to use the `IPMAddRecipient` function to add all the recipient addresses later.

`replyQueue` A pointer to an `OCERecipient` structure that specifies the queue in which you receive your incoming messages. The `OCERecipient` structure can specify the reply queue directly, or can specify a record in a catalog that contains the reply queue information.

Set this field to `nil` if you intend to use the `IPMAddReplyQueue` function to specify the reply queue later.

`procHint` A pointer to a process hint, which is a string of up to 32 characters, reserved for your use. The IPM Manager puts this string into the message header. You can use this field, for example, to provide information that helps your recipients determine how to process the message.

`msgType` A pointer to an `IPMMsgType` structure, which specifies the type of message that you are creating. The IPM Manager and other AOCE components do not read the message type; it is for the use of applications only.

`refCon` An unsigned 4-byte number, reserved for your use. The IPM Manager puts this value into the message header. You can use this field, for example, to indicate that the message has content of some particular type.

`newMsgRef` A reference number returned by the function. You must use this number when you call other functions to complete the process of creating the message.

`identity` The authentication identity of the creator of the message. If you provide a nonzero value for this field, the IPM Manager uses this information to fill in the `sender` field in the message header.

`sender` A pointer to an `IPMSender` structure, which identifies the sender of the message. If you specify 0 or a local identity for the `identity` field, you should provide a meaningful value for the sender. For an application-to-application message, you might use the application name as the sender. To specify an individual as a sender, you can put the user's name in the `IPMSender` structure or you can provide the record ID of the sender's user record. If you specify a specific identity value for the `identity` field, the function ignores the `sender` field.

DESCRIPTION

You must call the `IPMNewHFSMsg` function to begin the process of creating a new message that is to be saved to a file on disk. (Use the `IPMNewMsg` function to start a message to be sent to a recipient.) The IPM Manager fills in fields of the message header of the new message from information that you provide in the parameter block of the `IPMNewHFSMsg` function.

The IPM Manager uses any specific identity you provide in the `identity` field to fill in the `sender` field in the message header. If you specify 0 or a local identity for the `identity` field, then you should provide a meaningful value for the `sender` field, such as the name of the originator of the message.

Note

The IPM Manager does not provide any way to send a message that has been saved on disk. If you want to send a message and in addition save it to disk, you must build the message twice, once using the `IPMNewHFSMsg` function and once using the `IPMNewMsg` function. u

You can use the `SDPPromptForIdentity` function to obtain an identity for the originator of the message. This function allows the user to decide whether to provide a local identity, a specific identity, or no identity (guest access). The `SDPPromptForIdentity` function returns to your application the identity plus a value that tells you which kind of identity it is. To obtain a local identity without displaying a dialog box, use the `AuthGetLocalIdentity` function.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$041E

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter

SEE ALSO

See “Message Addressing Structures” on page 7-24 for more detailed information about the format and contents of an `OCERecipient` structure.

The `IPMSender` structure is described on page 7-39.

You can use the `IPMAddRecipient` function (page 7-50) to add recipient addresses to a message.

You can use the IPMAddReplyQueue function (page 7-52) to specify the reply queue.

See "Message and Block Types" on page 7-26 for more information about the IPMMsgType structure.

The RString structure and record IDs are described in the chapter "Introduction to the Apple Open Collaboration Environment" in this book.

You can use the SDPPromptForIdentity function to obtain an identity. That function is described in the chapter "Standard Catalog Package" in this book. You can use the AuthGetLocalIdentity function to obtain a local identity. See the chapter "Authentication Manager" in this book for a description of the AuthGetLocalIdentity function.

Use the IPMNewMsg function (page 7-43) to start a message to be sent.

IPMAddRecipient

The IPMAddRecipient function adds a recipient to a new message that you are creating.

```
pascal OSErr IPMAddRecipient(IPMParamBlockPtr paramBlock,
                             Boolean async);
```

paramBlock
: A pointer to a parameter block.

async
: A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to true for asynchronous execution.

See "Interprogram Messaging Parameter Block Header" on page 7-40 for descriptions of the ioCompletion and ioResult fields.

Parameter block

→	ioCompletion	ProcPtr	Pointer to a completion routine.
←	ioResult	OSErr	Result of the function.
→	msgRef	IPMMsgRef	Message reference number.
→	recipient	OCERecipient*	Pointer to the recipient's queue address.

Field descriptions

msgRef
: The message reference number of the message or nested-message block to which you want to add a recipient. This number is returned by the IPMNewMsg function for a message you intend to send, by the IPMNewHFSMsg function for a message you intend to save to disk, and by the IPMNewNestedMsgBlock function for a nested-message block.

`recipient`	A pointer to an `OCERecipient` structure that either specifies a destination message queue or that identifies a catalog record from which the IPM Manager can obtain the destination queue address.

DESCRIPTION

You can call the `IPMAddRecipient` function at any time during the message-creation process to add a recipient to the message. You repeat this call for each recipient that you add to the message (except for recipients that belong to a group; see "Message Addressing Structures" on page 7-24). You can add only one recipient to a message when you call the `IPMNewMsg` or `IPMNewHFSMsg` function; if you want to add more than one recipient to a message, you must call the `IPMAddRecipient` function.

When you call the `IPMAddRecipient` function for a new message, the function adds the specified recipient to the message header. If you are working with a nested message, the function adds the recipient to the header of the nested message.

If the `recipient` parameter specifies a record in a catalog, the `IPMAddRecipient` function does not look up the address of the recipient in the catalog. The IPM Manager looks up catalog records when you send the message.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0403

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMCorruptDataStructures`	–15099	Message is corrupt

SEE ALSO

See "Message Addressing Structures" on page 7-24 for more detailed information about the format and contents of an `OCERecipient` structure.

IPMAddReplyQueue

The `IPMAddReplyQueue` function adds the reply queue to the header of a new message that you are creating.

```
pascal OSErr IPMAddReplyQueue(IPMParamBlockPtr paramBlock,
                              Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result of the function.
→	`msgRef`	`IPMMsgRef`	Message reference number.
→	`replyQueue`	`OCERecipient*`	Pointer to the queue address for message replies.

See "Interprogram Messaging Parameter Block Header" on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`msgRef` The message reference number of the message or nested-message block to which you want to add the reply queue. This number is returned by the `IPMNewMsg` function for a message you intend to send, by the `IPMNewHFSMsg` function for a message you intend to save to disk, and by the `IPMNewNestedMsgBlock` function for a nested-message block.

`replyQueue` A pointer to an `OCERecipient` structure that specifies the queue in which you receive your incoming messages. The `OCERecipient` structure can specify the reply queue directly or specify a record in a catalog that contains the reply queue information.

If you specify `nil` for this field and specified a local identity for the `identity` field in the `IPMNewMsg` function, the IPM Manager uses the PowerTalk Setup catalog to fill in the reply queue field in the message header at the time the message is sent.

DESCRIPTION

You can call the `IPMAddReplyQueue` function at any time during the message-creation process. When you call the `IPMAddRecipient` function for a new message, the function adds the specified reply queue to the message header. If you are working with a nested message, the function adds the reply queue to the header of the nested message.

Each message or nested message has only one reply queue. If you have already specified a reply queue for the message that you specify in the `msgRef` field, the `IPMAddReplyQueue` function returns the `kOCEParamErr` result code.

If the `replyQueue` parameter specifies a record in a catalog, the `IPMAddRecipient` function does not look up the address of the reply queue in the catalog. The IPM Manager resolves addresses in catalog records at the time a message is sent.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$041D

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMCorruptDataStructures`	–15099	Message is corrupt

SEE ALSO

See “Message Addressing Structures” on page 7-24 for more detailed information about the format and contents of an `OCERecipient` structure.

IPMNewBlock

The `IPMNewBlock` function creates a new block at the end of the message or nested message that you are currently recording and returns the offset to its starting point.

```
pascal OSErr IPMNewBlock(IPMParamBlockPtr paramBlock,
                         Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	ioCompletion	ProcPtr	Pointer to a completion routine.
←	ioResult	OSErr	Result of the function.
→	msgRef	IPMMsgRef	Message reference number.
→	blockType	IPMBlockType	Type of block you are adding.
→	refCon	unsigned long	Reserved for your use.
←	startingOffset	long	Offset to new block.

See "Interprogram Messaging Parameter Block Header" on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`msgRef` The message reference number of the message or nested message to which you want to add the block. This number is returned by the `IPMNewMsg` function for a message you intend to send, by the `IPMNewHFSMsg` function for a message you intend to save to disk, and by the `IPMNewNestedMsgBlock` function for a nested message.

`blockType` A pointer to an `IPMBlockType` data type that specifies the type of block that you are adding to the message.

`refCon` An unsigned 4-byte number, reserved for your use. The IPM Manager puts this value into the TOC field of the message header. You can use this field, for example, to identify block subtypes for your own use.

`startingOffset`
The offset, in bytes, from the start of the message body to the start of the new block. This value is returned by the function. You can use this offset as a starting point when you call the `IPMWriteMsg` function to add data to the block.

DESCRIPTION

You can call the `IPMNewBlock` function at any time during the message-creation process to create a new message block.

The `IPMNewBlock` function creates the new block at the end of the message, records the offset to the new block, and then returns the offset to you. You can use this value to determine the offset to provide to the `IPMWriteMsg` function when you add data to the block or overwrite data in the block.

Note

The IPM Manager does not allow you to modify the starting point of a block. When you call the `IPMNewBlock` function to create a new block, you freeze the size of the previous block. You can use the `IPMWriteMsg` function to overwrite data in an existing block, but if you try to write more data than was originally in the block, you write over the block boundary into the following block. u

A nested message is contained entirely within a single block of the enclosing message and has exactly the same structure as any other message (see Figure 7-2 on page 7-5). Once you have called the `IPMNewNestedMsgBlock` function to start a nested message, you must call the `IPMEndMsg` function to end the nested message before adding another block to the outer message. If you specify the message reference of an outer message before completing a nested message, the `IPMNewBlock` function returns the `kIPMNestedMsgOpened` result code.

SPECIAL CONSIDERATIONS

If you specify `kIPMSignature` as the creator of the block in the `IPMBlockType` data type, the function returns the `kIPMMsgTypeReserved` result code.

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0404

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMMsgTypeReserved`	–15095	The `blockType` parameter specifies a block type reserved for system use
`kIPMNestedMsgOpened`	–15097	The message reference in the `msgRef` parameter specifies an outer message, but nested message is not yet closed
`kIPMCorruptDataStructures`	–15099	Message is corrupt

SEE ALSO

The `IPMBlockType` data type is defined on page 7-28.

You start a new message by calling the `IPMNewMsg` function (page 7-43) or the `IPMNewHFSMsg` function (page 7-47). You start a new nested message by calling the `IPMNewNestedMsgBlock` function (next).

You can use the `IPMWriteMsg` function (page 7-61) to add data to the block.

IPMNewNestedMsgBlock

The `IPMNewNestedMsgBlock` function starts a new nested message from information that you provide to the function. Use this function to begin recording a new nested message that you create from scratch.

```
pascal OSErr IPMNewNestedMsgBlock(IPMParamBlockPtr paramBlock,
                                  Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result of the function.
→	`msgRef`	`IPMMsgRef`	Message reference number.
→	`recipient`	`OCERecipient*`	Pointer to the recipient's queue address.
→	`replyQueue`	`OCERecipient*`	Pointer to the queue address for message replies.
→	`procHint`	`StringPtr`	Pointer to character string for your use.
→	`msgType`	`IPMMsgType*`	Pointer to the message type.
→	`refCon`	`unsigned long`	Reserved for your use.
←	`newMsgRef`	`IPMMsgRef`	Message reference number.
←	`startingOffset`	`long`	Offset to the start of the nested message.
→	`identity`	`AuthIdentity`	Authentication identity.
→	`sender`	`IPMSender*`	Pointer to sender's name.

See "Interprogram Messaging Parameter Block Header" on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`msgRef` The message reference number of the message or nested message into which you want to insert the new nested message. This number is returned by the `IPMNewMsg` function for a message you intend to send, by the `IPMNewHFSMsg` function for a message you intend to save to disk, and by the `IPMNewNestedMsgBlock` function for a nested message.

`recipient` A pointer to an `OCERecipient` structure that either specifies a destination message queue or that identifies a catalog record from which the IPM Manager can obtain the destination queue address.

Set this field to `nil` if you intend to use the `IPMAddRecipient` function to add all the recipient addresses later.

`replyQueue`	A pointer to an `OCERecipient` structure that specifies the queue in which you receive your incoming message reports. In most cases, you have only one message queue. Set this field to `nil` if you intend to use the `IPMAddReplyQueue` function to specify the reply queue later.
`procHint`	A pointer to a process hint, which is a string of up to 32 characters, reserved for your use. The IPM Manager puts this string into the nested-message header. You can use this field, for example, to provide information that helps your recipients determine how to process the nested message.
`msgType`	A pointer to an `IPMMsgType` structure, which specifies the type of nested message that you are creating. This value is application dependent and is not read by any AOCE component.
`refCon`	An unsigned 4-byte number, reserved for your use. The IPM Manager puts this value into the nested-message header.
`newMsgRef`	A reference number returned by the function. You must use this number when you call the `IPMAddRecipient`, `IPMAddReplyQueue`, `IPMNewBlock`, `IPMWriteMsg`, `IPMNestMsg`, `IPMNewNestedMsgBlock`, and `IPMEndMsg` functions to complete the process of creating this nested message.
`startingOffset`	The offset in bytes to the start of the new nested-message block from the start of the enclosing message body. This value is returned by the function.
`identity`	The authentication identity of the creator of the message. If you provide a nonzero value for this field, the IPM Manager uses this information to fill in the `sender` field in the message header.
`sender`	A pointer to an `IPMSender` structure, which identifies the sender of the message. If you specify 0 or a local identity for the `identity` field, you should provide a meaningful value for the sender. For an application-to-application message, you might use the application name as the sender. To specify an individual as a sender, you can put the user's name in the `IPMSender` structure or you can provide the record ID of the sender's user record. If you specify a specific identity in the `identity` field, the function ignores the `sender` field.

DESCRIPTION

You can call the `IPMNewNestedMsgBlock` function at any time during the message-creation process to start a new nested message.

The `IPMNewNestedMsgBlock` function first creates a new block at the end of the message. The `msgCreator` field of the block type of the new block is equal to the constant `kIPMSignature` and the `msgType` field is equal to `kIPMEnclosedMsgType`. The `IPMNewNestedMsgBlock` function then fills in fields of the message header of the new nested message from information that you provide in the parameter block of the `IPMNewNestedMsgBlock` function.

Note that, because the header of the nested message is located within a block of the enclosing message, the IPM Manager does not read the nested-message header and so does not use the information in its message-delivery process.

The IPM Manager uses any specific identity you provide in the `identity` field to fill in the `sender` field in the message header. If you specify 0 or a local identity for the `identity` field, then you should provide a meaningful value for the `sender` field, such as the name of the originator of the message.

After you call the `IPMNewNestedMsgBlock` function to start a nested message, you can call the `IPMNewBlock` function to add a new block to the nested message, the `IPMNewNestedMsgBlock` function to nest another message within the nested message, or any of the functions that add information to the message header or to the body of the message. When you call any of these functions, you must pass the message reference value returned by the `IPMNewNestedMsgBlock` function.

You must call the `IPMEndMsg` function to complete the nested message before you can add any more information to the enclosing message. After you call the `IPMEndMsg` function to end the nested message, you cannot add any recipients or blocks to the nested message.

SPECIAL CONSIDERATIONS

Although the IPM Manager allows you to add any number of nested-message blocks at the same nesting level in a message, the MSAM interface does not support this feature. Therefore, if you want your message to be compatible with MSAMs, you must not add more than one nested-message block at a given level of nesting. You can, however, nest a message within another nested message to as many nesting levels as disk and memory resources allow.

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0405

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMInvalidMsgType`	–15091	Message type is invalid
`kIPMInvalidProcHint`	–15092	Process hint is invalid
`kIPMCorruptDataStructures`	–15099	Message is corrupt
`kIPMInvalidSender`	–15103	Sender is invalid

SEE ALSO

See "Message Addressing Structures" on page 7-24 for more detailed information about the format and contents of an `OCERecipient` structure.

The `IPMSender` structure is described in "Sender Structure" on page 7-39.

You can use the `IPMAddRecipient` function (page 7-50) to add recipient addresses to a message.

You can use the `IPMAddReplyQueue` function (page 7-52) to specify the reply queue.

See "Message and Block Types" on page 7-26 for more information about the `IPMMsgType` structure.

IPMNestMsg

The `IPMNestMsg` function creates a new block at the end of the specified new message and stores the existing message that you specify into the new block.

```
pascal OSErr IPMNestMsg(IPMParamBlockPtr paramBlock,
                        Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result of the function.
→	`msgRef`	`IPMMsgRef`	Message reference number.
→	`refCon`	`unsigned long`	Reserved for your use.
→	`msgToNest`	`IPMMsgRef`	Message reference number of the message to nest.
←	`startingOffset`	`long`	Offset to the start of the nested message.

See "Interprogram Messaging Parameter Block Header" on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`msgRef` The message reference number of the message to which you want to add a nested message. This number is returned by the `IPMNewMsg` function for a message you intend to send, by the `IPMNewHFSMsg` function for a message you intend to save to disk, and by the `IPMNewNestedMsgBlock` function for a nested message.

`refCon`	An unsigned 4-byte number, reserved for your use. The IPM Manager puts this value into the nested-message header.
`msgToNest`	This parameter contains the message reference number of an existing message that you want to nest within the message that you specify in the `msgRef` field. This number is returned by the `IPMOpenMsg` function for a message you have read, by the `IPMOpenHFSMsg` function for a message you have read from disk, or by the `IPMOpenBlockAsMsg` for a nested message.
`startingOffset`	The offset, in bytes, to the start of the new nested-message block from the start of the body of the enclosing message. You can use this value if you want to create your own table of contents for a message you are creating.

DESCRIPTION

You can call the `IPMNestMsg` function at any time during the message-creation process. The `IPMNestMsg` function adds an existing message as a nested message at the end of the message that you specify in the `msgRef` field. Before you call the `IPMNestMsg` function, you must use the `IPMOpenMsg`, `IPMOpenHFSMsg`, or `IPMOpenBlockAsMsg` function to open the message to be nested.

The `IPMNestMsg` function first creates a new block at the end of the message. The `msgCreator` field of the block type of the new block is equal to the constant `kIPMSignature` and the `msgType` field is equal to `kIPMEnclMsgType`. The function then writes the specified message into the new block.

The `IPMNestMsg` function returns, in the `startingOffset` parameter, the offset to the start of the new block. The function provides this offset for your information only. You should not call `IPMWriteMsg` to make changes to this nested message.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0406

RESULT CODES

noErr	0	No error
kOCEParamErr	–50	Invalid parameter
kIPMInvalidMsgType	–15091	Message type is invalid
kIPMInvalidProcHint	–15092	Process hint is invalid
kIPMMsgTypeReserved	–15095	Message type reserved for system use
kIPMNestedMsgOpened	–15097	Nested message opened; cannot do operation
kIPMA1HdrCorrupt	–15098	Message is corrupt; may not be message
kIPMCorruptDataStructures	–15099	Message is corrupt
kIPMInvalidSender	–15103	Sender is invalid
kIPMStreamErr	–15108	Error on stream
kIPMPortClosed	–15109	Stream closed

SEE ALSO

See "Message and Block Types" on page 7-26 for more information about the IPMMsgType structure.

You can obtain a message reference number from the IPMOpenMsg function (page 7-82), the IPMOpenHFSMsg function (page 7-84), or the IPMOpenBlockAsMsg (page 7-86).

IPMWriteMsg

The IPMWriteMsg function writes data to the specified location within the body of a message.

```
pascal OSErr IPMWriteMsg(IPMParamBlockPtr paramBlock,
                         Boolean async);
```

paramBlock
: A pointer to a parameter block.

async
: A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to true for asynchronous execution.

Parameter block

→	ioCompletion	ProcPtr	Pointer to a completion routine.
←	ioResult	OSErr	Result of the function.
→	msgRef	IPMMsgRef	Message reference number.
→	mode	IPMAccessMode	The mode in which the function interprets the offset value.
→	offset	long	Offset at which to begin writing.
→	count	unsigned long	Number of bytes of data to write.
→	buffer	Ptr	Pointer to the data buffer.
←	actualCount	unsigned long	Number of bytes of data written.
→	currentBlock	Boolean	Set to true to restrict writing to the current block.

See "Interprogram Messaging Parameter Block Header" on page 7-40 for descriptions of the ioCompletion and ioResult fields.

Field descriptions

msgRef — The message reference number of the message or nested message to which you want to write. This number is returned by the IPMNewMsg function for a message you intend to send, by the IPMNewHFSMsg function for a message you intend to save to disk, and by the IPMNewNestedMsgBlock function for a nested message.

mode — The mode in which the offset parameter is to be interpreted. The function uses this field to determine whether to begin writing data at the end of the last data written or to use the offset value to calculate another starting point relative to the beginning of the message, the end of the message, or the current location. See the discussion following these field descriptions for details.

offset — An offset that the function uses when it calculates the starting point of the write operation. See the following discussion for details.

count — The number of bytes of data that you want the function to write from the buffer into the message.

buffer — A pointer to your data buffer.

actualCount — The number of bytes of data the function actually wrote into the message.

currentBlock — A Boolean value that specifies whether you want the entire write operation to occur within the current block. The **current block** is always the last block to be added to the message. If you set this field to true but the values you specify for the mode and offset fields require the function to write data into another block, the function cancels the write operation and returns the kIPMInvalidOffset result code.

DESCRIPTION

The IPM Manager uses a marker (referred to as the **message mark**) that points to the current location within a message that you are creating. After the `IPMNewBlock` function completes, the message mark points to the first byte in the new block. After the `IPMWriteMsg` function completes, the mark points to the end of the last byte written.

Note

The way you use the message mark, mode, and offset to read and write messages is similar to the way you use the file mark, positioning mode, and positioning offset to read and write files. See *Inside Macintosh: Files* for more information about how the File Manager treats these parameters. u

You use the `mode` and `offset` parameters to specify the point in the message at which the `IPMWriteMsg` function starts writing. The `mode` parameter indicates whether you want the `IPMWriteMsg` function to begin writing at the current position of the mark or to calculate another starting point relative to the beginning of the message, the end of the message, or the current mark location. (In the case of a nested message, offsets are relative to the start or end of the nested message, not the enclosing message.) You can set the `mode` parameter to any one of the following values:

```
enum {
   kIPMAtMark,
   kIPMFromStart,
   kIPMFromLEOM,
   kIPMFromMark
};
```

Constant descriptions

`kIPMAtMark` The `IPMWriteMsg` function starts writing at the current position of the mark. In this case, the function ignores the offset value. This mode is useful, for example, for writing data in sequence into a new block.

`kIPMFromStart` If the `currentBlock` parameter is set to `true`, the function interprets the value in the `offset` parameter as an offset from the beginning of the current block. If the `currentBlock` parameter is set to `false`, the function interprets the value in the `offset` parameter as an offset from the beginning of the message body. If you want to start writing at the beginning of the second block in the message, for example, you can set `currentBlock` to `false` and use the offset that the `IPMNewBlock` function returned when you created the second block. When you use this mode, you cannot set the `offset` parameter to a negative value.

`kIPMFromLEOM` The function interprets the value in the `offset` parameter as an offset from the current end of the message.

`kIPMFromMark` The function interprets the value in the `offset` parameter as an offset from the current position of 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.

If the mark is at the end of the last block, the function extends the end of the block and the end of the message as it writes data into the block.

Note

If you use a positive offset to position the mark past the current end of the message, the function extends the end of the message and writes the data in the location you requested. In this case, you incorporate into the message whatever happened to be on disk between the previous end of the message and the location at which you start writing. u

If you set the `currentBlock` parameter to `true`, the `IPMWriteMsg` function returns an error rather than starting to write in a block other than the last block to be added to the message.

Note that the IPM Manager places the offset to each block in the message header when you first create the block. You cannot change this information in the message header after the block is created. Therefore, when you call the `IPMNewBlock` function to create a new block, you freeze the size of the previous block. You can use the `IPMWriteMsg` function to write over data in an existing block, but you cannot change the size of the block. If you write too much data to fit in an existing block, the function writes over the block boundary into the following block.

When you call the `IPMWriteMsg` function, it first calculates the starting position of the write request. The function then checks the value of the `currentBlock` parameter to determine if it is in conflict with the starting position. That is, if you set `currentBlock` to `true` and specify a write location that falls in another block of the message, the `IPMWriteMsg` function returns the `kIPMInvalidOffset` error.

If the `currentBlock` setting is not in conflict with the specified starting position, the function writes the data from the buffer into the message and returns, in the `actualCount` field, the number of bytes written.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0407

RESULT CODES

noErr	0	No error
kOCEParamErr	–50	Invalid parameter
kIPMNotInABlock	–15096	The specified starting point does not fall within the body of the message
kIPMNestedMsgOpened	–15097	The message reference in the msgRef parameter specifies an outer message, but nested message is not yet closed
kIPMStreamErr	–15108	Error on stream
kIPMPortClosed	–15109	Stream closed

SEE ALSO

The IPMNewMsg function (page 7-43), the IPMNewHFSMsg function (page 7-47), and the IPMNewNestedMsgBlock function (page 7-56) all return message reference numbers.

The IPMNewBlock function (page 7-53) and the IPMNewNestedMsgBlock function (page 7-56) return the offset to the start of a new block.

IPMEndMsg

The IPMEndMsg function ends the message-creation process for the message or nested message that you specify. It can also provide a digital signature for the message.

```
pascal OSErr IPMEndMsg(IPMParamBlockPtr paramBlock,
                       Boolean async);
```

paramBlock
: A pointer to a parameter block.

async
: A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to true for asynchronous execution.

Parameter block

→	ioCompletion	ProcPtr	Pointer to a completion routine.
←	ioResult	OSErr	Result of the function.
→	msgRef	IPMMsgRef	Message reference number.
←	msgID	IPMMsgID	Message ID.
→	msgTitle	RString*	Message title.
→	deliveryNotification	IPMNotificationType	Delivery report specifier.
→	priority	IPMPriority	Message priority.
→	cancel	Boolean	Cancel the message?
→	signature	SIGSignaturePtr	Pointer to a digital signature.
→	signatureSize	Size	Size of the digital signature.
→	signatureContext	SIGContextPtr	Pointer to digital signature context.

See “Interprogram Messaging Parameter Block Header” on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`msgRef` The message reference number of the message or nested message that you want to complete. This number is returned by the `IPMNewMsg` function for a message you intend to send, by the `IPMNewHFSMsg` function for a message you intend to save to disk, and by the `IPMNewNestedMsgBlock` function for a nested message.

`msgID` The message ID, a unique identifier assigned to the message by the IPM Manager. You can use this value to identify a message.

`msgTitle` The message title. Because the Finder displays this title for the user for any message in the Out Tray, the message title should reflect the subject, contents, or purpose of the message. The maximum size of this title is 32 bytes (that is, an `RString32` structure).

`deliveryNotification` The types of delivery reports you want to receive. See “Delivery Notification,” beginning on page 7-28, for more information about this value.

`priority` The priority of the message. Set this parameter to any one of the following values: `kIPMNormalPriority`, `kIPMLowPriority`, or `kIPMHighPriority`.

`cancel` A Boolean value that specifies whether to cancel the message. Set this field to `true` to cancel the message or to `false` to send the message. If the `IPMEndMsg` function applies to a nested message, the function ignores the value of this field.

`signature` A pointer to a digital signature. You must allocate a buffer for the signature and pass a pointer to it in this field. If you specify `nil` for the `signatureContext` field, the function ignores the `signature` field. See the following discussion for more information about digital signatures.

`signatureSize` The size of the digital signature. This value is returned by the `SIGSignPrepare` function.

`signatureContext` A pointer to the signature context you obtained from the `SIGNewContext` function and provided to the `SIGSignPrepare` function. Specify `nil` for this pointer if you do not want a digital signature added to the message.

DESCRIPTION

When you call the `IPMEndMsg` function, it checks the setting of the `cancel` parameter to see if you are canceling the message. If so, the function destroys the message. Otherwise, the function completes the message-creation process for the specified message. If the message reference number you specify applies to a nested-message block, the IPM Manager ends the nested-message block and applies any subsequent functions that you call to the enclosing message. The enclosing message can be another nested message or

the top-level message (that is, the message you started with the IPMNewMsg or IPMNewHFSMsg function). To completely finish the message-creation process, you must call the IPMEndMsg function once for each nested message and once for the top-level message.

IMPORTANT

You cannot close an enclosing message until any messages nested within it have been closed. s

Once you have called the IPMEndMsg function to close the top-level message, you cannot make any more changes to the message. If you created the message with the IPMNewHFSMsg function, the IPM Manager saves the message to the disk file you specified when you called the IPMNewHFSMsg function. If you created the message with the IPMNewMsg function, the IPM Manager sends the message to each recipient and generates any requested reports.

The IPM Manager uses the value of the deliveryNotification parameter to determine when to generate report messages and whether to include the original message in any reply messages that are returned by the recipients.

If you want to add a digital signature to the message, you must call the SIGNewContext and SIGSignPrepare functions before you call the IPMEndMsg function. You can then allocate a buffer for the signature, or specify nil for the signature parameter, in which case the Digital Signature Manager allocates the buffer for you on your application heap. The size needed for the buffer is returned by the SIGSignPrepare function. Pass a pointer to the buffer in the signature parameter to the IPMEndMsg function, the size of the buffer in the signatureSize parameter, and a pointer to the signature context (returned by the SIGNewContext function) in the signatureContext parameter.

Note

If you are adding a digital signature to a large message, the IPMEndMsg function can take a long time to complete (up to several minutes on some computers). You should display a dialog box informing the user of this possibility. u

The IPMEndMsg function places the signature in a block with a creator of kIPMSignature and a type of kIPMDigitalSignature. A message can contain only one block of this type, and you must use the IPMEndMsg function to create this block.

Note

The signature context used to create a digital signature has no relationship to the contexts discussed in “Managing Message Queues” starting on page 7-68 and elsewhere in this chapter. u

SPECIAL CONSIDERATIONS

If you want to add a digital signature to the message (that is, you pass a non-nil value for the signatureContext parameter), you must call the IPMEndMsg function synchronously. There must also be at least 8.5 KB of stack space available.

If you pass `nil` for the `signatureContext` parameter, there must be enough space in your application heap to hold the signature.

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0408

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMInvalidOffset`	–15093	Bad offset for read or write operation
`kIPMNestedMsgOpened`	–15097	The message reference in the `msgRef` parameter specifies an outer message, but nested message is not yet closed
`kIPMA1HdrCorrupt`	–15098	Message is corrupt; may not be message
`kIPMCorruptDataStructures`	–15099	Message is corrupt
`kIPMAbortOfNestedMsg`	–15100	Adding nested message was canceled
`kIPMInvalidSender`	–15103	Sender is invalid
`kIPMNoRecipientsYet`	–15104	Require recipient to send
`kIPMStreamErr`	–15108	Error on stream
`kIPMPortClosed`	–15109	Stream closed

SEE ALSO

You start creating a message with the `IPMNewMsg` function (page 7-43) or the `IPMNewHFSMsg` function (page 7-47), and start a nested-message block with the `IPMNewNestedMsgBlock` function (page 7-56).

See “Delivery Notification,” beginning on page 7-28, for more information about the delivery notification flag byte.

Digital signatures and the `SIGNewContext` and `SIGSignPrepare` functions are discussed in the chapter “Digital Signature Manager” in this book.

Managing Message Queues

You can create any number of local input message queues for your own use. This section describes the functions you can use to create input message queues, open queues, enumerate their contents, and close them.

IPMCreateQueue

The `IPMCreateQueue` function creates a physical queue at the specified location.

```
pascal OSErr IPMCreateQueue(IPMParamBlockPtr paramBlock,
                            Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result of the function.
→	`queue`	`OCERecipient*`	Name and location of the new queue.
→	`identity`	`AuthIdentity`	Authentication identity.
→	`owner`	`PackedRecordID*`	Owner of the queue.

See “Interprogram Messaging Parameter Block Header” on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`queue` A pointer to an `OCERecipient` structure that specifies the name and location of the new queue. You must use the queue-name form of the `OCERecipient` structure for this field.

`identity` The authentication identity of the creator of the queue. If you are creating the queue on a server computer, the messaging server uses this identity to verify that you have the privileges necessary to create a queue. Only the administrator of that server can create queues.

The function ignores this field if you specify the local computer as the location of the new queue.

`owner` A pointer to the packed record ID of the owner of the queue. If you are creating a queue on a remote computer, you must specify an owner of the queue in this field. Only the creator of the queue and the owner of the queue can open or delete the queue.

The function ignores this field if you specify the local computer as the location of the new queue.

DESCRIPTION

You can create a new queue at any time. You can create a queue on the local computer or on a server computer.

IMPORTANT

You should use restraint in creating queues because the IPM Manager provides no interface for listing and managing queues. Also, each queue uses memory and disk resources. s

Once you have used the `IPMCreateQueue` function to create a physical queue, you must open one or more virtual queues to list and open the messages in the queue. Use the `IPMOpenQueue` function to open a virtual queue.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0411

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMBadQName`	–15112	Invalid queue name

SEE ALSO

"Message Addressing Structures" on page 7-24 defines the `OCERecipient` structure. The queue-name form of this structure is described in "Queue-Name Format for Attribute Values" on page 7-16.

You must use the `IPMOpenQueue` function (page 7-72) to open a queue before you can open the messages in the queue. You must have an open queue context before you can open a queue; use the `IPMOpenContext` function, described next, to open a context.

IPMOpenContext

The `IPMOpenContext` function creates a new queue context.

```
pascal OSErr IPMOpenContext(IPMParamBlockPtr paramBlock,
                            Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result of the function.
←	`contextRef`	`IPMContextRef`	Context reference number.

See "Interprogram Messaging Parameter Block Header" on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`contextRef` The context reference number for the new context. You must use this number when opening a queue or closing the context.

DESCRIPTION

You must specify a context reference number when you open a virtual queue. You must specify a virtual-queue reference number when you open a message. When you close a context, the IPM Manager closes all of the virtual queues that belong to that context and all of the open messages that belong to those queues. You can create as many contexts as you wish; in any case, you must call the `IPMOpenContext` function at least once to obtain a context reference number before you can open any queues.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0400

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter

SEE ALSO

Use the `IPMOpenQueue` function, discussed next, to open a virtual queue and add it to a context.

Use the `IPMCloseContext` function (page 7-77) to close all the virtual queues and open messages associated with a context.

IPMOpenQueue

The `IPMOpenQueue` function opens the specified queue and associates it with the specified context.

```
pascal OSErr IPMOpenQueue(IPMParamBlockPtr paramBlock,
                          Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result of the function.
→	`contextRef`	`IPMContextRef`	Context reference number.
→	`queue`	`OCERecipient*`	Queue that you want to open.
→	`identity`	`AuthIdentity`	Authentication identity.
→	`filter`	`IPMFilter*`	Pointer to the queue filter.
←	`newQueueRef`	`IPMQueueRef`	Virtual-queue reference number.
→	`notificationProc`	`IPMNoteProcPtr`	Reserved; set to `nil`.

See “Interprogram Messaging Parameter Block Header” on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`contextRef` A context reference number. When you close the context specified by this reference number, the IPM Manager closes all of the virtual queues that you opened using this reference number.

`queue` A pointer to an `OCERecipient` structure that specifies the name and location of the queue that you want to open. To open a user’s default messaging queue, just specify the user record of that user. To open a queue that you created, use the same `OCERecipient` structure that you used to create the queue.

`identity` The authentication identity of the opener of the queue. If the physical queue is on a server computer, only the server administrator and the owner of the physical queue can open a new virtual queue.

`filter` A pointer to the message filter for this virtual queue.

Set this field to `nil` if you do not want the IPM Manager to associate any filter with this queue.

`newQueueRef` The reference number for the queue. You must use this reference number when you change the queue filter or list, open, close, or delete messages.

`notificationProc`
Reserved. You must set this field to `nil`.

DESCRIPTION

The `IPMOpenQueue` function opens the message queue you specify, creating a virtual queue with the message filter you provide. The function returns a reference number that uniquely identifies this virtual queue. When you call this function, you must specify a message-context reference number. The context links together several queues so that you can simultaneously close them simply by closing the context. If you have not already created the message context to which you want this queue to belong, you must call the `IPMOpenContext` function before calling the `IPMOpenQueue` function. You can open the same physical queue any number of times, creating a new virtual queue each time.

You specify a virtual-queue reference number whenever you list or open messages. Once you have opened a message, you must provide the same queue reference number when you call the `IPMCloseMsg` function or the `IPMCloseQueue` function. If you call the `IPMCloseQueue` function, the IPM Manager simultaneously closes all the messages that you opened with that queue reference number. If you call the `IPMCloseContext` function, the IPM Manager simultaneously closes all the messages associated with all the queues that belong to that context, and closes all of those queues.

The message filter determines which messages in the physical queue are listed by the `IPMEnumerateQueue` function when you provide the reference number for this virtual queue, which messages you can open through the queue, and which messages you can close and delete through the queue. For example, you can open a virtual queue for the default input queue with a filter that passes only high-priority messages. Then, when you call the `IPMOpenMsg` function with that queue reference number, the function allows you to open only the high-priority messages in the default input queue. If you do not provide a filter for the queue, these functions operate on all the messages in the physical queue.

SPECIAL CONSIDERATIONS

Although you allocate the pointer to the queue filter, the IPM Manager owns the pointer until you close the queue or call the `IPMChangeQueueFilter` function to replace the filter. Do not reuse or dispose of this pointer until you close the queue or replace the filter.

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0409

RESULT CODES

noErr	0	No error
kOCEParamErr	–50	Invalid parameter
kIPMInvalidFilter	–15105	The specified filter is invalid
kIPMStreamErr	–15108	Error on stream
kIPMPortClosed	–15109	Stream closed
kIPMBadQName	–15112	Invalid queue name
kIPMBadContext	–15118	Invalid context reference
kIPMContextIsClosing	–15119	The specified context is closing

SEE ALSO

To create a new queue before opening it, use the `IPMCreateQueue` function (page 7-69).

You can change the queue filter by calling the `IPMChangeQueueFilter` function, described next. See "Filter Structures" on page 7-34 for information on queue filters.

Call the `IPMCloseQueue` function (page 7-76) to close a virtual queue.

IPMChangeQueueFilter

The `IPMChangeQueueFilter` function sets a new filter for a specific virtual queue.

```
pascal OSErr IPMChangeQueueFilter(IPMParamBlockPtr paramBlock,
                                  Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	ioCompletion	ProcPtr	Pointer to a completion routine.
←	ioResult	OSErr	Result of the function.
→	queueRef	IPMQueueRef	Virtual-queue reference number.
↔	filter	IPMFilter*	Pointer to the queue filter.

See "Interprogram Messaging Parameter Block Header" on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`queueRef` The virtual-queue reference number returned by the `IPMOpenQueue` function. This number identifies the virtual queue to which the request applies.

`filter` A pointer to an `IPMFilter` structure that specifies the new filter that you want the IPM Manager to apply to the queue. Set this field to `nil` to remove all filters from this queue.

When the `IPMChangeQueueFilter` function completes execution, it returns a pointer to the filter that was in effect when you called the function. The IPM Manager has no further use for this pointer, and you can now dispose of it.

DESCRIPTION

The `IPMChangeQueueFilter` function applies the filter specified in the `filter` parameter to the virtual queue indicated by the `queueRef` parameter. If you set the `filter` parameter to `nil`, the function sets the filter for the virtual queue to the default filter, which matches all messages in the physical queue.

SPECIAL CONSIDERATIONS

Although you allocate the pointer to the queue filter, the IPM Manager owns the pointer until you close the queue or call the `IPMChangeQueueFilter` function to replace the filter. Do not reuse or dispose of this pointer until you close the queue or replace the filter.

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0414

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMInvalidMsgType`	–15091	Message type is invalid
`kIPMInvalidFilter`	–15105	Filter is invalid
`kIPMStreamErr`	–15108	Error on stream
`kIPMPortClosed`	–15109	Stream closed

SEE ALSO

See “Filter Structures” on page 7-34 for information on queue filters.

You set the queue filter initially when you open the queue; see the description of the `IPMOpenQueue` function on page 7-72.

IPMCloseQueue

The `IPMCloseQueue` function closes the specified virtual message queue.

```
pascal OSErr IPMCloseQueue(IPMParamBlockPtr paramBlock,
                           Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result of the function.
→	`queueRef`	`IPMQueueRef`	Virtual-queue reference number.

See “Interprogram Messaging Parameter Block Header” on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`queueRef` The virtual-queue reference number returned by the `IPMOpenQueue` function. This number identifies the virtual queue you wish to close.

DESCRIPTION

You can call the `IPMCloseQueue` function at any time that the specified virtual queue is open. When you call this function, the function first closes any messages that you opened using the queue reference number for this queue. The function then closes the virtual queue and disassociates the queue from its context.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$040A

RESULT CODES

noErr	0	No error
kOCEParamErr	–50	Invalid parameter

SEE ALSO

You use the IPMOpenQueue function (page 7-72) to open a virtual queue.

You can use the IPMCloseMsg function (page 7-104) to close an individual message.

You can use the IPMCloseContext function, described next, to close simultaneously all of the queues associated with a specific context.

You can use the IPMDeleteQueue function (page 7-78) to delete a physical queue after you have closed all of its associated virtual queues.

IPMCloseContext

The IPMCloseContext function closes all of the messages and queues that are associated with the specified context and then eliminates that context.

```
pascal OSErr IPMCloseContext(IPMParamBlockPtr paramBlock,
                             Boolean async);
```

paramBlock
A pointer to a parameter block.

async A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to true for asynchronous execution.

Parameter block

→	ioCompletion	ProcPtr	Pointer to a completion routine.
←	ioResult	OSErr	Result of the function.
→	contextRef	IPMQueueRef	Context reference number.

See “Interprogram Messaging Parameter Block Header” on page 7-40 for descriptions of the ioCompletion and ioResult fields.

Field descriptions

contextRef The context reference number returned by the IPMOpenContext function. This number identifies the context you wish to close.

DESCRIPTION

When you open a virtual queue, you provide a context reference number that specifies the context to which that queue belongs. When you close a context, the IPMCloseContext function first closes all of the messages that you opened for the queues that belong to that context. Next, it closes all of the queues that belong to the

context, and finally, it eliminates the context itself, so that the context reference number is no longer valid.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0401

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMBadContext`	–15118	Invalid context reference
`kIPMContextIsClosing`	–15119	The specified context is already closed

SEE ALSO

You use the `IPMOpenContext` function (page 7-70) to create a context.

You use the `IPMOpenQueue` function (page 7-72) to open a queue and associate it with a specific context.

You can use the `IPMCloseMsg` function (page 7-104) to close a specific message and the `IPMCloseQueue` function (page 7-76) to close a specific queue.

IPMDeleteQueue

The `IPMDeleteQueue` function deletes the specified physical message queue.

```
pascal OSErr IPMDeleteQueue(IPMParamBlockPtr paramBlock,
                            Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result of the function.
→	`queue`	`OCERecipient*`	Queue that you want to delete.
→	`identity`	`AuthIdentity`	Authentication identity.
→	`owner`	`PackedRecordID*`	Owner of the queue.

See "Interprogram Messaging Parameter Block Header" on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`queue` A pointer to an `OCERecipient` structure that specifies the name and location of the queue that you want to delete.

`identity` The authentication identity of the owner of the queue or of the server administrator if this queue is on a server computer.

The IPM Manager ignores this field if the queue is on the local computer.

`owner` A pointer to the packed record ID of the owner of the queue. If the queue is on a remote computer, you must specify the owner of the queue in this field.

The IPM Manager ignores this field if the queue is on the local computer.

DESCRIPTION

Before you can delete a physical queue, you must close any open virtual queues associated with that physical queue. You can delete a queue at any time that the queue is not open, provided it is on the local computer or, if it is on a server computer, you have the appropriate access privileges. The AOCE server allows only the server administrator to delete a queue.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0412

RESULT CODES

noErr	0	No error
kOCEParamErr	–50	Invalid parameter
kIPMBadQName	–15112	Invalid queue name
kIPMQBusy	–15126	Queue busy; cannot delete

SEE ALSO

You use the IPMCreateQueue function (page 7-69) to create a physical queue and the IPMOpenQueue function (page 7-72) to open a virtual queue.

You use the IPMCloseQueue function (page 7-76) to close a virtual queue.

Listing and Reading Messages

A queue can contain any number of messages. This section describes the functions you can use to list the messages in a message queue, open a message or a nested-message block, read a message header and message blocks, and close a message.

IPMEnumerateQueue

The IPMEnumerateQueue function returns a list of messages in the specified queue that match the filter criteria that you provide in the function.

```
pascal OSErr IPMEnumerateQueue(IPMParamBlockPtr paramBlock,
                               Boolean async);
```

paramBlock
A pointer to a parameter block.

async A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to true for asynchronous execution.

Parameter block

→	ioCompletion	ProcPtr	Pointer to a completion routine.
←	ioResult	OSErr	Result of the function.
→	queueRef	IPMQueueRef	Queue reference number.
→	startSeqNum	IPMSeqNum	First message to list.
→	getProcHint	Boolean	List process hints?
→	getMsgType	Boolean	List message types?
→	filter	IPMFilter*	Pointer to queue filter.
→	numToGet	unsigned short	Number of messages to list.
←	numGotten	unsigned short	Number of messages listed.
→	enumCount	unsigned long	Buffer size.
→	enumBuffer	Ptr	Pointer to buffer.
←	actEnumCount	unsigned long	Number of bytes returned in buffer.

See “Interprogram Messaging Parameter Block Header” on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`queueRef`	A pointer to an `OCERecipient` structure that specifies the name and location of the virtual queue that you want to enumerate.
`startSeqNum`	The sequence number of the first message for which you want the function to return information. Sequence numbers start with 1.
`getProcHint`	A Boolean value that indicates whether you want the function to include the process hint of each listed message. You can specify a process hint for a message when you call the `IPMNewMsg`, `IPMNewHFSMsg`, or `IPMNewNestedMsgBlock` function to start the message.
`getMsgType`	A Boolean value that indicates whether you want the function to include the message type of each listed message.
`filter`	A pointer to the filter to use for this enumeration of the queue. If you provide a valid pointer to a filter, the function uses it only for this enumeration; the current filter for this virtual queue remains in effect after the function completes execution. (The current filter is the one you specified most recently with the `IPMOpenQueue` or `IPMChangeQueueFilter` function.) Set the `filter` field to `nil` to use the current filter. Set this field to –1 to ignore all filters and list all the messages in the physical queue.
`numToGet`	The number of messages that you want listed.
`numGotten`	The number of messages that the function actually listed in your buffer.
`enumCount`	The size, in bytes, of the buffer you are providing.
`enumBuffer`	A pointer to the buffer that you are providing.
`actEnumCount`	The number of bytes of data that the function wrote to your buffer.

DESCRIPTION

For each message in the physical input queue that matches your filter criteria, the `IPMEnumerateQueue` function places a structure of type `IPMMsgInfo` in your buffer. You must allocate a buffer large enough to hold at least one complete `IPMMsgInfo` structure. The last two fields in this structure, `procHint` and `msgType`, are present only if you specify `true` for the `getProcHint` and `getMsgType` parameters of the `IPMEnumerateQueue` function. Both the `procHint` and `msgType` fields, if present, are packed structures and can be anywhere from 0 to 33 bytes in size.

You can use the `numToGet` parameter to specify the total number of messages you want listed. In the `numGotten` parameter, the function returns the actual number of messages listed and, in the `actEnumCount` parameter, the number of bytes it wrote to your buffer. The function does not return partial `IPMMsgInfo` structures.

The first time you call the `IPMEnumerateQueue` function to list the messages in a queue, specify 1 for the `startSeqNum` parameter. If the function returns information for as many messages as you requested in the `numToGet` parameter or puts as many

IPMMsgInfo structures in your buffer as the buffer will hold, you can assume that the queue holds more messages to be listed. In this case, increment the number in the startSeqNum parameter by the number of messages listed (that is, by the number returned in the numGotten parameter) and call the function again.

Note
Do not call the IPMEnumerateQueue function any more often than necessary; every user connected to a server periodically requests a list of messages, and a server's overall performance can be noticeably affected if it has to process too many enumeration requests. u

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0413

RESULT CODES

noErr	0	No error
kOCEParamErr	–50	Invalid parameter
kIPMInvalidMsgType	–15091	Message type is invalid
kIPMInvalidFilter	–15105	Filter is invalid
kIPMStreamErr	–15108	Error on stream
kIPMPortClosed	–15109	Stream closed
kIPMeoQ	–15120	No more messages

SEE ALSO

The IPMEnumerateQueue function places structures of type IPMMsgInfo in your buffer. The IPMMsgInfo data type is described in "Message Information Structure" on page 7-36.

IPMOpenMsg

The IPMOpenMsg function opens the specified message in the specified queue.

```
pascal OSErr IPMOpenMsg(IPMParamBlockPtr paramBlock,
                        Boolean async);
```

paramBlock
A pointer to a parameter block.

async A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result of the function.
→	`queueRef`	`IPMQueueRef`	Queue reference number.
→	`sequenceNum`	`IPMSeqNum`	Message sequence number requested.
←	`newMsgRef`	`IPMMsgRef`	Message reference number.
←	`actualSeqNum`	`IPMSeqNum`	Sequence number of message actually opened.
→	`exactMatch`	`Boolean`	Match requested sequence number exactly?

See "Interprogram Messaging Parameter Block Header" on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`queueRef` The queue reference number of the virtual queue containing the message that you want to open.

`sequenceNum` The sequence number of the message you wish to open, or, if you set the `exactMatch` field to `false`, the sequence number at which you want the function to start looking for a message to open. Sequence numbers start with 1.

`newMsgRef` The message reference number of the opened message. You must use this number when you call the `IPMVerifySignature` function to verify a signature, when you call the `IPMCloseMsg` function to close the message, and any time you read information from the message.

`actualSeqNum` The actual sequence number of the message opened by the function. This value always equals the number you specify in the `sequenceNum` field unless you set the `exactMatch` field to `false`, in which case the message opened might have a sequence number higher than the one you requested.

`exactMatch` A Boolean value that specifies whether the sequence number of the message opened must be exactly the same as the number you specify in the `sequenceNum` field. If you set the `exactMatch` field to `false`, the function opens the next message that has a sequence number equal to or greater than the one you specify in the `sequenceNum` field and that passes the current filter criteria for the queue.

DESCRIPTION

You must call the `IPMOpenMsg` function before you can read any of the information in a message in a message queue.

The IPM Manager assigns a sequence number to each message in a physical queue when it adds that message to the queue. Because the IPM Manager does not reuse the number

of a message that is removed from the queue, some sequence numbers might be missing from the queue.

The `IPMOpenMsg` function opens a message only if it meets the current filter criteria for the virtual queue. If you specify a message sequence number for a message that does not meet the filter criteria and set the `exactMatch` field to `true`, the `IPMOpenMsg` function returns the `kIPMEltNotFound` result code.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$040B

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMA1HdrCorrupt`	–15098	Message is corrupt; may not be message
`kIPMStreamErr`	–15108	Error on stream
`kIPMPortClosed`	–15109	Stream closed

SEE ALSO

You can use the `IPMEnumerateQueue` function (page 7-80) to list the messages in a queue.

Use the `IPMOpenHFSMsg` function, described next, to open a message on disk.

Use the `IPMOpenBlockAsMsg` function (page 7-86) to open a nested message.

IPMOpenHFSMsg

The `IPMOpenHFSMsg` function opens the specified HFS file as a message.

```
pascal OSErr IPMOpenHFSMsg(IPMParamBlockPtr paramBlock,
                           Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result of the function.
→	`hfsPath`	`FSSpec*`	Specifier of the file to open.
←	`newMsgRef`	`IPMMsgRef`	Message reference number.

See "Interprogram Messaging Parameter Block Header" on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`hfsPath` The file system specification structure for the file you wish to open as a message.

`newMsgRef` The message reference number of the opened message. You must use this number when you read information from the message, when you call the `IPMVerifySignature` function to verify a signature, or when you call the `IPMCloseMsg` function to close the message.

DESCRIPTION

You must call the `IPMOpenHFSMsg` function before you can read any of the information in a message that is in an HFS file on disk.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0417

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMA1HdrCorrupt`	–15098	Message is corrupt; may not be message
`kIPMStreamErr`	–15108	Error on stream
`kIPMPortClosed`	–15109	Stream closed

SEE ALSO

Use the `IPMOpenMsg` function (page 7-82) to open a message in a message queue.

Use the `IPMOpenBlockAsMsg` function, described next, to open a nested message.

IPMOpenBlockAsMsg

The `IPMOpenBlockAsMsg` function opens a nested message.

```
pascal OSErr IPMOpenBlockAsMsg(IPMParamBlockPtr paramBlock,
                               Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result of the function.
→	`msgRef`	`IPMMsgRef`	Message reference number of the enclosing message.
←	`newMsgRef`	`IPMMsgRef`	Message reference number of the nested message.
→	`blockIndex`	`unsigned short`	Index value of block containing nested message.

See "Interprogram Messaging Parameter Block Header," beginning on page 7-40, for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`msgRef` The message reference number of the message that contains the nested message you want to read. This number is returned by the `IPMOpenMsg`, `IPMOpenHFSMsg`, or `IPMOpenBlockAsMsg` function when you open the containing message.

`newMsgRef` The message reference number of the opened nested message. You must use this number when you read information from the message, when you call the `IPMVerifySignature` function to verify a signature, or when you call the `IPMCloseMsg` function to close the message.

`blockIndex` The sequential position of the block that you want to open as a message. For example, if you want to open the tenth block, you set `blockIndex` to 10. You can use the `IPMGetBlkIndex` function to get the index number of a block.

DESCRIPTION

The `IPMOpenBlockAsMsg` function opens a nested message so that you can use other IPM Manager functions to read information from it. Before you use this function, you must open the containing message (which can also be a nested message), and you must know the index number of the nested-message block within the containing message. A

nested message has a creator type of `kIPMSignature` and a block type of `kIPMEnclosedMsgType`.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$040F

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMA1HdrCorrupt`	–15098	Message is corrupt; may not be message
`kIPMBlockIsNotNestedMsg`	–15101	Block is not message
`kIPMStreamErr`	–15108	Error on stream
`kIPMPortClosed`	–15109	Stream closed

SEE ALSO

Use the `IPMGetBlkIndex` function (page 7-96) to get the index number of a block.

IPMGetMsgInfo

The `IPMGetMsgInfo` function returns information about a message in a message queue.

```
pascal OSErr IPMGetMsgInfo(IPMParamBlockPtr paramBlock,
                           Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result of the function.
→	`msgRef`	`IPMMsgRef`	Message reference number.
↔	`info`	`IPMMsgInfo*`	Pointer to returned information.

See "Interprogram Messaging Parameter Block Header," beginning on page 7-40, for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`msgRef` The message reference number of the message about which you want information. This number is returned by the `IPMOpenMsg` function when you open the message.

`info` A pointer to an `IPMMsgInfo` structure in which the function returns information about the message. You must allocate this structure. The function always returns the full `IPMGetMsgInfo` structure, which is of variable length and packed; the maximum size of this structure is 130 bytes.

DESCRIPTION

You can call the `IPMGetMsgInfo` function after you open a message in a queue. You cannot use the `IPMGetMsgInfo` function to obtain information about a message stored in a file on disk or to get information about a nested message.

The `IPMGetMsgInfo` function returns the same information about a message as the `IPMEnumerateQueue` function returns.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0419

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMStreamErr`	–15108	Error on stream
`kIPMPortClosed`	–15109	Stream closed

SEE ALSO

The `IPMGetMsgInfo` function returns the same information about a message as the `IPMEnumerateQueue` function (page 7-80) returns.

The `IPMGetMsgInfo` function returns information in an `IPMGetMsgInfo` structure, described in "Message Information Structure" on page 7-36.

Use the `IPMReadHeader` function, described next, to obtain header information from nested messages and messages stored on disk, or to get information from header fields not returned by the `IPMGetMsgInfo` function.

IPMReadHeader

The `IPMReadHeader` function reads the contents of a specified header field of a message.

```
pascal OSErr IPMReadHeader(IPMParamBlockPtr paramBlock,
                           Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result of the function.
→	`msgRef`	`IPMMsgRef`	Message reference number.
→	`fieldSelector`	`unsigned short`	Message header field selector.
→	`offset`	`long`	Offset to header field.
→	`count`	`unsigned long`	The size, in bytes, of the output buffer.
→	`buffer`	`Ptr`	Pointer to your buffer.
←	`actualCount`	`unsigned long`	Number of bytes of data read.
←	`remaining`	`unsigned long`	Number of bytes of data remaining to be read.

See “Interprogram Messaging Parameter Block Header” on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`msgRef` The message reference number of the message whose header you want to read. This number is returned by the `IPMOpenMsg`, `IPMOpenHFSMsg`, or `IPMOpenBlockAsMsg` function when you open the message.

`fieldSelector` The message-header field or fields that you want to read. You can set the `fieldSelector` field to the values shown in the description section that follows.

`offset`	The offset to the header field at which you want to start reading. Set this field to 0 to start reading a header field at the beginning. If the `IPMReadHeader` function returns a value in the `remaining` field, you can increment the value in the `offset` field by the value returned in the `actualCount` field and call the function again to continue reading from the header field.
`count`	The size, in bytes, of the buffer you provide.
`buffer`	A pointer to your buffer.
`actualCount`	The number of bytes of data actually written to your buffer.
`remaining`	The number of bytes of data in this header field remaining to be read.

DESCRIPTION

The `IPMReadHeader` function returns information about one or more fields of a message header. If the buffer you provide is not large enough to hold all the data you request, the function returns, in the `remaining` parameter, the number of bytes remaining. You can then increment the value in the `offset` parameter by the value in the `actualCount` parameter and call the function again. You must open the message with the `IPMOpenMsg`, `IPMOpenHFSMsg`, or `IPMOpenBlockAsMsg` function before you can call the `IPMReadHeader` function.

Use the `fieldSelector` parameter to indicate the field of the message header that you want to read. You can set this parameter to any of the following values:

```
enum {
   kIPMTOC = 0,
   kIPMSender = 1,
   kIPMProcessHint = 2,
   kIPMMessageTitle = 3,
   kIPMMessageType = 4,
   kIPMFixedInfo = 7
};

typedef Byte IPMHeaderSelector;
```

Constant descriptions

`kIPMTOC`	The message table of contents (TOC). The TOC contains information about each block in the message. The `IPMReadHeader` function returns an array of `IPMTOC` structures, each containing information about one block. The `IPMTOC` structure is described on page 7-37.

`kIPMSender` The sender of the message, in an `IPMSender` structure. If the message is authenticated, the IPM Manager fills in this field from the identity of the originator of the message, and this field provides the authenticated originator of the message. If the message is not authenticated, the creator of the message specifies the contents of this field. The `IPMSender` structure is described on page 7-40. The `IPMFixedHdrInfo` structure (page 7-38) includes an `authenticated` field.

`kIPMProcessHint`
The process hint of the message, which is a Pascal string of up to 32 characters. The value of meaning of the process hint is defined by the creator of the message.

`kIPMMessageTitle`
The message title. This title is specified by the creator of the message and normally indicates the subject, purpose, or content of the message.

`kIPMMessageType`
The message type, in an `IPMMsgType` structure (page 7-28).

`kIPMFixedInfo` A standard subset of the fields in the header, in an `IPMFixedHdrInfo` structure (page 7-38). When you set the `fieldSelector` parameter to `kIPMFixedInfo`, the IPM Manager ignores the `offset` and `count` fields.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$040E

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMInvalidOffset`	–15093	Bad offset for read or write operation
`kIPMA1HdrCorrupt`	–15098	Message is corrupt; may not be message
`kIPMStreamErr`	–15108	Error on stream
`kIPMPortClosed`	–15109	Stream closed

SEE ALSO

The `IPMSender` structure is described in “Sender Structure” on page 7-39.

The `IPMTOC` structure is described on page 7-37.

The `IPMMsgType` structure is described on page 7-28.

The `IPMFixedHdrInfo` structure is described on page 7-38.

IPMReadRecipient

The `IPMReadRecipient` function reads a recipient from a message header.

```
pascal OSErr IPMReadRecipient(IPMParamBlockPtr paramBlock,
                              Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result of the function.
→	`msgRef`	`IPMMsgRef`	Message reference number.
→	`rcptIndex`	`unsigned short`	Recipient index number.
→	`offset`	`long`	Offset to recipient data.
→	`count`	`unsigned long`	Buffer size.
→	`buffer`	`Ptr`	Pointer to your buffer.
←	`actualCount`	`unsigned long`	Number of bytes of data read.
→	`reserved`	`short`	Must be 0.
←	`remaining`	`unsigned long`	Number of bytes of data remaining to be read.
←	`originalIndex`	`unsigned short`	Original recipient index.
←	`OCERecipientOffsetFlags`	`recipientOffsetFlags`	Recipient-type flags.

See "Interprogram Messaging Parameter Block Header" on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`msgRef` The message reference number of the message whose recipient data you want to read. This number is returned by the `IPMOpenMsg`, `IPMOpenHFSMsg`, or `IPMOpenBlockAsMsg` function when you open the message.

`rcptIndex` The index number of the recipient you want to read. Recipient index numbers are sequential, starting with 1.

`offset`	The offset to the data for the specified recipient at which to start reading. The first time you call the `IPMReadRecipient` function for a given recipient you should set this field to 0. If your buffer is not large enough to hold all of the recipient data, you can increment the value in the `offset` field by the value returned in the `actualCount` field and call the function again.
`count`	The size, in bytes, of your buffer.
`buffer`	A pointer to your buffer. The function places the information about the recipient in your buffer in the form of an `OCEPackedRecipient` structure.
`actualCount`	The number of bytes of data the function wrote to your buffer.
`reserved`	Reserved; you must set this field to 0.
`remaining`	The number of bytes of data remaining to be read. If this field returns a nonzero value, you should increment the value in the `offset` field by the value returned in the `actualCount` field and call the function again.
`originalIndex`	The index of this recipient in the original recipient list (that is, the recipient list before the IPM Manager resolves any group addresses).
`OCERecipientOffsetFlags`	A flag byte that provides information about the recipient.

DESCRIPTION

The `IPMReadRecipient` function returns recipient information from the header of a message. If the original message header contained recipient addresses that were groups or that identified records containing the address of the actual recipient, the `IPMReadRecipient` function returns the final recipients of the message.

The `OCERecipientOffsetFlags` field contains the following bits:

```
enum {
   kIPMFromDistListBit = 0,
   kIPMDummyRecBit = 1,
   kIPMFeedbackRecBit = 2,
   kIPMReporterRecBit = 3,
   kIPMBCCRecBit = 4
};
```

Flag descriptions

`kIPMFromDistListBit`	Reserved.
`kIPMDummyRecBit`	If this flag is set to 1, the IPM Manager delivered the message to this recipient.
`kIPMFeedbackRecBit`	Reserved.

`kIPMReporterRecBit`
Reserved.

`kIPMBCCRecBit` If this flag is set to 1, this is a "bcc" (blind carbon copy) recipient; in other words, this recipient is not included in the recipient list received by the other recipients of the message. You can receive this flag only if you sent the letter or if you were the bcc recipient.

You can use the following mask values to test these flags:

```
enum {
   kIPMFromDistListMask=   1<<kIPMFromDistListBit,
   kIPMDummyRecMask=       1<<kIPMDummyRecBit,
   kIPMFeedbackRecMask=    1<<kIPMFeedbackRecBit,
   kIPMReporterRecMask=    1<<kIPMReporterRecBit,
   kIPMBCCRecMask=         1<<kIPMBCCRecBit
};
```

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0410

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMInvalidOffset`	–15093	Bad offset for read or write operation
`kIPMA1HdrCorrupt`	–15098	Message is corrupt; may not be message
`kIPMStreamErr`	–15108	Error on stream
`kIPMPortClosed`	–15109	Stream closed

SEE ALSO

The `IPMReadRecipient` function places the information about the recipient in your buffer in the form of an `OCEPackedRecipient` structure (page 7-25).

IPMReadReplyQueue

The `IPMReadReplyQueue` function reads the reply queue field of the message header.

```
pascal OSErr IPMReadReplyQueue(IPMParamBlockPtr paramBlock,
                               Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result of the function.
→	`msgRef`	`IPMMsgRef`	Message reference number.
→	`offset`	`long`	Offset to reply queue data.
→	`count`	`unsigned long`	Buffer size.
→	`buffer`	`Ptr`	Pointer to your buffer.
←	`actualCount`	`unsigned long`	Number of bytes of data read.
→	`reserved`	`short`	Must be 0.
←	`remaining`	`unsigned long`	Number of bytes of data remaining to be read.

See “Interprogram Messaging Parameter Block Header” on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`msgRef` The message reference number of the message whose reply queue data you want to read. This number is returned by the `IPMOpenMsg`, `IPMOpenHFSMsg`, or `IPMOpenBlockAsMsg` function when you open the message.

`offset` The offset to the data at which to start reading. The first time you call the `IPMReadReplyQueue` function, you should set this value to 0. If your buffer is not large enough to hold all of the reply queue data, you can increment the value in the `offset` field by the value returned in the `actualCount` field and call the function again.

`count` The size, in bytes, of your buffer.

`buffer` A pointer to your buffer. The function places the information about the reply queue in your buffer in the form of an `OCEPackedRecipient` structure.

`actualCount` The number of bytes of data the function wrote to your buffer.

`reserved` Reserved; you must set this field to 0.

`remaining` The number of bytes of data remaining to be read. If this field returns a nonzero value, you should increment the value in the `offset` field by the value returned in the `actualCount` field and call the function again.

DESCRIPTION

The reply queue is the address to which the IPM Manager returns delivery and nondelivery reports and to which you should address reply messages.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0421

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMInvalidOffset`	–15093	Bad offset for read or write operation
`kIPMA1HdrCorrupt`	–15098	Message is corrupt; may not be message
`kIPMAttrNotInHdr`	–15106	No reply queue in message header
`kIPMStreamErr`	–15108	Error on stream
`kIPMPortClosed`	–15109	Stream closed

SEE ALSO

The `IPMReadReplyQueue` function places the information about the reply queue in your buffer in the form of an `OCEPackedRecipient` structure (page 7-25).

IPMGetBlkIndex

The `IPMGetBlkIndex` function returns the block type and index value for the first block encountered that matches the specifications you provide.

```
pascal OSErr IPMGetBlkIndex(IPMParamBlockPtr paramBlock,
                            Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	ioCompletion	ProcPtr	Pointer to a completion routine.
←	ioResult	OSErr	Result of the function.
→	msgRef	IPMMsgRef	Message reference number.
→	blockType	IPMBlockType	Block types to return.
→	index	unsigned short	Number of matches to find before returning information.
→	startingFrom	unsigned short	Starting index.
←	actualBlockType	IPMBlockType	Block type of block returned.
←	actualBlockIndex	unsigned short	Index value of block returned.

See "Interprogram Messaging Parameter Block Header" on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`msgRef` The message reference number of the message from which you want information. This number is returned by the `IPMOpenMsg`, `IPMOpenHFSMsg`, or `IPMOpenBlockAsMsg` function when you open the message.

`blockType` The creator and type of the block for which you want an index value. You can use the `kIPMTypeWildCard` wildcard value for the creator field, the type field, or both.

`index` The number of matches the function should find before it returns the index and type of a block. For example, if you set the `index` field to 5, the function returns the index and type of the fifth block it finds that matches the value you specify in the `blockType` field.

`startingFrom` The index number of the block at which to begin the search. Index numbers start at 1.

`actualBlockType` The creator and type of the block that matches all of your search criteria.

`actualBlockIndex` The index number of the block that matches all of your search criteria.

DESCRIPTION

Each IPM message can contain message blocks. You can use the `IPMGetBlkIndex` function to determine the type and creator of each block, or the sequential position (referred to as the *index number*) of blocks that have specific types.

If you want to get information about every block in the message, you can specify the wildcard value `kIPMTypeWildCard` for the creator and type and call the function repeatedly, incrementing the value in the `startingFrom` field each time. If you want to get information about every block of a specific type or with a specific creator, put that type or creator in the `blockType` field and call the function repeatedly, incrementing the value in the `index` field each time.

If the function does not find any more matches to your criteria, it returns the `kOCEInvalidIndex` result code.

You can use the value returned in the `actualBlockIndex` field to identify a block you want to read when you call the `IPMReadMsg` function.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0418

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMA1HdrCorrupt`	–15098	Message is corrupt; may not be message
`kIPMBlkNotFound`	–15107	Specified block nonexistent
`kIPMStreamErr`	–15108	Error on stream
`kIPMPortClosed`	–15109	Stream closed

SEE ALSO

To read a message block, call the `IPMReadMsg` function, described next.

IPMReadMsg

The `IPMReadMsg` function reads data from an IPM message.

```
pascal OSErr IPMReadMsg(IPMParamBlockPtr paramBlock,
                        Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	ioCompletion	ProcPtr	Pointer to a completion routine.
←	ioResult	OSErr	Result of the function.
→	msgRef	IPMMsgRef	Message reference number.
→	mode	IPMAccessMode	Mode in which the offset should be interpreted.
→	offset	long	Offset to the starting point of the read.
→	count	unsigned long	Buffer size.
→	buffer	Ptr	Pointer to your buffer.
←	actualCount	unsigned long	Number of bytes of data read.
→	blockIndex	unsigned short	Index number of the block to read.
←	remaining	unsigned long	Number of bytes of data remaining to be read.

See "Interprogram Messaging Parameter Block Header" on page 7-40 for descriptions of the ioCompletion and ioResult fields.

Field descriptions

msgRef	The message reference number of the message you want to read. This number is returned by the IPMOpenMsg, IPMOpenHFSMsg, or IPMOpenBlockAsMsg function when you open the message.
mode	The mode in which the offset parameter is to be interpreted. The function uses this field to determine whether to begin reading data relative to the end of the last data read, to the beginning of the block, or to the end of the block. See the discussion following these field descriptions for details.
offset	An offset that the function uses when it calculates the starting point of the read operation. Set this value to 0 when you start reading a block from the beginning. See the following discussion for details.
count	The size, in bytes, of the buffer that you are providing.
buffer	A pointer to your buffer.
actualCount	The number of bytes of data actually written to your buffer.
blockIndex	The sequential position of the block that you want to read. For example, if you want to read the tenth block, you set blockIndex to 10. You can use the IPMGetBlkIndex function to get the creator, block type, and index number of a block. If you set the blockIndex field to 0, the IPMReadMsg function treats all the blocks in the message, including the message header, as a single unit, ignoring all block boundaries.
remaining	The number of bytes of data remaining to be read. If this field returns a nonzero value, you can increment the value in the offset field by the value in the actualCount field and call the IPMReadMsg function again to read the next portion of data.

DESCRIPTION

The IPMReadMsg function can treat the entire message body as a single unit (if you set the blockIndex parameter to 0) or can read a specific message block.

The IPM Manager uses a marker (referred to as the *message mark*) that points to the current location within a message that you are reading. After the IPMReadMsg function completes, the message mark points to the byte following the last byte read.

You use the mode and offset parameters to specify the point in the message at which the IPMReadMsg function starts reading. The mode parameter indicates whether you want the IPMReadMsg function to begin reading at the current position of the mark or to calculate another starting point relative to the beginning of the message, the beginning of the block, the end of the message, or the current mark location. You can set the mode parameter to any one of the following values:

```
enum {
   kIPMAtMark,
   kIPMFromStart,
   kIPMFromLEOM,
   kIPMFromMark
};
```

Constant descriptions

kIPMAtMark — The IPMReadMsg function starts reading at the current position of the mark. In this case, the function ignores the offset value. This mode is useful, for example, for reading in sequence through a block.

kIPMFromStart — The function interprets the value in the offset parameter as an offset from the beginning of the block you specify by the blockIndex parameter. If you specify 0 for the blockIndex parameter, the function interprets the value in the offset parameter as an offset from the beginning of the message body.

If you want to start reading at the 100th byte of the second block in the message, for example, set the blockIndex parameter to 2, the mode parameter to kIPMFromStart, and the offset parameter to 100. When you use this mode, you cannot set the offset parameter to a negative value or you will be reading data that is not part of the message.

kIPMFromLEOM — The function interprets the value in the offset parameter as an offset from the end of the block you specify by the blockIndex parameter. If you specify 0 for the blockIndex parameter, the function interprets the value in the offset parameter as an offset from the end of the message. When you use this mode, the offset parameter must be a negative value or you will be reading data that is not part of the message.

`kIPMFromMark` The function interprets the value in the `offset` parameter as an offset from the current position of 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.

A message block that has a creator type of `kIPMSignature` and a block type of `kIPMEnclosedMsgType` contains a nested message. To read the contents of such a block, first use the `IPMOpenBlockAsMsg` function to open the nested message and then use the other functions in this section to read its contents.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$040D

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMInvalidOffset`	–15093	Bad offset for read or write operation
`kIPMA1HdrCorrupt`	–15098	Message is corrupt; may not be message
`kIPMCorruptDataStructures`	–15099	Message is corrupt
`kIPMStreamErr`	–15108	Error on stream
`kIPMPortClosed`	–15109	Stream closed

SEE ALSO

Use the `IPMGetBlkIndex` function (page 7-96) to get the creator, block type, and index number of a block.

Use the `IPMOpenBlockAsMsg` function (page 7-86) to read a block containing a nested message.

IPMVerifySignature

The `IPMVerifySignature` function verifies a digital signature for a message.

```
pascal OSErr IPMVerifySignature(IPMParamBlockPtr paramBlock);
```

`paramBlock`
A pointer to a parameter block.

Parameter block

←	`ioResult`	`OSErr`	Result of the function.
→	`msgRef`	`IPMMsgRef`	Message reference number.
→	`signatureContext`	`SIGContextPtr`	Signature context.

See "Interprogram Messaging Parameter Block Header" on page 7-40 for a description of the `ioResult` field.

Field descriptions

`msgRef` The message reference number of the message from which you want information. This number is returned by the `IPMOpenMsg`, `IPMOpenHFSMsg`, or `IPMOpenBlockAsMsg` function when you open the message.

`signatureContext`
The signature context you obtained from the `SIGNewContext` function and provided to the `SIGVerifyPrepare` function.

DESCRIPTION

If the creator of the message used the `IPMEndMsg` function to add a digital signature to the message, you can use the `IPMVerifySignature` function to verify the signature. You can use the `IPMReadHeader` function to determine whether a message has a digital signature. The `IPMEndMsg` function places the digital signature in a block with a creator of `kIPMSignature` and a type of `kIPMDigitalSignature`.

To verify a signature, use the `IPMGetBlkIndex` function to get the index number of the signature block and the `IPMReadMsg` function to read the signature into a buffer. Then call the `SIGNewContext` and `SIGVerifyPrepare` functions to begin the process of verifying the signature. When you pass a pointer to the signature context (returned by the `SIGNewContext` function) in the `signatureContext` parameter to the `IPMVerifySignature` function, the function verifies the signature.

Note
The signature context used to create a digital signature has no relationship to the contexts discussed in "Managing Message Queues" starting on page 7-68 and elsewhere in this chapter. u

Because the IPM Manager modifies some fields in the message header during message transmission and delivery, not all header fields can be signed. For example, the final number of recipients, resolution count, and hop count fields are not signed. All message blocks except the signature block itself are signed.

SPECIAL CONSIDERATIONS

You cannot execute the `IPMVerifySignature` function asynchronously; therefore, you can not call this function at interrupt time.

There must also be at least 8.5 KB of stack space available when you call this function.

If you are verifying a digital signature for a large message, the `IPMVerifySignature` function can take a long time to complete (up to several minutes on some computers). You should display a dialog box informing the user of this possibility.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0422

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMA1HdrCorrupt`	–15098	Message is corrupt; may not be message
`kIPMBlkNotFound`	–15107	Specified block nonexistent
`kIPMStreamErr`	–15108	Error on stream
`kIPMPortClosed`	–15109	Stream closed

SEE ALSO

You use the `IPMEndMsg` function (page 7-65) to add a digital signature to a message.

You can use the `IPMReadHeader` function (page 7-89) to determine if a message has been signed.

Use the `IPMGetBlkIndex` function (page 7-96) to get the index number of the signature block and the `IPMReadMsg` function (page 7-98) to read the signature block.

Digital signatures and the `SIGNewContext` and `SIGVerifyPrepare` functions are discussed in the chapter “Digital Signature Manager” in this book.

IPMCloseMsg

The `IPMCloseMsg` function closes a message, invalidating the message reference number, and can delete the message.

```
pascal OSErr IPMCloseMsg(IPMParamBlockPtr paramBlock,
                         Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result of the function.
→	`msgRef`	`IPMMsgRef`	Message reference number.
→	`deleteMsg`	`Boolean`	Delete the message?

See "Interprogram Messaging Parameter Block Header" on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`msgRef` The message reference number of the message you want to close. This number is returned by the `IPMOpenMsg`, `IPMOpenHFSMsg`, or `IPMOpenBlockAsMsg` function when you open the message.

`deleteMsg` A Boolean value specifying whether you want to delete the message after closing it. If you set this field to `true` for a message in a message queue, the IPM Manager removes the message from the physical queue. If you set this field to `true` for a message that is an HFS file, the IPM Manager deletes the file. If the message is a nested message, the `IPMCloseMsg` function ignores this field.

DESCRIPTION

When you have finished reading information from a message, you should call the `IPMCloseMsg` function so that the IPM Manager can release the memory it allocates when you open a message. You can set the `deleteMsg` parameter to `true` to have the IPM Manager delete the message after it closes it. (The `IPMCloseMsg` function will always close a message that was opened through the queue you specify with the message reference number, but if the same message is also open through another virtual queue, the function does not delete it. In that case, the function returns the `kIPMEltBusy` result code.) If you do not delete the message, it remains in the message queue or on disk and you can open it again at any time.

After you close a message, its message reference number is no longer valid.

You can close a message containing an open nested message; however, you can't delete such a message.

When you call the `IPMCloseQueue` function to close a message queue, the function automatically closes all of the messages that you opened through that queue's reference number. When you call the `IPMCloseContext` function to close a context, it first closes all of the messages that you opened for the queues that belong to that context.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$040C

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMEltBusy`	–15116	Message is in use

SEE ALSO

You use the `IPMOpenMsg` (page 7-82), `IPMOpenHFSMsg` (page 7-84), or `IPMOpenBlockAsMsg` (page 7-86) function to open a message.

The `IPMCloseQueue` function (page 7-76) closes all the messages associated with a specific virtual queue. The `IPMCloseContext` function (page 7-77) closes all the messages associated with a context.

You can use the `IPMDeleteMsgRange` function (page 7-106) to delete one or more messages in a specific virtual queue.

Deleting Messages

You can use the `IPMDeleteMsgRange` function, described in this section, to delete one or more messages in a message queue. The `IPMCloseMsg` function (page 7-104) can delete a single message after closing it.

IPMDeleteMsgRange

The `IPMDeleteMsgRange` function deletes one or more messages from a message queue.

```
pascal OSErr IPMDeleteMsgRange(IPMParamBlockPtr paramBlock,
                               Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A Boolean value that specifies whether the IPM Manager should execute the function asynchronously. Set this parameter to `true` for asynchronous execution.

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result of the function.
→	`queueRef`	`IPMQueueRef`	Queue reference number.
→	`startSeqNum`	`IPMSeqNum`	The starting message sequence number.
→	`endSeqNum`	`IPMSeqNum`	The ending message sequence number.
←	`lastSeqNum`	`IPMSeqNum`	The sequence number of the next message.

See "Interprogram Messaging Parameter Block Header" on page 7-40 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`queueRef` The virtual-queue reference number returned by the `IPMOpenQueue` function. This number identifies the virtual queue to which the request applies.

`startSeqNum` The sequence number of the first message that you want the function to delete.

`endSeqNum` The sequence number of the last message that you want the function to delete.

`lastSeqNum` The sequence number of the next message that remains in the queue following the last deleted message.

If the function is unable to delete all of the requested messages, this field contains the sequence number of the message that the function was attempting to delete when the error occurred.

DESCRIPTION

The `IPMDeleteMsgRange` function deletes one or more messages from the physical message queue. To be deleted, a message must match the current filter for the virtual queue you specify with the `queueRef` parameter and have a sequence number falling within the range you specify with the `startSeqNum` and `endSeqNum` parameters. Note that the sequence numbers are inclusive; for instance, if you set the `startSeqNum`

parameter to 5 and the `endSeqNum` parameter to 10, messages with sequence numbers 5 and 10 (if present in the specified virtual queue) are both deleted.

If the function cannot delete a particular message for some reason, the IPM Manager cancels the function without proceeding any further. In this case, the function returns the sequence number of the message that it was attempting to delete when the error occurred and also returns a result code that indicates the error. The `IPMDeleteMsgRange` function does not delete a message if it is open through any virtual queue. If you have closed the message through the virtual queue but still receive the `kIPMEltBusy` result code, the message might be open through another virtual queue. If the message is closed but contains a nested message that is still open, the function does not delete the message and returns the `kIPMEltBusy` result code.

Once you have deleted a message, you cannot open it again.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. If you call it asynchronously, you can call it at interrupt time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0415

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kIPMStreamErr`	–15108	Error on stream
`kIPMPortClosed`	–15109	Stream closed
`kIPMEltBusy`	–15116	Message is in use

SEE ALSO

You can use the `IPMCloseMsg` function (page 7-104) to delete a single message from a queue.

Utility Functions

You can use the routines in this section to work with `OCERecipient` structures.

The functions described in this section use a different assembly-language calling sequence from the other IPM Manager routines (see page 7-43). Listing 7-2 illustrates one way to do this.

Listing 7-2 Calling an MSAM utility function from assembly language

```
_oceMessaging     OPWORD   $AA5C
   SUBQ     #2,A7                  ; make room for function result
   MOVEA    param1,-(SP)           ; push the first parameter onto stack
...                                ; push additional parameters onto stack
   MOVEQ    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)               ; push the op code onto stack
   _oceMessaging                   ; trap call
   MOVE.W   (SP)+, D0              ; get result code
```

OCESizePackedRecipient

The `OCESizePackedRecipient` function computes the number of bytes of memory needed to hold a packed `OCERecipient` structure.

```
pascal unsigned short OCESizePackedRecipient(
                                          const OCERecipient *rcpt);
```

`rcpt` A pointer to an `OCERecipient` structure whose size, when packed, you want to determine.

DESCRIPTION

The `OCESizePackedRecipient` function computes the number of bytes required to hold the information contained in an `OCERecipient` structure when it is packed. The number of bytes returned by the `OCESizePackedRecipient` function includes the `dataLength` field of the `OCEPackedRecipient` structure.

SPECIAL CONSIDERATIONS

The `OCESizePackedRecipient` function does not pad the value contained in the `extensionSize` field of the `OCERecipient` structure pointed to by the `rcpt` parameter. For this reason, the `OCESizePackedRecipient` function might return an odd value rather than an even one. Therefore, you need to pad the necessary fields in the `OCERecipient` structure yourself before using it as an address for a message or before passing it to any of the IPM Manager functions that require an `OCERecipient` structure of even size.

This function does not purge or move memory.

ASSEMBLY-LANGUAGE INFORMATION

Trap	Selector
_OCEMessaging	$033E

SEE ALSO

The OCERecipient structure is defined on page 7-24.

The OCEPackedRecipient structure is defined on page 7-25.

To pack an OCERecipient structure, use the OCEPackRecipient function, described next.

OCEPackRecipient

The OCEPackRecipient function forms an OCEPackedRecipient structure from an OCERecipient structure.

```
pascal unsigned short OCEPackRecipient(const OCERecipient *rcpt,
                                       void* buffer);
```

rcpt	A pointer to the OCERecipient structure you want to pack.
buffer	A pointer to the buffer in which the packed data is placed by the OCEPackRecipient function. You must allocate this structure.

DESCRIPTION

The OCEPackRecipient function packs the contents of an OCERecipient structure into an OCEPackedRecipient structure. The OCEPackedRecipient structure must be large enough to contain the packed RecordID information and any extension value of the OCERecipient structure. You obtain the buffer size needed by calling the OCESizePackedRecipient function (page 7-108).

SPECIAL CONSIDERATIONS

This function does not purge or move memory.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_OCEMessaging	$033F

SEE ALSO

The OCERecipient structure is defined on page 7-24.

The OCEPackedRecipient structure is defined on page 7-25.

For information on unpacking an OCEPackedRecipient structure, see the OCEUnpackRecipient function, described next.

OCEUnpackRecipient

The OCEUnpackRecipient function unpacks an OCEPackedRecipient structure.

```
pascal OSErr OCEUnpackRecipient(const void* buffer,
                                OCERecipient *rcpt,
                                RecordID *entitySpecifier);
```

buffer A pointer to the OCEPackedRecipient structure you want to unpack.

rcpt A pointer to an OCERecipient structure. You must allocate this structure.

entitySpecifier
A pointer to a RecordID structure. The OCEUnpackRecipient function extracts the record identifier information from the OCEPackedRecipient structure and places it in this RecordID structure. You must allocate this structure.

DESCRIPTION

The OCEUnpackRecipient function extracts the information from an OCEPackedRecipient structure and places it in an OCERecipient structure and a RecordID structure. The OCEUnpackRecipient function extracts the record identifier (if any) and places it in the RecordID structure, places the rest of the information in the OCERecipient structure, and then sets the entitySpecifier field of the OCERecipient structure to point to the RecordID structure. The OCEUnpackRecipient function returns, in the extensionValue field of the OCERecipient structure, a pointer to the extension (if any), and returns the length of that extension in the extensionSize field of the OCERecipient structure. If there is no extension, the OCEUnpackRecipient function sets the extensionValue field of the OCERecipient structure to nil.

SPECIAL CONSIDERATIONS

This function does not move or purge memory.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_OCEMessaging	$0340

RESULT CODES

noErr	0	No error
kOCEParamErr	–50	Invalid parameter

SEE ALSO

The OCERecipient structure is defined on page 7-24.

The OCEPackedRecipient structure is defined on page 7-25.

To pack an OCERecipient structure, see the OCEPackRecipient function (page 7-109).

OCEStreamRecipient

The OCEStreamRecipient function converts an OCERecipient structure from a pointer-based structure into a stream of bytes.

```
pascal OSErr OCEStreamRecipient(const OCERecipient* rcpt,
                                OCERecipientStreamer stream,
                                long userData,
                                unsigned long* actualCount);
```

rcpt — A pointer to the OCERecipient structure you want to process.

stream — A pointer to a stream function that you supply.

userData — Data supplied by you that is passed to your stream function. The userData parameter can contain anything your particular stream method needs.

actualCount — A pointer to the total number of bytes (streamed out) by the OCEStreamRecipient function.

DESCRIPTION

The OCEStreamRecipient function converts an OCERecipient structure into a stream of bytes by calling the stream function that you provide. You can use this function anytime that you want to write the contents of an OCERecipient structure as a series of bytes to a file, into a buffer in memory, or any other place.

The stream function that you provide contains the specific code that writes out the data. The `OCEStreamRecipient` function calls your recipient stream function repeatedly and passes your function the current portion of the data that needs to be streamed, the length of this data, an `eof` flag that is set by the `OCEStreamRecipient` function if this is the last of the data to be streamed, and a `userData` parameter containing any application-specific data that you define. For example, if you were writing a stream function that wrote out an `OCERecipient` structure to a file on a hard disk, you might want to store a pointer in the `userData` parameter to a block of data that contains such information as the filename and size of the file.

If your stream function sends the `OCEStreamRecipient` function an error in the valid range for AOCE error codes, `OCEStreamRecipient` halts execution and returns the error as its result code.

SPECIAL CONSIDERATIONS

This function does not move or purge memory. However, it calls the recipient stream function that you supply in the `stream` parameter, and the stream function could move memory.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_OCEMessaging`	$0341

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter

SEE ALSO

The `OCERecipient` structure is defined on page 7-24.

OCESetRecipientType

Given a creation ID, the `OCESetRecipientType` function sets the extension type of an `OCERecipient` structure.

```
pascal void OCESetRecipientType(OSType extensionType,
                                CreationID *cid);
```

`extensionType`
The type you wish to specify in an `OCERecipient` structure's `extensionType` field.

`cid` A pointer to a `CreationID` structure identifying a record. The `OCERecipient` structure for that record is the one modified by this function.

DESCRIPTION

The `OCESetRecipientType` function sets an `OCERecipient` structure's `extensionType` field to the value in the `extensionType` parameter. The `OCERecipient` is determined from the specified `cid` parameter.

If the `extensionType` field has a value of `'entn'`, then the `cid` parameter is assumed to be a valid extension and is not modified. If the `extensionType` field's value is anything else besides `'entn'`, then this routine sets the `CreationID` structure's `source` field to 0.

SPECIAL CONSIDERATIONS

The `OCESetRecipientType` function does not check whether the `cid` pointer is set to `nil`. Calling this function with the `cid` parameter set to `nil` has an indeterminate but harmful result.

This function does not move or purge memory.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_OCEMessaging`	$0343

SEE ALSO

The `OCERecipient` structure is defined on page 7-24.

To get the extension type of an `OCERecipient` structure, see the `OCEGetRecipientType` function, described next.

OCEGetRecipientType

Given a creation ID, the `OCEGetRecipientType` function returns the extension type of an `OCERecipient` structure.

```
pascal OSType OCEGetRecipientType(const CreationID *cid);
```

`cid` A pointer to a `CreationID` structure identifying a record. The `OCERecipient` structure for that record is the one read by this function.

DESCRIPTION

If you used the `OCESetRecipientType` function (page 7-112) to set the extension type of an `OCERecipient` structure, you can use the `OCEGetExtensionType` function to read the extension type.

SPECIAL CONSIDERATIONS

This function does not purge or move memory.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_OCEMessaging`	$0342

SEE ALSO

The `OCERecipient` structure is defined on page 7-24.

The `CreationID` structure is defined in the chapter "AOCE Utilities" in this book.

To set the extension type of an `OCERecipient` structure, see the `OCESetRecipientType` function (page 7-112).

Application-Defined Functions

This section describes routines that you can provide to be called by the IPM Manager in specific circumstances. The `MyCompletionRoutine` function is a completion routine called when an IPM Manager routine that you call asynchronously completes execution. The `MyRecipientStreamer` function is a stream-processing function that you supply to the `OCEStreamRecipient` function.

MyCompletionRoutine

When you call an IPM Manager function asynchronously, you can provide a pointer to a completion routine.

```
pascal void MyCompletionRoutine (Ptr paramBlk);
```

`paramBlk` A pointer to the parameter block you used when you called the IPM Manager function.

Parameter block

→ `ioResult` `OSErr` Result of the function.

Other fields returned depend on the function that called the completion routine; see the other function descriptions in this chapter for details.

DESCRIPTION

When the IPM Manager function you called asynchronously completes execution, it calls your completion routine. Your completion routine can check the function result plus any parameters returned by the function and take appropriate action.

SPECIAL CONSIDERATIONS

The IPM Manager saves the value of your A5 register at the time you call the function and then restores the A5 value before calling your completion routine.

ASSEMBLY-LANGUAGE INFORMATION

The A0 register contains a pointer to the parameter block. You can look for the result code either in the `ioResult` field of the parameter block or in the D0 register.

MyRecipientStreamer

Your recipient stream function provides a method for processing data from the `OCEStreamRecipient` function.

```
pascal OSErr MyRecipientStreamer(void* buffer,
                        unsigned long count, Boolean eof,
                        long userData);
```

`buffer` A pointer to the data that your stream method processes. This is supplied by the `OCEStreamRecipient` function each time it calls your recipient stream function.

`count` The length, in bytes, of the current data in the buffer.

`eof` A flag that the `OCEStreamRecipient` function sets when it last calls your recipient stream function. This flag signals that the `OCEStreamRecipient` function has finished processing the `OCERecipient` structure.

`userData` The data that you supply in the `userData` parameter to the `OCEStreamRecipient` function. This data is passed directly to your recipient stream function.

DESCRIPTION

The `OCEStreamRecipient` function (page 7-111) calls your recipient stream function to process the data from an `OCERecipient` structure in discrete segments. You write this routine to process the data in the way that you want. The `OCEStreamRecipient` function calls your recipient stream function various times and passes your function progress information as well as the current portion of the `OCERecipient` to process. Any errors returned by this function are passed to the `OCEStreamRecipient` function.

SEE ALSO

The `OCERecipient` data structure is defined on page 7-24.

The `OCEStreamRecipient` function is described on page 7-111.

Summary of the IPM Manager

C Summary

Constants and Data Types

```
/* values of IPMPriority */
enum {
   kIPMAnyPriority = 0,
   kIPMNormalPriority = 1,
   kIPMLowPriority,
   kIPMHighPriority
};

typedef Byte IPMPriority;

/* values of IPMAccessMode */
enum {
   kIPMAtMark,
   kIPMFromStart,
   kIPMFromLEOM,
   kIPMFromMark
};

typedef unsigned short IPMAccessMode;

enum {
   kIPMUpdateMsgBit = 4,
   kIPMNewMsgBit = 5,
   kIPMDeleteMsgBit = 6
};

/* values of IPMNotificationType */
enum {
   kIPMUpdateMsgMask =  1<<kIPMUpdateMsgBit,
   kIPMNewMsgMask =     1<<kIPMNewMsgBit,
   kIPMDeleteMsgMask =  1<<kIPMDeleteMsgBit
```

```
};

typedef Byte IPMNotificationType;

/* values of IPMSenderTag */
enum {
   kIPMSenderRStringTag,
   kIPMSenderRecordIDTag
};

typedef unsigned short IPMSenderTag;

enum {
   kIPMFromDistListBit = 0,
   kIPMDummyRecBit = 1,
   kIPMFeedbackRecBit = 2,
   kIPMReporterRecBit = 3,
   kIPMBCCRecBit = 4
};

/* values of OCERecipientOffsetFlags */
enum {
   kIPMFromDistListMask =  1<<kIPMFromDistListBit,
   kIPMDummyRecMask =      1<<kIPMDummyRecBit,
   kIPMFeedbackRecMask =   1<<kIPMFeedbackRecBit,
   kIPMReporterRecMask =   1<<kIPMReporterRecBit,
   kIPMBCCRecMask =        1<<kIPMBCCRecBit
};

typedef Byte OCERecipientOffsetFlags;

#define kIPMTypeWildCard         'ipmw'

#define kIPMFamilyUnspecified    0
#define kIPMFamilyWildCard       0x3F3F3F3FL    /* '????' */

/* well-known signature */
#define kIPMSignature            'ipms'   /* base type */

/* well-known message types */
#define kIPMReportNotify         'rptn'   /* routing feedback */

/* well-known message block types */
```

```
#define kIPMEnclosedMsgType      'emsg'   /* enclosed (nested) message */
#define kIPMReportInfo           'rpti'   /* recipient information */
#define kIPMDigitalSignature     'dsig'   /* digital signature */

/* values of IPMMsgFormat */
enum {
   kIPMOSFormatType = 1,
   kIPMStringFormatType = 2
};

typedef unsigned short IPMMsgFormat;

/*
Following are the known extension values for IPM addresses handled by Apple
Computer, Inc.
*/

enum {
   kOCEalanXtn= 'alan',
   kOCEentnXtn= 'entn', /* 'entn' = entity name (aka DSSpec) */
   kOCEaphnXtn= 'aphn'
};

/* 'entn' extension forms */
enum {
   kOCEAddrXtn= 'addr', /* reserved */
   kOCEQnamXtn= 'qnam', /* queue-name form */
   kOCEAttrXtn= 'attr', /* an attribute specification */
   kOCESpAtXtn= 'spat'  /* specific attribute */
};

/* phoneNumber subtype constants */
enum {
   kOCEUseHandyDial = 1,
   kOCEDontUseHandyDial = 2
};

/* addresses with kIPMNBPXtn should specify this nbp type */
#define kIPMWSReceiverNBPType "\pMsgReceiver"

/* values of IPMHeaderSelector */
enum {
   kIPMTOC = 0,
   kIPMSender = 1,
   kIPMProcessHint = 2,
```

```
   kIPMMessageTitle = 3,
   kIPMMessageType = 4,
   kIPMFixedInfo = 7
};

typedef Byte IPMHeaderSelector;

enum {
   kIPMDeliveryNotificationBit      = 0,
   kIPMNonDeliveryNotificationBit   = 1,
   kIPMEncloseOriginalBit           = 2,
   kIPMSummaryReportBit             = 3,
   kIPMOriginalOnlyOnErrorBit       = 4
};

typedef Byte IPMNotificationType;

enum {
   kIPMNoNotificationMask           = 0x00,
   kIPMDeliveryNotificationMask     = 1<<kIPMDeliveryNotificationBit,
   kIPMNonDeliveryNotificationMask  = 1<<kIPMNonDeliveryNotificationBit,
   kIPMDontEncloseOriginalMask      = 0x00,
   kIPMEncloseOriginalMask          = 1<<kIPMEncloseOriginalBit,
   kIPMImmediateReportMask          = 0x00,
   kIPMSummaryReportMask            = 1<<kIPMSummaryReportBit,
   kIPMOriginalOnlyOnErrorMask      = 1<<kIPMOriginalOnlyOnErrorBit,
   kIPMEncloseOriginalOnErrorMask   =
                        (kIPMOriginalOnlyOnErrorMask|kIPMEncloseOriginalMask)
};

/* standard nondelivery codes */
enum {
   kIPMNoSuchRecipient              =  0x0001,
   kIPMRecipientMalformed           =  0x0002,
   kIPMRecipientAmbiguous           =  0x0003,
   kIPMRecipientAccessDenied        =  0x0004,
   kIPMGroupExpansionProblem        =  0x0005,
   kIPMMsgUnreadable                =  0x0006,
   kIPMMsgExpired                   =  0x0007,
   kIPMMsgNoTranslatableContent     =  0x0008,
   kIPMRecipientReqStdCont          =  0x0009,
   kIPMRecipientReqSnapShot         =  0x000A,
   kIPMNoTransferDiskFull           =  0x000B,
```

```
   kIPMNoTransferMsgRejectedbyDest  =  0x000C,
   kIPMNoTransferMsgTooLarge        =  0x000D
};

typedef unsigned long   IPMContextRef;

typedef unsigned long   IPMQueueRef;

typedef unsigned long   IPMMsgRef;

typedef unsigned long   IPMSeqNum;

typedef Str32 IPMProcHint;

typedef Str32 IPMQueueName;

typedef OCECreatorType IPMBlockType;
```

Message Addressing Structures

```
typedef DSSpec OCERecipient;

/* format of a packed form recipient */
#define OCEPackedRecipientHeader\
   unsigned short       dataLength;

struct ProtoOCEPackedRecipient {
   OCEPackedRecipientHeader
};
typedef struct ProtoOCEPackedRecipient ProtoOCEPackedRecipient;

# define kOCEPackedRecipientMAXBYTES (4096 - sizeof(ProtoOCEPackedRecipient))

struct OCEPackedRecipient {
   OCEPackedRecipientHeader
   Byte     data[kOCEPackedRecipientMaxBytes];
};
typedef struct OCEPackedRecipient OCEPackedRecipient;

struct IPMEntnQueueExtension {
   Str32 queueName;
};
typedef struct IPMEntnQueueExtension IPMEntnQueueExtension;
```

```
struct IPMEntnAttributeExtension {/* kOCEAttrXtn */
   AttributeType attributeName;
};
typedef struct IPMEntnAttributeExtension IPMEntnAttributeExtension;

struct IPMEntnSpecificAttributeExtension {   /* reserved */
   AttributeCreationID attributeCreationID;
   AttributeType attributeName;
};
typedef struct IPMEntnSpecificAttributeExtension
IPMEntnSpecificAttributeExtension;

struct IPMEntityNameExtension {
   OSType subExtensionType;
   union {
      IPMEntnSpecificAttributeExtension specificAttribute;
      IPMEntnAttributeExtension     attribute;
      IPMEntnQueueExtension         queue;
   } u;
};
typedef struct IPMEntityNameExtension IPMEntityNameExtension;
```

Message and Block Types

```
struct OCECreatorType {
   OSType         msgCreator;
   OSType         msgType;
};
typedef struct OCECreatorType OCECreatorType;

typedef Str32 IPMStringMsgType;

struct IPMMsgType {
   IPMMsgFormat         format;/* IPMMsgFormat*/
   union{
      OCECreatorType    msgOSType;
      IPMStringMsgType  msgStrType;
   }theType;
};
typedef struct IPMMsgType IPMMsgType;
```

Delivery Notification

```
struct IPMMsgID {
   unsigned long id[4];
   };
typedef struct IPMMsgID IPMMsgID;

struct IPMReportBlockHeader {
   IPMMsgID       msgID;         /* message ID of the original */
   UTCTime        creationTime;  /* creation time of the report */
};
typedef struct IPMReportBlockHeader IPMReportBlockHeader;

struct OCERecipientReport {
   unsigned short rcptIndex;     /* index of recipient in original message */
   OSErr          result;        /* result of sending letter to recipient */
};
typedef struct OCERecipientReport OCERecipientReport;
```

Filter Structures

```
struct IPMSingleFilter {   /* each field should be packed and word aligned */
   IPMPriority    priority;
   Byte           padByte;
   OSType         family;  /* family of this msg, '????' for all */
   ScriptCode     script;  /* language identifier */
   IPMProcHint    hint;
   IPMMsgType     msgType;
};
typedef struct IPMSingleFilter IPMSingleFilter;

struct IPMFilter {
   unsigned short    count;
   IPMSingleFilter   sFilters[1];
};
typedef struct IPMFilter IPMFilter;
```

Message Information Structure

```
struct IPMMsgInfo {  /* master message info */
   IPMSeqNum         sequenceNum;

   unsigned long     userData;
   unsigned short    respIndex;
```

```
   Byte              padByte;
   IPMPriority       priority;
   unsigned long     msgSize;
   unsigned short    originalRcptCount;
   unsigned short    reserved;
   UTCTime           creationTime;
   IPMMsgID          msgID;
   OSType            family;     /* family of this msg (e.g., mail) */
   IPMProcHint       procHint;   /* packed and even-length padded */
   IPMMsgType        msgType;    /* packed and even-length padded */
};
typedef struct IPMMsgInfo IPMMsgInfo;
```

Header Information Structures

```
struct IPMTOC {
   IPMBlockType      blockType;
   long              blockOffset;
   unsigned long     blockSize;
   unsigned long     blockRefCon;
};
typedef struct IPMTOC IPMTOC;

struct IPMFixedHdrInfo {
   unsigned short       version;          /* IPM Manager version */
   Boolean              authenticated;    /* was message authenticated? */
   Boolean              signatureEnclosed;/* digital signature enclosed? */
   unsigned long        msgSize;          /* size of message */
   IPMNotificationType  notification;     /* notification type requested */
   IPMPriority          priority;         /* message priority */
   unsigned short       blockCount;       /* number of blocks */
   unsigned short       originalRcptCount;/* original number of recipients */
   unsigned long        refCon;           /* application-defined data */
   unsigned short       reserved;         /* reserved */
   UTCTime              creationTime;     /* message creation time */
   IPMMsgID             msgID;            /* message ID */
   OSType               family;           /* family of this msg */
};
```

Sender Structure

```
struct IPMSender {
   IPMSenderTag      sendTag;
   union{
```

```
      RString        rString;
      PackedRecordID rid;
   } theSender;
};
typedef struct IPMSender IPMSender;
```

Parameter Block Header

```
#define IPMParamHeader     \
   Ptr      qLink;         \
   long     reservedH1;    \
   long     reservedH2;    \
   ProcPtr  ioCompletion;  \
   OSErr    ioResult;      \
   long     saveA5;        \
   short    reqCode;
```

Parameter Blocks for Creating a New Message

```
struct IPMNewMsgPB {
   IPMParamHeader
   unsigned long     filler;
   OCERecipient*     recipient;
   OCERecipient*     replyQueue;
   StringPtr         procHint;
   unsigned short    filler2;
   IPMMsgType*       msgType;
   unsigned long     refCon;
   IPMMsgRef         newMsgRef;
   unsigned short    filler3;
   long              filler4;
   AuthIdentity      identity;
   IPMSender*        sender;
   unsigned long     internalUse;
   unsigned long     internalUse2;
};
typedef struct IPMNewMsgPB IPMNewMsgPB;

struct IPMNewHFSMsgPB {
   IPMParamHeader
   FSSpec*           hfsPath;
   OCERecipient*     recipient;
   OCERecipient*     replyQueue;
   StringPtr         procHint;
```

```
   unsigned short    filler2;
   IPMMsgType*       msgType;
   unsigned long     refCon;
   IPMMsgRef         newMsgRef;
   unsigned short    filler3;
   long              filler4;
   AuthIdentity      identity;
   IPMSender*        sender;
   unsigned long     internalUse;
   unsigned long     internalUse2;
};
typedef struct IPMNewHFSMsgPB IPMNewHFSMsgPB;

typedef struct IPMAddRecipientPB {
   IPMParamHeader
   IPMMsgRef         msgRef;
   OCERecipient*     recipient;
   long              reserved;
};
typedef struct IPMAddRecipientPB IPMAddRecipientPB;

struct IPMAddReplyQueuePB {
   IPMParamHeader
   IPMMsgRef         msgRef;
   long              filler;
   OCERecipient*     replyQueue;
};
typedef struct IPMAddReplyQueuePB IPMAddReplyQueuePB;

struct IPMNewBlockPB {
   IPMParamHeader
   IPMMsgRef         msgRef;
   IPMBlockType      blockType;
   unsigned short    filler[5];
   unsigned long     refCon;
   unsigned short    filler2[3];
   long              startingOffset;
};
typedef struct IPMNewBlockPB IPMNewBlockPB;

struct IPMNewNestedMsgBlockPB {
   IPMParamHeader
   IPMMsgRef         msgRef;
   OCERecipient*     recipient;
```

```
   OCERecipient*     replyQueue;
   StringPtr         procHint;
   unsigned short    filler1;
   IPMMsgType*       msgType;
   unsigned long     refCon;
   IPMMsgRef         newMsgRef;
   unsigned short    filler2;
   long              startingOffset;
   AuthIdentity      identity;
   IPMSender*        sender;
   unsigned long     internalUse;
   unsigned long     internalUse2;
};
typedef struct IPMNewNestedMsgBlockPB IPMNewNestedMsgBlockPB;

struct IPMNestMsgPB {
   IPMParamHeader
   IPMMsgRef         msgRef;
   unsigned short    filler[9];
   unsigned long     refCon;
   IPMMsgRef         msgToNest;
   unsigned short    filler2;
   long              startingOffset;
};
typedef struct IPMNestMsgPB IPMNestMsgPB;

struct IPMWriteMsgPB {
   IPMParamHeader
   IPMMsgRef            msgRef;
   IPMAccessMode        mode;
   long                 offset;
   unsigned long        count;
   Ptr                  buffer;
   unsigned long        actualCount;
   Boolean              currentBlock;
};
typedef struct IPMWriteMsgPB IPMWriteMsgPB;

struct IPMEndMsgPB {
   IPMParamHeader
   IPMMsgRef            msgRef;
   IPMMsgID             msgID;
   RString*             msgTitle;
   IPMNotificationType  deliveryNotification;
```

```
   IPMPriority          priority;
   Boolean              cancel;
   Byte                 padByte;
   long                 reserved;
   SIGSignaturePtr      signature;
   Size                 signatureSize;
   SIGContextPtr        signatureContext;
   OSType               family;  /* family of this msg
                                    use kIPMFamilyUnspecified by default */
};
typedef struct IPMEndMsgPB IPMEndMsgPB;
```

Parameter Blocks for Managing Message Queues

```
struct IPMCreateQueuePB {
   IPMParamHeader
   long              filler1;
   OCERecipient*     queue;
   AuthIdentity      identity;/* used only if queue is remote */
   PackedRecordID*   owner;   /* used only if queue is remote */
};
typedef struct IPMCreateQueuePB IPMCreateQueuePB;

struct IPMOpenContextPB {
   IPMParamHeader
   IPMContextRef        contextRef;
};
typedef struct IPMOpenContextPB IPMOpenContextPB;

struct IPMOpenQueuePB {
   IPMParamHeader
   IPMContextRef        contextRef;
   OCERecipient*        queue;
   AuthIdentity         identity;
   IPMFilter*           filter;
   IPMQueueRef          newQueueRef;
   IPMNoteProcPtr       notificationProc;    /* must be nil */
   unsigned long        userData;            /* reserved */
   IPMNotificationType  noteType;            /* reserved */
   Byte                 padByte;             /* reserved */
   long                 reserved;
   long                 reserved2;
};
typedef struct IPMOpenQueuePB IPMOpenQueuePB;
```

```
typedef IPMEnumerateQueuePB   IPMChangeQueueFilterPB;

typedef IPMOpenContextPB      IPMCloseContextPB;

struct IPMCloseQueuePB {
   IPMParamHeader
   IPMQueueRef    queueRef;
};
typedef struct IPMCloseQueuePB IPMCloseQueuePB;

typedef IPMCreateQueuePB   IPMDeleteQueuePB;
```

Parameter Blocks for Listing and Reading Messages

```
struct IPMEnumerateQueuePB {
   IPMParamHeader
   IPMQueueRef    queueRef;
   IPMSeqNum      startSeqNum;
   Boolean        getProcHint;
   Boolean        getMsgType;
   short          filler;
   IPMFilter*     filter;
   unsigned short numToGet;
   unsigned short numGotten;
   unsigned long  enumCount;
   Ptr            enumBuffer;    /* will be packed array of IPMMsgInfo */
   unsigned long  actEnumCount;
};
typedef struct IPMEnumerateQueuePB IPMEnumerateQueuePB;

struct IPMOpenMsgPB {
   IPMParamHeader
   IPMQueueRef    queueRef;
   IPMSeqNum      sequenceNum;
   IPMMsgRef      newMsgRef;
   IPMSeqNum      actualSeqNum;
   Boolean        exactMatch;
   Byte           padByte;
   long           reserved;
};
typedef struct IPMOpenMsgPB IPMOpenMsgPB;

struct IPMOpenHFSMsgPB {
   IPMParamHeader
   FSSpec*        hfsPath;
```

```
   long           filler;
   IPMMsgRef      newMsgRef;
   long           filler2;
   Byte           filler3;
   long           reserved;
};
typedef struct IPMOpenHFSMsgPB IPMOpenHFSMsgPB;

struct IPMOpenBlockAsMsgPB {
   IPMParamHeader
   IPMMsgRef      msgRef;
   unsigned long  filler;
   IPMMsgRef      newMsgRef;
   unsigned short filler2[7];
   unsigned short blockIndex;
};
typedef struct IPMOpenBlockAsMsgPB IPMOpenBlockAsMsgPB;

struct IPMGetMsgInfoPB {
   IPMParamHeader
   IPMMsgRef              msgRef;
   IPMMsgInfo*            info;
};
typedef struct IPMGetMsgInfoPB IPMGetMsgInfoPB;

struct IPMReadHeaderPB {
   IPMParamHeader
   IPMMsgRef              msgRef;
   unsigned short         fieldSelector;
   long                   offset;
   unsigned long          count;
   Ptr                    buffer;
   unsigned long          actualCount;
   unsigned short         filler;
   unsigned long          remaining;
};
typedef struct IPMReadHeaderPB IPMReadHeaderPB;

struct IPMReadRecipientPB {
   IPMParamHeader
   IPMMsgRef              msgRef;
   unsigned short         rcptIndex;
   long                   offset;
   unsigned long          count;
```

```
   Ptr                     buffer;
   unsigned long           actualCount;
   short                   reserved;   /* must be 0 */
   unsigned long           remaining;
   unsigned short          originalIndex;
   OCERecipientOffsetFlags recipientOffsetFlags;
};
typedef struct IPMReadRecipientPB IPMReadRecipientPB;

typedef IPMReadRecipientPB IPMReadReplyQueuePB;

struct IPMGetBlkIndexPB {
   IPMParamHeader
   IPMMsgRef      msgRef;
   IPMBlockType   blockType;
   unsigned short index;
   unsigned short startingFrom;
   IPMBlockType   actualBlockType;
   unsigned short actualBlockIndex;
};
typedef struct IPMGetBlkIndexPB IPMGetBlkIndexPB;

struct IPMReadMsgPB {
   IPMParamHeader
   IPMMsgRef      msgRef;
   IPMAccessMode  mode;
   long           offset;
   unsigned long  count;
   Ptr            buffer;
   unsigned long  actualCount;
   unsigned short blockIndex;
   unsigned long  remaining;
};
typedef struct IPMReadMsgPB IPMReadMsgPB;

struct IPMVerifySignaturePB {
   IPMParamHeader
   IPMMsgRef      msgRef;
   SIGContextPtr  signatureContext;
};
typedef struct IPMVerifySignaturePB IPMVerifySignaturePB;
```

```
struct IPMCloseMsgPB {
   IPMParamHeader
   IPMMsgRef      msgRef;
   Boolean        deleteMsg;
};
typedef struct IPMCloseMsgPB IPMCloseMsgPB;
```

Parameter Block for Deleting Messages

```
struct IPMDeleteMsgRangePB {
   IPMParamHeader
   IPMQueueRef    queueRef;
   IPMSeqNum      startSeqNum;
   IPMSeqNum      endSeqNum;
   IPMSeqNum      lastSeqNum;
};
typedef struct IPMDeleteMsgRangePB IPMDeleteMsgRangePB;
```

Parameter Block Union Structure

```
union IPMParamBlock {
   struct {IPMParamHeader} header;
   IPMOpenContextPB        openContextPB;
   IPMCloseContextPB       closeContextPB;
   IPMCreateQueuePB        createQueuePB;
   IPMDeleteQueuePB        deleteQueuePB;
   IPMOpenQueuePB          openQueuePB;
   IPMCloseQueuePB         closeQueuePB;
   IPMEnumerateQueuePB     enumerateQueuePB;
   IPMChangeQueueFilterPB  changeQueueFilterPB;
   IPMDeleteMsgRangePB     deleteMsgRangePB;
   IPMOpenMsgPB            openMsgPB;
   IPMOpenHFSMsgPB         openHFSMsgPB;
   IPMOpenBlockAsMsgPB     openBlockAsMsgPB;
   IPMCloseMsgPB           closeMsgPB;
   IPMGetMsgInfoPB         getMsgInfoPB;
   IPMReadHeaderPB         readHeaderPB;
   IPMReadRecipientPB      readRecipientPB;
   IPMReadReplyQueuePB     readReplyQueuePB;
   IPMGetBlkIndexPB        getBlkIndexPB;
   IPMReadMsgPB            readMsgPB;
   IPMVerifySignaturePB    verifySignaturePB;
   IPMNewMsgPB             newMsgPB;
   IPMNewHFSMsgPB          newHFSMsgPB;
```

```
   IPMNestMsgPB            nestMsgPB;
   IPMNewNestedMsgBlockPB  newNestedMsgBlockPB;
   IPMEndMsgPB             endMsgPB;
   IPMAddRecipientPB       addRecipientPB;
   IPMAddReplyQueuePB      addReplyQueuePB;
   IPMNewBlockPB           newBlockPB;
   IPMWriteMsgPB           writeMsgPB;
};
typedef union IPMParamBlock IPMParamBlock;
typedef IPMParamBlock *IPMParamBlockPtr;
```

IPM Manager Functions

Creating a New Message

```
pascal OSErr IPMNewMsg      (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMNewHFSMsg   (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMAddRecipient
                            (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMAddReplyQueue
                            (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMNewBlock    (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMNewNestedMsgBlock
                            (IPMParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr IPMNestMsg     (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMWriteMsg    (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMEndMsg      (IPMParamBlockPtr paramBlock, Boolean async);
```

Managing Message Queues

```
pascal OSErr IPMCreateQueue
                            (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMOpenContext
                            (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMOpenQueue   (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMChangeQueueFilter
                            (IPMParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr IPMCloseQueue  (IPMParamBlockPtr paramBlock, Boolean async);
```

```
pascal OSErr IPMCloseContext
                            (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMDeleteQueue
                            (IPMParamBlockPtr paramBlock, Boolean async);
```

Listing and Reading Messages

```
pascal OSErr IPMEnumerateQueue
                            (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMOpenMsg     (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMOpenHFSMsg  (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMOpenBlockAsMsg
                            (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMGetMsgInfo  (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMReadHeader
                            (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMReadRecipient
                            (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMReadReplyQueue
                            (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMGetBlkIndex
                            (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMReadMsg     (IPMParamBlockPtr paramBlock, Boolean async);
pascal OSErr IPMVerifySignature
                            (IPMParamBlockPtr paramBlock);
pascal OSErr IPMCloseMsg    (IPMParamBlockPtr paramBlock, Boolean async);
```

Deleting Messages

```
pascal OSErr IPMDeleteMsgRange
                            (IPMParamBlockPtr paramBlock, Boolean async);
```

Utility Functions

```
pascal unsigned short OCESizePackedRecipient
                            (const OCERecipient *rcpt);
pascal unsigned short OCEPackRecipient
                            (const OCERecipient *rcpt, void* buffer);
pascal OSErr OCEUnpackRecipient
                            (const void* buffer, OCERecipient *rcpt,
                             RecordID *entitySpecifier);
```

```
pascal OSErr OCEStreamRecipient
                            (const OCERecipient* rcpt, OCERecipientStreamer
                             stream, long userData, unsigned long*
                             actualCount);
pascal OSType OCEGetRecipientType
                            (const CreationID *cid);
pascal void OCESetRecipientType
                            (OSType extensionType, CreationID *cid);
```

Application-Defined Functions

```
pascal void MyCompletionRoutine
                            (Ptr paramBlk);
pascal OSErr MyRecipientStreamer
                            (void* buffer, unsigned long count,
                             Boolean eof, long userData);
```

Pascal Summary

Constants

```
CONST

{ values of IPMPriority }
   kIPMAnyPriority                 = 0;     { for filter only}
   kIPMNormalPriority              = 1;

{ values of IPMAccessMode }
   kIPMAtMark                      = 0;
   kIPMFromStart                   = 1;
   kIPMFromLEOM                    = 2;
   kIPMFromMark                    = 3;

   kIPMUpdateMsgBit                = 4;
   kIPMNewMsgBit                   = 5;
   kIPMDeleteMsgBit                = 6;

{ values of IPMNotificationType }
   kIPMUpdateMsgMask               = $10;   {1<<kIPMUpdateMsgBit}
   kIPMNewMsgMask                  = $20;   {1<<kIPMNewMsgBit}
   kIPMDeleteMsgMask               = $40;   {1<<kIPMDeleteMsgBit}
```

```
{ values of IPMSenderTag }
   kIPMSenderRStringTag             = 0;
   kIPMSenderRecordIDTag            = 1;

   kIPMFromDistListBit              = 0;
   kIPMDummyRecBit                  = 1;
   kIPMFeedbackRecBit               = 2;     { redirect to feedback queue }
   kIPMReporterRecBit               = 3      { redirect to reporter original
                                               queue }

   kIPMBCCRecBit                    = 4;     { this recipient is blind to all
                                               recipients of message }

{ values of OCERecipientOffsetFlags }
   kIPMFromDistListMask             = $01;   {1<<kIPMFromDistListBit}
   kIPMDummyRecMask                 = $02;   {1<<kIPMDummyRecBit}
   kIPMFeedbackRecMask              = $04;   {1<<kIPMFeedbackRecBit}
   kIPMReporterRecMask              = $08;   {1<<kIPMReporterRecBit}
   kIPMBCCRecMask                   = $10;   {1<<kIPMBCCRecBit}

   kIPMTypeWildCard                 = 'ipmw';

   kIPMFamilyUnspecified            = 0;
   kIPMFamilyWildCard               = '????';

{ well known signature }
   kIPMSignature                    = 'ipms';{ base type }

{ well known message types }

   kIPMReportNotify                 = 'rptn';{ routing feedback }

{ well known message block types }
   kIPMEnclosedMsgType              = 'emsg';{ enclosed (nested) message }
   kIPMReportInfo                   = 'rpti';{ recipient information }
   kIPMDigitalSignature             = 'dsig';{ digital signature }

{ values of IPMMsgFormat }
   kIPMOSFormatType                 = 1;
   kIPMStringFormatType             = 2;
```

```
{Following are the known extension values for IPM addresses handled by
Apple Computer, Inc.}

   kOCEalanXtn                      = 'alan';
   kOCEentnXtn                      = 'entn';{ 'entn' = entity name
                                              (DSSpec: aka) }
   kOCEaphnXtn                      = 'aphn';

{ 'entn' extension forms }
   kOCEAddrXtn                      = 'addr';{ reserved }
   kOCEQnamXtn                      = 'qnam';{queue-name form }
   kOCEAttrXtn                      = 'attr';{ an attribute specification }
   kOCESpAtXtn                      = 'spat';{ specific attribute }

{ phoneNumber subtype constants }
   kOCEUseHandyDial                 = 1;
   kOCEDontUseHandyDial             = 2;

   kOCEPackedRecipientMaxBytes      =
                        (4096 - sizeof(ProtoOCEPackedRecipient));

{ addresses with kIPMNBPXtn should specify this nbp type }
   kIPMWSReceiverNBPType            = 'MsgReceiver';

{ values of IPMHeaderSelector }
   kIPMTOC                          = 0;
   kIPMSender                       = 1;
   kIPMProcessHint                  = 2;
   kIPMMessageTitle                 = 3;
   kIPMMessageType                  = 4;
   kIPMFixedInfo                    = 7;

   kIPMDeliveryNotificationBit      = 0;
   kIPMNonDeliveryNotificationBit   = 1;
   kIPMEncloseOriginalBit           = 2;
   kIPMSummaryReportBit             = 3;
   kIPMOriginalOnlyOnErrorBit       = 4;

   kIPMNoNotificationMask           = $00;
   kIPMDeliveryNotificationMask     = $01;  {1<<kIPMDeliveryNotificationBit}
   kIPMNonDeliveryNotificationMask  = $02;
                        {1<<kIPMNonDeliveryNotificationBit}
   kIPMDontEncloseOriginalMask      = $00;
   kIPMEncloseOriginalMask          = $04;  {1<<kIPMEncloseOriginalBit}
   kIPMImmediateReportMask          = $00;
```

```
   kIPMSummaryReportMask            = $08;  {1<<kIPMSummaryReportBit}
   kIPMOriginalOnlyOnErrorMask      = $10;  {1<<kIPMOriginalOnlyOnErrorBit}
   kIPMEncloseOriginalOnErrorMask   =
                     kIPMOriginalOnlyOnErrorMask + kIPMEncloseOriginalMask;

{ standard Nondelivery codes }
   kIPMNoSuchRecipient              = $0001;
   kIPMRecipientMalformed           = $0002;
   kIPMRecipientAmbiguous           = $0003;
   kIPMRecipientAccessDenied        = $0004;
   kIPMGroupExpansionProblem        = $0005;
   kIPMMsgUnreadable                = $0006;
   kIPMMsgExpired                   = $0007;
   kIPMMsgNoTranslatableContent     = $0008;
   kIPMRecipientReqStdCont          = $0009;
   kIPMRecipientReqSnapShot         = $000A;
   kIPMNoTransferDiskFull           = $000B;
   kIPMNoTransferMsgRejectedbyDest  = $000C;
   kIPMNoTransferMsgTooLarge        = $000D;
```

Data Types

```
TYPE

IPMPriority = Byte;

IPMAccessMode = INTEGER;

IPMNotificationType = Byte;

IPMSenderTag = INTEGER;

OCERecipientOffsetFlags = Byte;

IPMMsgFormat = INTEGER;

IPMHeaderSelector = Byte;

IPMContextRef = LONGINT;

IPMQueueRef = LONGINT;

IPMMsgRef = LONGINT;

IPMSeqNum = LONGINT;
```

```
IPMProcHint = Str32;

IPMQueueName = Str32;

IPMBlockType = OCECreatorType;
```

Message Addressing Structures

```
OCERecipient = DSSpec;

ProtoOCEPackedRecipient = RECORD
   dataLength:    INTEGER;
   END;

OCEPackedRecipient = RECORD
   dataLength:    INTEGER;
   data:          PACKED ARRAY[1..kOCEPackedRecipientMaxBytes] OF Byte;
   END;

OCEPackedRecipientPtr = ^OCEPackedRecipient;

IPMEntnQueueExtension = RECORD
   queueName: Str32;
   END;

IPMEntnAttributeExtension = RECORD{ kOCEAttrXtn }
   attributeName: AttributeType;
   END;

IPMEntnSpecificAttributeExtension = RECORD{ kOCESpAtXtn }
   attributeCreationID:    AttributeCreationID;
   attributeName:          AttributeType;
   END;

IPMEntityNameExtension = RECORD
   subExtensionType: OSType;
   CASE INTEGER OF
      1: (specificAttribute:  IPMEntnSpecificAttributeExtension);
      2: (attribute:          IPMEntnAttributeExtension);
      3: (queue:              IPMEntnQueueExtension);
   END;
```

Message and Block Types

```
OCECreatorType = RECORD
   msgCreator:       OSType;
   msgType:          OSType;
   END;

IPMStringMsgType = Str32;

IPMMsgType = RECORD
   format: IPMMsgFormat;{ IPMMsgFormat}
   CASE INTEGER OF
      1: (msgOSType:    OCECreatorType);
      2: (msgStrType:   IPMStringMsgType);
   END;
```

Delivery Notification Structures

```
IPMMsgID = RECORD
   id: ARRAY[1..4] OF LONGINT;
   END;

IPMReportBlockHeader = RECORD
   msgID:            IPMMsgID;{ message ID of the original }
   creationTime:     UTCTime;{ creation time of the report }
   END;

OCERecipientReport = RECORD
   rcptIndex:        INTEGER;{ index of recipient in original message }
   result:           OSErr;   { result of sending letter to this recipient}
   END;
```

Filter Structures

```
IPMSingleFilter = PACKED RECORD
   priority:   IPMPriority;
   padByte:    Byte;
   family:     OSType;     { family of this msg, '????' for all }
   script:     ScriptCode; { language identifier }
   hint:       IPMProcHint;
   msgType:    IPMMsgType;
   END;
```

```
IPMFilter = RECORD
   count:      INTEGER;
   sFilters:   ARRAY[1..1] OF IPMSingleFilter;
   END;
```

Message Information Structure

```
IPMMsgInfo = PACKED RECORD
   sequenceNum:         IPMSeqNum;
   userData:            LONGINT;
   respIndex:           INTEGER;
   padByte:             Byte;
   priority:            IPMPriority;
   msgSize:             LONGINT;
   originalRcptCount:   INTEGER;
   reserved:            INTEGER;
   creationTime:        UTCTime;
   msgID:               IPMMsgID;
   family:              OSType;        { family of this msg
                                         (e.g. mail) }
   procHint:            IPMProcHint;   { packed and even-length padded }
   msgType:             IPMMsgType;    { packed and even-length padded }
   END;
```

Header Information Structures

```
IPMTOC = RECORD
   blockType:        IPMBlockType;
   blockOffset:      LONGINT;
   blockSize:        LONGINT;
   blockRefCon:      LONGINT;
   END;

IPMFixedHdrInfo = PACKED RECORD
   version:             INTEGER;           { IPM Manager version }
   authenticated:       BOOLEAN;           { was message authenticated? }
   signatureEnclosed:   BOOLEAN;           { digital signature enclosed? }
   msgSize:             LONGINT;           { size of message }
   notification:        IPMNotificationType;{ notification type requested }
   priority:            IPMPriority;       { message priority }
   blockCount:          INTEGER;           { number of blocks }
   originalRcptCount:   INTEGER;           { original number of recipients }
   refCon:              LONGINT;           { application-defined data }
   reserved:            INTEGER;           { reserved }
```

```
   creationTime:        UTCTime;          { message creation time }
   msgID:               IPMMsgID;         { message ID }
   family:              OSType;           { family of this msg }
   END;
```

Sender Structure

```
IPMSender = RECORD
   sendTag: IPMSenderTag;
   CASE INTEGER OF
      1: (rString: RString);
      2: (rid:     PackedRecordID);
   END;
```

Parameter Block Header

```
IPMParamHeader = RECORD
   qLink:         Ptr;
   reservedH1:    LONGINT;
   reservedH2:    LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   END;
```

Parameter Blocks for Creating a New Message

```
IPMNewMsgPB = RECORD
   qLink: Ptr;
   reservedH1:       LONGINT;
   reservedH2:       LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   filler:           LONGINT;
   recipient:        ^OCERecipient;
   replyQueue:       ^OCERecipient;
   procHint:         StringPtr;
   filler2:          INTEGER;
   msgType:          ^IPMMsgType;
   refCon:           LONGINT;
   newMsgRef:        IPMMsgRef;
```

```
   filler3:          INTEGER;
   filler4:          LONGINT;
   identity:         AuthIdentity;
   sender:           ^IPMSender;
   internalUse:      LONGINT;
   internalUse2:     LONGINT;
   END;

IPMNewHFSMsgPB = RECORD
   qLink:            Ptr;
   reservedH1:       LONGINT;
   reservedH2:       LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:            LONGINT;
   reqCode:          INTEGER;
   hfsPath:          ^FSSpec;
   recipient:        ^OCERecipient;
   replyQueue:       ^OCERecipient;
   procHint:         StringPtr;
   filler2:          INTEGER;
   msgType:          ^IPMMsgType;
   refCon:           LONGINT;
   newMsgRef:        IPMMsgRef;
   filler3:          INTEGER;
   filler4:          LONGINT;
   identity:         AuthIdentity;
   sender:           ^IPMSender;
   internalUse:      LONGINT;
   internalUse2:     LONGINT;
   END;

IPMAddRecipientPB = RECORD
   qLink:            Ptr;
   reservedH1:       LONGINT;
   reservedH2:       LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   msgRef:           IPMMsgRef;
   recipient:        ^OCERecipient;
   reserved:         LONGINT;
   END;
```

```
IPMAddReplyQueuePB = RECORD
   qLink:            Ptr;
   reservedH1:       LONGINT;
   reservedH2:       LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   msgRef:           IPMMsgRef;
   filler:           LONGINT;
   replyQueue:       ^OCERecipient;
   END;

IPMNewBlockPB = RECORD
   qLink:            Ptr;
   reservedH1:       LONGINT;
   reservedH2:       LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   msgRef:           IPMMsgRef;
   blockType:        IPMBlockType;
   filler:           ARRAY[1..5] OF INTEGER;
   refCon:           LONGINT;
   filler2:          ARRAY[1..3] OF INTEGER;
   startingOffset:   LONGINT;
   END;

IPMNewNestedMsgBlockPB = RECORD
   qLink:            Ptr;
   reservedH1:       LONGINT;
   reservedH2:       LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   msgRef:           IPMMsgRef;
   recipient:        ^OCERecipient;
   replyQueue:       ^OCERecipient;
   procHint:         StringPtr;
   filler1:          INTEGER;
   msgType:          ^IPMMsgType;
   refCon:           LONGINT;
```

```
   newMsgRef:        IPMMsgRef;
   filler2:          INTEGER;
   startingOffset:   LONGINT;
   identity:         AuthIdentity;
   sender:           ^IPMSender;
   internalUse:      LONGINT;
   internalUse2:     LONGINT;
   END;

IPMNestMsgPB = RECORD
   qLink:            Ptr;
   reservedH1:       LONGINT;
   reservedH2:       LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   msgRef:           IPMMsgRef;
   filler:           ARRAY[1..9] OF INTEGER;
   refCon:           LONGINT;
   msgToNest:        IPMMsgRef;
   filler2:          INTEGER;
   startingOffset:   LONGINT;
   END;

IPMWriteMsgPB = RECORD
   qLink:            Ptr;
   reservedH1:       LONGINT;
   reservedH2:       LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   msgRef:           IPMMsgRef;
   mode:             IPMAccessMode;
   offset:           LONGINT;
   count:            LONGINT;
   buffer:           Ptr;
   actualCount:      LONGINT;
   currentBlock:     BOOLEAN;
   END;
```

```
IPMEndMsgPB = PACKED RECORD
   qLink:               Ptr;
   reservedH1:          LONGINT;
   reservedH2:          LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   msgRef:              IPMMsgRef;
   msgID:               IPMMsgID;
   msgTitle:            ^RString;
   deliveryNotification: IPMNotificationType;
   priority:            IPMPriority;
   cancel:              BOOLEAN;
   padByte:             Byte;
   reserved:            LONGINT;
   signature:           SIGSignaturePtr;
   signatureSize:       Size;
   signatureContext:    SIGContextPtr;
   family:              OSType;          { family of this msg (e.g.,
                                           mail) use kIPMFamilyUnspecified
                                           by default }
   END;
```

Parameter Blocks for Managing Message Queues

```
IPMCreateQueuePB = RECORD
   qLink:               Ptr;
   reservedH1:          LONGINT;
   reservedH2:          LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   filler1:             LONGINT;
   queue:               ^OCERecipient;
   identity:            AuthIdentity;     { used only if queue is remote }
   owner:               ^PackedRecordID;  { used only if queue is remote }
   END;

IPMOpenContextPB = RECORD
   qLink:               Ptr;
   reservedH1:          LONGINT;
   reservedH2:          LONGINT;
```

```
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   contextRef:          IPMContextRef; { context reference to be used in
                                         further calls}
   END;

IPMOpenQueuePB = PACKED RECORD
   qLink:               Ptr;
   reservedH1:          LONGINT;
   reservedH2:          LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   contextRef:          IPMContextRef;
   queue:               ^OCERecipient;
   identity:            AuthIdentity;
   filter:              ^IPMFilter;
   newQueueRef:         IPMQueueRef;
   notificationProc:    IPMNoteProcPtr;
   userData:            LONGINT;
   noteType:            IPMNotificationType;
   padByte:             Byte;
   reserved:            LONGINT;
   reserved2:           LONGINT;
   END;

IPMChangeQueueFilterPB = IPMEnumerateQueuePB;

IPMCloseContextPB = IPMOpenContextPB;

IPMCloseQueuePB = RECORD
   qLink:            Ptr;
   reservedH1:       LONGINT;
   reservedH2:       LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   queueRef:         IPMQueueRef;
   END;

IPMDeleteQueuePB = IPMCreateQueuePB;
```

Parameter Blocks for Listing and Reading Messages

```
IPMEnumerateQueuePB = RECORD
   qLink:            Ptr;
   reservedH1:       LONGINT;
   reservedH2:       LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   queueRef:         IPMQueueRef;
   startSeqNum:      IPMSeqNum;
   getProcHint:      BOOLEAN;
   getMsgType:       BOOLEAN;
   filler:           INTEGER;
   filter:           ^IPMFilter;
   numToGet:         INTEGER;
   numGotten:        INTEGER;
   enumCount:        LONGINT;
   enumBuffer:       Ptr;        { will be packed array of IPMMsgInfo }
   actEnumCount:     LONGINT;
   END;

IPMOpenMsgPB = PACKED RECORD
   qLink:            Ptr;
   reservedH1:       LONGINT;
   reservedH2:       LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   queueRef:         IPMQueueRef;
   sequenceNum:         IPMSeqNum;
   newMsgRef:           IPMMsgRef;
   actualSeqNum:        IPMSeqNum;
   exactMatch:          BOOLEAN;
   padByte:             Byte;
   reserved:            LONGINT;
   END;

IPMOpenHFSMsgPB = RECORD
   qLink:            Ptr;
   reservedH1:       LONGINT;
   reservedH2:       LONGINT;
```

```
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   hfsPath:          ^FSSpec;
   filler:           LONGINT;
   newMsgRef:        IPMMsgRef;
   filler2:          LONGINT;
   filler3:          Byte;
   reserved:         LONGINT;
   END;

IPMOpenBlockAsMsgPB = RECORD
   qLink:            Ptr;
   reservedH1:       LONGINT;
   reservedH2:       LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   msgRef:           IPMMsgRef;
   filler:           LONGINT;
   newMsgRef:        IPMMsgRef;
   filler2:          ARRAY[1..7] OF INTEGER;
   blockIndex:       INTEGER;
   END;

IPMGetMsgInfoPB = RECORD
   qLink:            Ptr;
   reservedH1:       LONGINT;
   reservedH2:       LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   msgRef:           IPMMsgRef;
   info:             ^IPMMsgInfo;
   END;

IPMReadHeaderPB = RECORD
   qLink:            Ptr;
   reservedH1:       LONGINT;
   reservedH2:       LONGINT;
   ioCompletion:     ProcPtr;
```

```
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   msgRef:           IPMMsgRef;
   fieldSelector:    INTEGER;
   offset:           LONGINT;
   count:            LONGINT;
   buffer:           Ptr;
   actualCount:      LONGINT;
   filler:           INTEGER;
   remaining:        LONGINT;
   END;

IPMReadRecipientPB = PACKED RECORD
   qLink:               Ptr;
   reservedH1:          LONGINT;
   reservedH2:          LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   msgRef:              IPMMsgRef;
   rcptIndex:           INTEGER;
   offset:              LONGINT;
   count:               LONGINT;
   buffer:              Ptr;
   actualCount:         LONGINT;
   reserved:            INTEGER;    { must be 0 }
   remaining:           LONGINT;
   originalIndex:       INTEGER;
   recipientOffsetFlags: OCERecipientOffsetFlags;
   END;

IPMReadReplyQueuePB = IPMReadRecipientPB;

IPMGetBlkIndexPB = RECORD
   qLink:               Ptr;
   reservedH1:          LONGINT;
   reservedH2:          LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   msgRef:              IPMMsgRef;
```

```
   blockType:           IPMBlockType;
   index:               INTEGER;
   startingFrom:        INTEGER;
   actualBlockType:     IPMBlockType;
   actualBlockIndex:    INTEGER;
   END;

IPMReadMsgPB = RECORD
   qLink:               Ptr;
   reservedH1:          LONGINT;
   reservedH2:          LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   msgRef:              IPMMsgRef;
   mode:                IPMAccessMode;
   offset:              LONGINT;
   count:               LONGINT;
   buffer:              Ptr;
   actualCount:         LONGINT;
   blockIndex:          INTEGER;
   remaining:           LONGINT;
   END;

IPMVerifySignaturePB = RECORD
   qLink:               Ptr;
   reservedH1:          LONGINT;
   reservedH2:          LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   msgRef:              IPMMsgRef;
   signatureContext:    SIGContextPtr;
   END;

IPMCloseMsgPB = RECORD
   qLink:               Ptr;
   reservedH1:          LONGINT;
   reservedH2:          LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
```

```
   reqCode:              INTEGER;
   msgRef:               IPMMsgRef;
   deleteMsg:            BOOLEAN;
   END;
```

Parameter Blocks for Deleting Messages

```
IPMDeleteMsgRangePB = RECORD
   qLink:                Ptr;
   reservedH1:           LONGINT;
   reservedH2:           LONGINT;
   ioCompletion:         ProcPtr;
   ioResult:             OSErr;
   saveA5:               LONGINT;
   reqCode:              INTEGER;
   queueRef:             IPMQueueRef;
   startSeqNum:          IPMSeqNum;
   endSeqNum:            IPMSeqNum;
   lastSeqNum:           IPMSeqNum;
   END;
```

Parameter Block Union Structure

```
IPMParamBlock = RECORD
   CASE INTEGER OF
       1:(header: IPMParamHeader);
       2:(openContextPB: IPMOpenContextPB);
       3:(closeContextPB: IPMCloseContextPB);
       4:(createQueuePB: IPMCreateQueuePB);
       5:(deleteQueuePB: IPMDeleteQueuePB);
       6:(openQueuePB: IPMOpenQueuePB);
       7:(closeQueuePB: IPMCloseQueuePB);
       8:(enumerateQueuePB: IPMEnumerateQueuePB);
       9:(changeQueueFilterPB: IPMChangeQueueFilterPB);
      10:(deleteMsgRangePB: IPMDeleteMsgRangePB);
      11:(openMsgPB: IPMOpenMsgPB);
      12:(openHFSMsgPB: IPMOpenHFSMsgPB);
      13:(openBlockAsMsgPB: IPMOpenBlockAsMsgPB);
      14:(closeMsgPB: IPMCloseMsgPB);
      15:(getMsgInfoPB: IPMGetMsgInfoPB);
      16:(readHeaderPB: IPMReadHeaderPB);
      17:(readRecipientPB: IPMReadRecipientPB);
      18:(readReplyQueuePB: IPMReadReplyQueuePB);
      19:(getBlkIndexPB: IPMGetBlkIndexPB);
```

```
      20:(readMsgPB: IPMReadMsgPB);
      21:(verifySignaturePB: IPMVerifySignaturePB);
      22:(newMsgPB: IPMNewMsgPB);
      23:(newHFSMsgPB: IPMNewHFSMsgPB);
      24:(nestMsgPB: IPMNestMsgPB);
      25:(newNestedMsgBlockPB: IPMNewNestedMsgBlockPB);
      26:(endMsgPB: IPMEndMsgPB);
      27:(addRecipientPB: IPMAddRecipientPB);
      28:(addReplyQueuePB: IPMAddReplyQueuePB);
      29:(newBlockPB: IPMNewBlockPB);
      30:(writeMsgPB: IPMWriteMsgPB);
   END;

IPMParamBlockPtr = ^IPMParamBlock;
```

IPM Manager Functions

Creating a New Message

```
FUNCTION IPMNewMsg          (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                             OSErr;
FUNCTION IPMNewHFSMsg       (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                             OSErr;
FUNCTION IPMAddRecipient    (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                             OSErr;
FUNCTION IPMAddReplyQueue   (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                             OSErr;
FUNCTION IPMNewBlock        (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                             OSErr;
FUNCTION IPMNewNestedMsgBlock
                            (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                             OSErr;
FUNCTION IPMNestMsg         (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                             OSErr;
FUNCTION IPMWriteMsg        (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                             OSErr;
FUNCTION IPMEndMsg          (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                             OSErr;
```

Managing Message Queues

```
FUNCTION IPMCreateQueue      (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                              OSErr;
FUNCTION IPMOpenContext      (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                              OSErr;
FUNCTION IPMOpenQueue        (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                              OSErr;
FUNCTION IPMChangeQueueFilter
                             (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                              OSErr;
FUNCTION IPMCloseQueue       (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                              OSErr;
FUNCTION IPMCloseContext     (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                              OSErr;
FUNCTION IPMDeleteQueue      (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                              OSErr;
```

Listing and Reading Messages

```
FUNCTION IPMEnumerateQueue   (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                              OSErr;
FUNCTION IPMOpenMsg          (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                              OSErr;
FUNCTION IPMOpenHFSMsg       (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                              OSErr;
FUNCTION IPMOpenBlockAsMsg   (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                              OSErr;
FUNCTION IPMGetMsgInfo       (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                              OSErr;
FUNCTION IPMReadHeader       (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                              OSErr;
FUNCTION IPMReadRecipient    (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                              OSErr;
FUNCTION IPMReadReplyQueue   (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                              OSErr;
FUNCTION IPMGetBlkIndex      (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                              OSErr;
FUNCTION IPMReadMsg          (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                              OSErr;
```

```
FUNCTION IPMVerifySignature
                            (paramBlock: IPMParamBlockPtr): OSErr;{ Always
                             synchronous }
FUNCTION IPMCloseMsg        (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                             OSErr;
```

Deleting Messages

```
FUNCTION IPMDeleteMsgRange  (paramBlock: IPMParamBlockPtr; async: BOOLEAN):
                             OSErr;
```

Utility Routines

```
FUNCTION OCESizePackedRecipient
                            (rcpt: OCERecipient): INTEGER;
FUNCTION OCEPackRecipient   (rcpt: OCERecipient; buffer: UNIV Ptr): INTEGER;
FUNCTION OCEUnpackRecipient
                            (buffer: UNIV Ptr; VAR rcpt: OCERecipient;
                             VAR entitySpecifier: RecordID): OSErr;
FUNCTION OCEStreamRecipient (rcpt: OCERecipient; stream:
                             OCERecipientStreamer;
                             userData: LONGINT; VAR actualCount: LONGINT):
                             OSErr;
FUNCTION OCEGetRecipientType
                            (cid: CreationID): OSType;
PROCEDURE OCESetRecipientType
                            (extensionType: OSType; VAR cid: CreationID);
```

Application-Defined Functions

```
FUNCTION MyCompletionRoutine
                            (paramBlk: Ptr);
FUNCTION MyRecipientStreamer
                            (VAR buffer: void; count: LONGINT; eof:
                             BOOLEAN; userData: LONGINT): OSErr;)
```

Assembly-Language Summary

Trap Macros Requiring Routine Selectors

`__OCETBDispatch`

Selector	Routine
$0400	`IPMOpenContext`
$0401	`IPMCloseContext`
$0402	`IPMNewMsg`
$0403	`IPMAddRecipient`
$0404	`IPMNewBlock`
$0405	`IPMNewNestedMsgBlock`
$0406	`IPMNestMsg`
$0407	`IPMWriteMsg`
$0408	`IPMEndMsg`
$0409	`IPMOpenQueue`
$040A	`IPMCloseQueue`
$040B	`IPMOpenMsg`
$040C	`IPMCloseMsg`
$040D	`IPMReadMsg`
$040E	`IPMReadHeader`
$040F	`IPMOpenBlockAsMsg`
$0410	`IPMReadRecipient`
$0411	`IPMCreateQueue`
$0412	`IPMDeleteQueue`
$0413	`IPMEnumerateQueue`
$0414	`IPMChangeQueueFilter`
$0415	`IPMDeleteMsgRange`
$0417	`IPMOpenHFSMsg`
$0418	`IPMGetBlkIndex`
$0419	`IPMGetMsgInfo`
$041D	`IPMAddReplyQueue`
$041E	`IPMNewHFSMsg`
$0421	`IPMReadReplyQueue`
$0422	`IPMVerifySignature`

__OCEMessaging

Selector	Routine
$033E	OCESizePackedRecipient
$033F	OCEPackRecipient
$0340	OCEUnpackRecipient
$0341	OCEStreamRecipient
$0342	OCEGetRecipientType
$0343	OCESetRecipientType

Result Codes

The allocated range of result codes for the Interprogram Messaging Manager is –15090 through –15169. Routines may also return result codes from other AOCE managers and standard Macintosh result codes such as noErr 0 (no error) and fnfErr –43 (file not found).

noErr	0	No error
kOCEParamErr	–50	Parameter error
kOCEConnectionClosed	–1513	Network connection has closed
kIPMCantCreateIPMCatEntry	–15090	Internal error
kIPMInvalidMsgType	–15091	Message type is invalid
kIPMInvalidProcHint	–15092	Process hint is invalid
kIPMInvalidOffset	–15093	Bad offset for read or write operation
kIPMUpdateCatFailed	–15094	Internal error
kIPMMsgTypeReserved	–15095	Message type reserved for system use
kIPMNotInABlock	–15096	Specified starting point not within the message
kIPMNestedMsgOpened	–15097	Nested message opened; cannot do operation
kIPMA1HdrCorrupt	–15098	Message is corrupt; may not be message
kIPMCorruptDataStructures	–15099	Message is corrupt
kIPMAbortOfNestedMsg	–15100	Canceled nested message
kIPMBlockIsNotNestedMsg	–15101	Block is not message (IPMOpenBlockAsMsg)
kIPMCacheFillError	–15102	Internal error
kIPMInvalidSender	–15103	Sender is invalid
kIPMNoRecipientsYet	–15104	Require recipient to send
kIPMInvalidFilter	–15105	Filter is invalid
kIPMAttrNotInHdr	–15106	Specified attribute not in message header
kIPMBlkNotFound	–15107	Specified block nonexistent
kIPMStreamErr	–15108	Error on stream
kIPMPortClosed	–15109	Stream closed
kIPMBinBusy	–15110	Internal error
kIPMCorruptedBin	–15111	IPM BIN is damaged
kIPMBadQName	–15112	Invalid queue name
kIPMEndOfBin	–15113	Internal error
kIPMBinNeedsConversion	–15114	IPM BIN needs conversion
kIPMMgrInternalErr	–15115	Internal error
kIPMEltBusy	–15116	Message or letter opened (on delete operation)
kIPMEltClosedNotDeleted	–15117	Element was closed but not deleted

kIPMBadContext	–15118	Invalid reference
kIPMContextIsClosing	–15119	Reference is closing
kIPMeoQ	–15120	No more messages (IPMEnumerateQueue)
kIPMEltNotFound	–15122	No such item or message
kIPMQBusy	–15126	Specified queue busy; cannot delete
kIPMLookupAttrTooBig	–15129	Attribute in lookup is too big
kIPMAccessDenied	–15141	Access denied
kIPMNoAttrsFound	–15146	No attributes found in lookup
kIPMBadMailSlotAttrVal	–15149	Invalid mail slot attribute value

CHAPTER 8

Catalog Manager

Contents

Catalog Manager

This chapter describes the Catalog Manager, which provides access to AOCE catalogs, including PowerShare server-based catalogs, personal catalogs, and external catalogs. AOCE catalogs are repositories of information that have a standard interface defined by AOCE system software. Although AOCE catalogs are commonly used to store addresses for mail and messaging services, there are no restrictions on the type or internal structure of the data they may contain. This chapter tells you how to use the Catalog Manager to create, modify, and read information in AOCE catalogs.

You can add a catalog-browsing interface to your application with the routines described in the chapter "Standard Catalog Package" in this book. Users can browse and modify the information in AOCE catalogs through the Finder when they install PowerTalk system software. The AOCE Catalogs Extension to the Finder is described in the chapter "AOCE Templates," which also tells you how to extend the catalog browser to handle new types of data.

This chapter describes the nature and types of AOCE catalogs and presents the low-level interface to AOCE catalogs. You can use the routines in this chapter if you want to provide capabilities to access catalogs beyond those provided by the Standard Catalog Package and the catalog browser. If you are creating an AOCE catalog service access module for another type of catalog, you need the information in this chapter plus the chapter "Catalog Service Access Modules," in *Inside Macintosh: AOCE Service Access Modules*.

This chapter starts with a general introduction to AOCE catalogs, followed by an introduction to the Catalog Manager. Then it describes how you can use the Catalog Manager to

- get information about AOCE catalogs and catalog nodes
- create, open, and close personal catalogs
- manage the organization of an AOCE catalog
- manage the content of an AOCE catalog
- control access to a catalog and to the contents of a catalog

Apple's PowerShare serves include a catalog and authentication server. The authentication process determines whether a user should be granted access to a PowerShare catalog. The application program interface (API) for the identification and authentication of users is handled by the Authentication Manager, described in the chapter "Authentication Manager" in this book.

For a general overview of AOCE, see the chapter "Introduction to the Apple Open Collaboration Environment" in this book.

Introduction to AOCE Catalogs

There are three types of AOCE catalogs: PowerShare server-based catalogs, personal catalogs, and external catalogs. You use the same set of Catalog Manager routines to read and modify the contents of any of these catalogs. The term "AOCE catalog" may refer to any or all of these types of catalogs.

A **PowerShare server** uses the Apple Catalog and Authentication Protocol to communicate with the AOCE Catalog and Authentication Managers. A PowerShare server can be installed on an AppleTalk network to provide catalog services to any number of entities on that network. In addition to providing a PowerShare catalog, a PowerShare server can identify and authenticate users to ensure that only authorized people or agents gain access to the catalog information. For each user, the server administrator can restrict access to the entire catalog or to any portion of the data in the catalog.

A **personal catalog** is an HFS (Hierarchical File System) file located on a user's local disk. A personal catalog can store anything that can be kept in a PowerShare catalog and is often used to store frequently used information from such a catalog.

An **information card** is a type of personal catalog that contains a single record. Typically, it contains all of a user's electronic address information. Because it contains only one record, it can be sent quickly and easily to other users as needed.

An **external catalog** is one that is accessible to your application through the Catalog Manager API by means of a **catalog service access module** (**CSAM**). You access and use an external catalog exactly as you do a PowerShare catalog. The services of the Catalog Manager can be extended to any catalog through a catalog service access module. You do not need to know about catalog service access modules to gain access to external catalogs through the Catalog Manager.

Every catalog provides a set of capability flags that define which features of the Catalog Manager API the catalog supports. In general, your application uses the capability flags to determine what it can do relative to a given catalog; the underlying catalog type (PowerShare, personal, or external) is, with few exceptions, irrelevant. The capability flags are discussed in "Feature Flag Bit Array" beginning on page 8-28. That section also contains information about what features are supported by all PowerShare catalogs and all personal catalogs.

Each catalog is identified by a name and a reference number known as a **catalog discriminator.** The combination of name and discriminator is almost certain to be unique, and you must use both when calling Catalog Manager routines to address PowerShare or external catalogs. The Catalog Manager returns a *personal catalog reference number* when you open a personal catalog, and you use that number when addressing a personal catalog through the Catalog Manager API.

Catalog Nodes

AOCE catalogs contain information arranged in a hierarchical structure similar to that of the Macintosh hierarchical file system (HFS). At the root level of the hierarchy is the AOCE catalog itself. Each catalog can contain any number of nodes; a **catalog node** (or **dNode**) is a container that can hold other dNodes, records, or both dNodes and records. A dNode is analogous to an HFS folder, which can contain other folders, files, or both folders and files. A **record** is analogous to an HFS file. A record contains the actual data stored in the catalog.

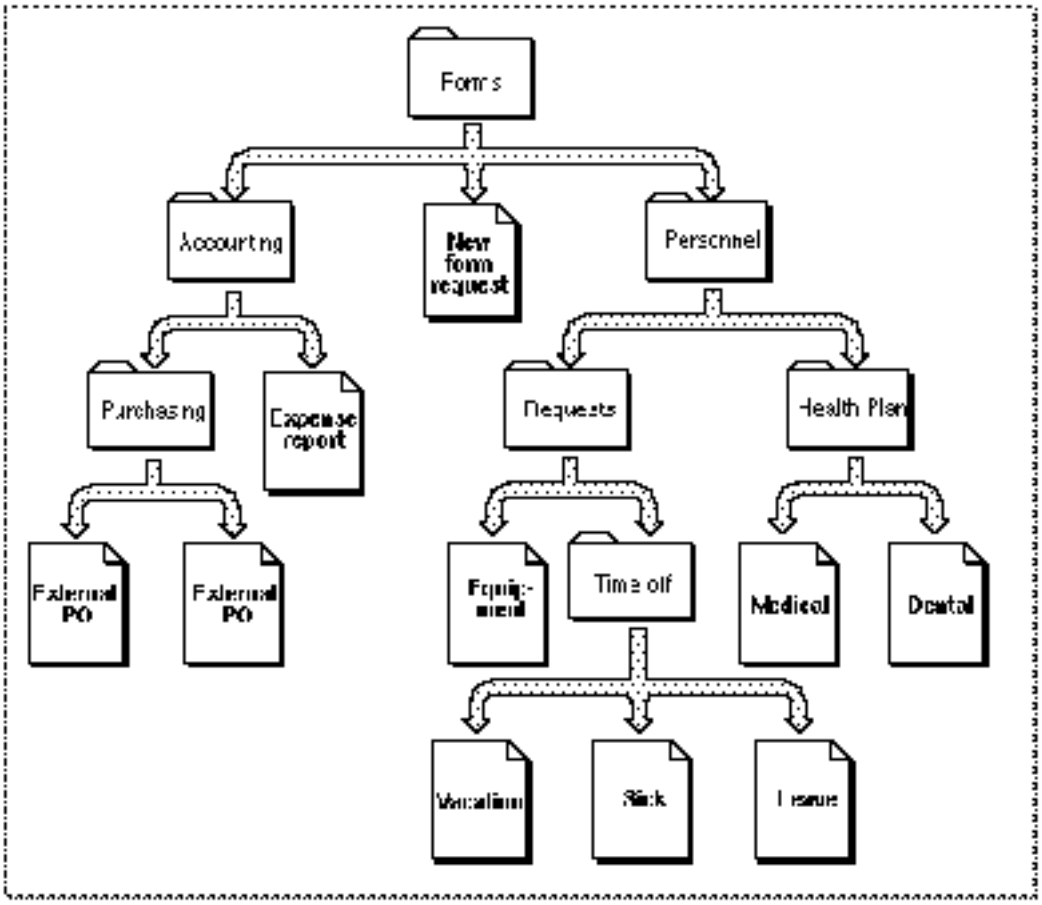
You can identify a specific node within a catalog in three ways. You can specify a dNode by its pathname. A **pathname** consists of the name of each dNode in the catalog tree starting from the first node after the root node and including each intervening node to the node in question. In addition, some catalogs assign a unique number, called a **dNode number,** to each dNode. For such catalogs, you can use the dNode number rather than the pathname to identify a particular dNode. A partial pathname specification is the third way to identify a dNode. Not all catalogs accept a partial pathname. A **partial pathname** consists of a dNode number plus the name of each dNode starting from the one after the dNode specified by the dNode number and continuing to the node in question.

Figure 8-1 on page 8-6 illustrates the structure of a sample AOCE catalog. In this example, the catalog, named Forms, contains personnel and accounting forms for a small company. The stacks of documents in the figure represent catalog nodes, and the individual documents in the figure represent records containing forms. Immediately under the catalog are two nodes, named Accounting and Personnel, and one record containing a form to request new forms. Notice that any given node can contain both records and other nodes. In this example, the pathname for the node containing time-off-request forms is Personnel:Requests:Time off, and the pathname for the node containing purchase orders is Accounting:Purchase. (The colons in the pathname are included for readability only; they are not part of the actual pathname.)

Unlike HFS pathnames, which include the volume name, AOCE pathnames do not include the name of the root catalog. Assume that the nodes named Personnel, Requests, and Time off have the dNode numbers 10, 20, and 30, respectively. In this case, you can either identify the node containing time-off-request forms with a partial pathname that consists of the dNode number 10 and the path Requests:Time off or with the dNode number 20 and the path Time off.

Note that a specific type of catalog might support all or only part of this model. For example, a personal catalog contains only records, no dNodes. An external catalog may support all or any part of this catalog structure.

Figure 8-1 Structure of an AOCE catalog



Catalog Records and Attributes

A record is uniquely identified by a **record ID** that allows the Catalog Manager to classify and locate the record. The Catalog Manager defines the structure of a record but places no restrictions on the type of data it may contain.

A record ID consists of

- record location information
- a record name
- the record type
- a creation ID

The record location information consists of the catalog name and discriminator, the dNode number, and the pathname for the dNode containing the record. The record name can be any string of type `RString` (type `RString` is described in the chapter "AOCE Utilities" in this book). The **record type** indicates the type of entity represented by the record; for example, Printer, User, or Icon. Apple Computer, Inc., defines certain

record types to facilitate collaboration within the AOCE environment; you can define additional record types. The **record creation ID**, assigned by the catalog, is a number that uniquely identifies the record within the catalog. Typically, a user interface uses the record name and type to identify the record, whereas software uses the record creation ID. Not all catalogs support record creation IDs. If a catalog does not support record creation IDs, you use the record name and record type to identify a record.

The information in a record is stored in attribute. An **attribute** is completely specified by an attribute type, an attribute creation ID, an attribute tag, and the actual attribute value. Attribute values are grouped together by attribute type. The **attribute type** reflects the type of data stored in the attribute value; for example, telephone number, mailing address, or picture. Apple defines a number of attribute types; you can define additional attribute types. An attribute type may have zero or more attribute values associated with it. An **attribute creation ID** uniquely identifies the attribute value. Some catalogs may not support attribute creation IDs. If a catalog does not support attribute creation IDs, you use the attribute value itself and the attribute type to identify an attribute value. An **attribute tag** indicates the format of the attribute value. Apple Computer has defined a few attribute tags for use by Apple's PowerShare catalogs; developers of catalog service access modules can define their own attribute tags to support collaborative applications. There is a maximum size for attribute values stored in PowerShare and personal catalogs, but there are no restrictions on their content. The `Attribute`, `AttributeType` and `AttributeValue` data types are described in the chapter "AOCE Utilities" in this book.

Aliases and Pseudonyms

Some catalogs support the use of alternative names (or **pseudonyms**) for catalog records. For example, the record "Sally Simon" might have the pseudonym "Simon, Sally". You can use a pseudonym in any Catalog Manager routine that requires a record name.

The Catalog Manager also allows you to create aliases for records. A **record alias** is itself a record and so can be placed in any catalog. The Catalog Manager creates the record and marks it as an alias. It is up to you to store the information that your application requires to resolve the alias. For example, you might store the record location information for the original record in an attribute value in the alias record. Not all catalogs support the ability to create aliases.

Access Controls

The Catalog Manager defines access controls for dNodes, records, and attribute types. Types of access privilege include the ability to add and delete attribute types within records, to see attribute values, to add and delete records within dNodes, and so forth. Service requesters are either authenticated, that is, represented by a record in the catalog that contains the dNode, record, or attribute type to which they seek access, or they are guests. There are several categories of authenticated requesters. Access controls are discussed in more detail in the section "Getting Access Controls" on page 8-11.

PowerShare catalogs support a full range of access controls. External catalogs can support any level of access controls up to the full set defined by the Catalog Manager. Personal catalogs do not support Catalog Manager access controls as such. Instead, personal catalogs derive their access controls from the read and read/write privileges allowed by the Macintosh file system. However, you can use the Catalog Manager to read the access controls for a personal catalog. In that case, the Catalog Manager maps file system settings into its own access control settings.

Identities and the PowerTalk Setup Catalog

The PowerTalk system software creates a special personal catalog called the PowerTalk Setup catalog. The **PowerTalk Setup catalog** contains information about the catalogs and electronic mail systems that are available to the principal user of the computer. The PowerTalk Setup catalog is a personal catalog stored on the user's local disk. The records in the PowerTalk Setup catalog represent, among other things, PowerShare catalogs, external catalogs, and catalog service access modules. Catalogs and catalog service access modules represented by records in the PowerTalk Setup catalog are said to be "listed in PowerTalk Setup." The Catalog Manager provides routines that allow you to add and remove records from this and other personal catalogs.

Most Catalog Manager routines take an identity as an input. An **identity** is a number derived from a user name and password.

A "master" name and password protect the information in the PowerTalk Setup catalog. When a user enters his or her name and password after starting up PowerTalk, the Authentication Manager transforms the name and password into a special value called the local identity. The **local identity** is a "master" identity that provides you with transparent access to all of the specific names and passwords stored in the PowerTalk Setup catalog. You can obtain the local identity by calling the Authentication Manager's `AuthGetLocalIdentity` function.

There is another type of identity called specific identity. A **specific identity** is derived from the name and password of a user who has an account on a specific server. This user can be the principle user of the computer or an alternate user (or *visitor*). Specific identities make it possible for several people to use the same Macintosh to gain access to their PowerShare catalog services. You can obtain a specific identity by calling the Authentication Manager's `AuthBindIdentity` function.

The identity that you provide is used to determine if the requester is authorized to make the service request. In any Catalog Manager function, you may specify either a local identity, a specific identity, or 0, which indicates guest access. If you specify the local identity, you do not need to know the requester's specific identity for a particular catalog. The Catalog Manager uses the local identity to obtain the specific identity before processing the request.

PowerShare catalogs require an identity for most service requests; external catalogs may not. Personal catalogs do not require an identity with service requests.

For more information about the PowerTalk Setup catalog, see the chapter "Service Access Module Setup" in *Inside Macintosh: AOCE Service Access Modules*. For more

information about local identity and specific identities, see the chapter “Authentication Manager” in this book.

About the Catalog Manager

The Catalog Manager, the Interprogram Messaging Manager, and the Authentication Manager together constitute the fundamental AOCE services. The Catalog Browser and the Standard Catalog Package provide high-level interfaces to the Catalog Manager, and catalog service access modules provide a way for developers to extend AOCE catalog services to external catalogs. See the chapter “Introduction to the Apple Open Collaboration Environment for a description of the position of the Catalog Manager within the AOCE software architecture.

The Catalog Manager includes routines that provide the following services:

- getting information about catalogs, including the catalogs that are listed in the PowerTalk Setup catalog, getting information about a specific catalog’s capabilities, getting information about the icons that represent a specific external catalog, and obtaining the name of the network in which a catalog is located
- getting information about a catalog hierarchy, including enumerating dNodes, mapping dNode numbers to pathnames and vice versa, detecting changes in dNodes, and getting information about a specific dNode
- managing the PowerTalk Setup catalog, including listing a PowerShare catalog in PowerTalk Setup, removing a PowerShare catalog from PowerTalk Setup, and searching a network for PowerShare catalogs that you want to use
- creating, opening, and closing personal catalogs
- managing records, including adding and deleting records and aliases, listing records and aliases, getting and setting the name and type of a record, getting information about a record, and adding, deleting, and listing pseudonyms
- managing attribute values and types, including adding and deleting attribute values, changing attribute values, looking for specific attribute values, looking up attribute values, and listing attribute types
- determining access to dNodes, records, and attribute types

Note that the Catalog Manager API does not provide routines for catalog configuration and administration that allow you to create a catalog, to name or rename a catalog, to add and delete nodes, and so forth. These functions, unique to each type of catalog, are handled by the catalog’s administration software and are beyond the scope of the Catalog Manager.

Get/Parse Function Pairs

The Catalog Manager API supplies several get / parse function pairs that work together to provide you with information about dNodes, records, access controls, and so forth. The “get” routine of each of these pairs writes the data in a format that is private to the

Catalog Manager into a buffer that you supply. The corresponding "parse" routine extracts the data from the buffer and passes it in logical chunks to a callback routine that you supply.

For example, the `DirEnumerateDirectoriesGet` function stores in a buffer information about all of the catalogs that are listed in the PowerTalk Setup catalog. The `DirEnumerateDirectoriesParse` function parses that information and calls your callback routine for each catalog about which there is information in the buffer. Each time it calls your callback routine, the parse function passes it a catalog name, the catalog discriminator, and information about the features supported by the catalog.

Callback Routines

When you call a Catalog Manager parse function, you pass it a pointer to a callback routine that you provide. If you call the parse function synchronously, the same execution environment (low-memory global variables, A5 world, stack, interrupt state, and any programming restrictions) that was in effect when the Catalog Manager began executing the parse function is also in effect when your callback routine is executed. Therefore, if it is safe to allocate memory or make synchronous calls when you call the parse routine, then your callback routine can also allocate memory or make synchronous calls.

If you call the parse function asynchronously, it saves only the A5 world and restores it when it calls your callback routine. In this case you have access to your application's global variables, but you cannot allocate memory or make synchronous calls.

Callback routines should not call Catalog Manager functions, call the `WaitNextEvent` or `SystemTask` routines, invoke the Notification Manager, or call any function that calls any of these routines.

One of the parameters a parse function passes to your callback routine is the value you placed in the `clientData` field of the parse function's parameter block. You can use this value for whatever purpose you wish; for example, you can use it to distinguish between asynchronous parse requests if you have more than one pending completion or use it to point to a private data area.

The parameters that a parse function passes to a callback routine are described under each routine in the section "Application-Defined Functions" beginning on page 8-150.

Every callback routine returns a Boolean result. If you want the parse function to continue parsing the data in your buffer, return `false`; otherwise, return `true`.

Determining Features Supported

A catalog may not support all the features of the Catalog Manager API. You call the `DirGetDirectoryInfo` function to determine the features that a catalog supports before calling other Catalog Manager functions that address that catalog. The feature information is specified in a feature flag bit array. The bits are defined in "Feature Flag Bit Array" beginning on page 8-28.

Getting Access Controls

The information discussed in this section applies primarily to PowerShare catalogs. Access controls for personal catalogs consist of read and read / write settings implemented by the File Manager. These are mapped into the Catalog Manager access control privileges when you ask for the access controls for a personal catalog.

Three interrelated components to the access controls are available through the Catalog Manager. The first component is the container whose access is controlled. Consider dNodes, records, and attribute types as sets of nested abstract containers. DNodes may contain records, aliases, pseudonyms and other dNodes; records may contain attribute types; and attribute types may contain attribute values. The second component is the requester seeking access to a container. The third component is the kind of access privilege that the requester seeks. DNodes, records, and attribute types are discussed in "Access Controls" on page 8-7. Requesters and access privileges are discussed in the following sections.

Types of Requesters

PowerShare catalogs classify all requesters into five categories. Each dNode, record, and attribute type maintains a set of access privileges for each of the categories. A single requester may fall into one or more categories.

An external catalog, by contrast, does not necessarily use requester categories. It may maintain access privileges for each individual requester. Alternatively, it may use categories different from those used by PowerShare catalogs.

You use a variable of type `CategoryMask` to specify the type of requestor about which you want information.

```
typedef unsigned long CategoryMask;
```

The bits in the `CategoryMask` data type are defined as follows:

```
enum {
   kThisRecordOwnerBit           =  0,
   kFriendsBit                   =  1,
   kAuthenticatedInDNodeBit      =  2,
   kAuthenticatedInDirectoryBit  =  3,
   kGuestBit                     =  4,
   kMeBit                        =  5
};
```

You can use the following values to set and test the bits in a variable of type `CategoryMask`.

```
enum {         /* Values of CategoryMask */
   kThisRecordOwnerMask          = (1L << kThisRecordOwnerBit),
   kFriendsMask                  = (1L << kFriendsBit),
```

```
kAuthenticatedInDNodeMask     = (1L << kAuthenticatedInDNodeBit),
kAuthenticatedInDirectoryMask = (1L << kAuthenticatedInDirectoryBit),
kGuestMask                    = (1L << kGuestBit),
kMeMask                       = (1L << kMeBit)};
```

Descriptions

`kThisRecordOwnerMask`
A requester in this category is the owner of the record or attribute type to which the requester wants access. (This category has no meaning at the dNode level.) At most, only one requester can belong in this category for each record or attribute type. The owner of a record is the person or process represented by the record. The owner of an attribute type is the person or process represented by the record that contains the attribute type. The creation ID of the requester's own record is the same as the creation ID of the record to which access is sought. (Or the record names and record types are the same in catalogs that do not support creation IDs.)

`kFriendsMask` A requester in this category is specially selected and may have different (usually broader) access privileges to a dNode, record, or attribute type than those available to requesters who belong to the more general categories. For PowerShare catalogs, the attribute type `kOwnersAttrTypeNum` is defined to identify persons or processes as friends. An attribute value of attribute type `kOwnersAttrTypeNum` is a `DSSpec` data structure that contains a record ID. Every requester represented by such a value in the `kOwnersAttrTypeNum` attribute type belongs by definition to the friends category. You can add a person or process to the friends category by adding a value specifying that person or process to the `kOwnersAttrTypeNum` attribute type. Note that to do this you need a level of access privilege that allows you to change the access control privileges for a dNode, record, or attribute type as well as to add values. In PowerShare catalogs, the requesters in the friends category for any attribute type within a record are exactly the same as the requesters in the friends category for the record itself.

`kAuthenticatedInDNodeMask`
A requester in this category is represented by a record located in the same dNode as the dNode, record, or attribute type to which the requester wants access.

`kAuthenticatedInDirectoryMask`
A requester in this category is represented by a record located in the same catalog as the dNode, record, or attribute type to which the requester wants access.

`kGuestMask` A requester in this category is not represented by a record that resides in the same catalog as the dNode, record, or attribute type that the requester wants to access.

`kMeMask` This is a quasi-category called “me.” The `DirGet`*xxx*`AccessControlGet` functions provide it as an output category when the requester asks only for information on his or her own access privileges. It is a convenient way of providing access privilege information that pertains to the requester regardless of the categories to which that requester belongs.

Types of Access Privileges

The Catalog Manager defines different kinds of access privileges. These kinds of privileges are specified by the access control bit masks that are described next.

Mask descriptions

`kNoPrivs` This mask specifies that the requester has no access to a dNode, a record, or an attribute type.

`kSeeMask` When the container is a dNode, this mask specifies the ability to view the records, aliases, and pseudonyms of the dNode. When the container is a record, this mask specifies the ability to view the contents of the record. When the container is an attribute type, this mask specifies the ability to view the contents of the attribute type.

`kAddMask` When the container is a dNode, this mask specifies the ability to add records, aliases, pseudonyms, and dNodes to the dNode. When the container is a record, this mask specifies the ability to add attribute types to the record. When the container is an attribute type, this mask specifies the ability to add values to the attribute type.

`kDeleteMask` When the container is a dNode, this mask specifies the ability to delete records, aliases, pseudonyms, and dNodes from the dNode. When the container is a record, this mask specifies the ability to delete attribute types from the record. When the container is an attribute type, this mask specifies the ability to delete values from an attribute type.

`kChangeMask` When the container is a record, this mask specifies the ability to change the contents of the record; that is, to replace some or all of the record’s contents without changing the record’s creation ID. When the container is an attribute type, this mask specifies the ability to change the contents of an attribute type; that is, to replace an attribute value without changing the attribute creation ID. Changing the contents of a dNode is undefined and not supported.

`kRenameMask` When the container is a dNode or a record, this mask specifies the ability to rename it by changing its record name and record type. Renaming an attribute type is not supported.

`kChangePrivsMask`
: This mask specifies the ability to change the access control privileges for a dNode, a record, or an attribute type. The ability to change access privileges for a container includes the ability to change privileges for the content of the container as well. If you can change access privileges for a dNode, you can also change access privileges for the records and attribute types within the dNode. Likewise, if you can change access privileges for a record, you can also change access privileges for the attribute types within the record.

`kSeeFoldersMask`
: When the container is a dNode or a record, this mask specifies the ability to view dNodes within the container dNode.

`kAllPrivs`
: This mask specifies the sum of all the specific access privileges just described for a dNode, a record, or an attribute type.

Access Control Lists

PowerShare catalogs maintain an access control list for each dNode, record, and attribute type in the catalog. Each entry in an access control list specifies a category and the category's access privileges with respect to that dNode, record, or attribute type. Each access control list consists of five entries, one for each category.

The Catalog Manager API, however, does not restrict access control lists to the PowerShare implementation. An external catalog may maintain access control lists that consist of individual requesters and their access privileges, instead of categories of requesters. Or it may consist of the PowerShare categories or different categories or some combination of these. The access control list of an external catalog may have any number of entries.

A personal catalog has a single entry in its access control list to which every requester belongs. For a personal catalog, every requester has exactly the same access privileges.

Regardless of the type of access control list that a catalog maintains, all catalogs should be able to provide a requester with the access privileges that apply to that requester. (This is the quasi-category "me" for PowerShare catalogs.)

PowerShare catalogs implement access controls for most catalog service requests. The PowerShare catalog identifies the categories to which the requester belongs and determines if a requester in those categories has sufficient access to perform the requested action.

The Catalog Manager provides a get/parse pair of functions for each type of container so that you can obtain access control information about the container. A `DirGet`*xxx*`AccessControlGet` function obtains access control information from the access control list associated with a dNode, record, or attribute type and stores the information in a buffer that you provide. A `DirGet`*xxx*`AccessControlParse` function retrieves information for one access control list entry at a time from your buffer and passes it to your callback routine. You can request access control information for every entry on the access control list, for a subset of entries, or for only the requester (specified in the `identity` field of the `DirParamBlock` parameter block).

Note
The utility routine `OCEGetAccessControlDSSpec` converts a category mask into the catalog service specification that you need to pass to a `DirGet`*xxx*`AccessControlGet` function. This routine is described on page 8-132. u

For PowerShare catalogs and personal catalogs, the information that a `DirGet`*xxx*`AccessControlParse` function passes to your callback routine consists of a catalog service specification that identifies a category plus the access control masks that apply to that category. The access control masks may be described as either active or default. An active access control mask is a mask that currently applies to a dNode, record, or attribute type; it specifies which operations a requester in the category is authorized to perform on the dNode, record, or attribute type. A default access control mask is the mask that is applied to new objects within the container at the time they are created. For example, any new records that are created within a dNode automatically acquire the access controls that are specified by the dNode's default access control mask for records.

When you request dNode access control information, you get the active access control mask for the dNode as well as the default access control masks that apply to newly created records and attribute types within the dNode. When you request record access control information, you get the active access control masks for the record and the dNode containing the record. You also get the default access control mask that applies to newly created attribute types within the record. When you request attribute type access control information, you get the active access control masks for the attribute type as well as the record and the dNode containing the attribute type.

Using the Catalog Manager

The Catalog Manager API supplies several get / parse function pairs that work together to provide you with information about dNodes, records, access controls, and so forth. The "get" routine of each of these pairs writes the data in a format that is private to the Catalog Manager into a buffer that you supply. The corresponding "parse" routine extracts the data from the buffer and passes it in logical chunks to a callback routine that you supply.

If the initial buffer size is not sufficient to hold all the data, there are two different ways in which the get / parse function pairs work—depending on which Catalog Manager routines are called. The first example, "Getting Attribute Value Information" beginning on page 8-16 shows how the `DirLookupGet` and `DirLookupParse` work together to extract information. These two functions use identical parameter blocks They are the only get / parse function pair that work this way.

The example, "Getting Attribute Type Information" beginning on page 8-20 illustrates how all other get / parse function pairs work.

There is also one "get" function, `DirGetExtendedDirectoryInfo`, that has no corresponding "parse" routine. In this case, you must write your own "parse" routine. See "Getting Extended Catalog Information" beginning on page 8-24 for an example that shows how to do this.

Determining Whether the Collaboration Toolbox Is Available

Before calling any of the Catalog Manager functions, you 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" of *Inside Macintosh: Operating System Utilities*.

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

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 you are using the Collaboration toolbox on a computer that has a PowerShare server, the function returns the version number of the server in the high-order word of the `response` parameter. If the Collaboration toolbox or server is not present and available, the `Gestalt` function returns 0 for the relevant version number. You can use the constant `gestaltOCETB` for AOCE Collaboration toolbox version 1.0.

Getting Attribute Value Information

The `DoProcessAttributeValues` function in Listing 8-1 lists the attribute values for a particular catalog. It uses two Catalog Manager routines: `DirLookupGet` (page 8-118) and `DirLookupParse` (page 8-121). Because these two routines have the same parameter block, you can call `DirLookupGet` as many times as necessary, using the same parameter block you last passed to `DirLookupParse`, if the buffer you allocate is too small to hold all the data that `DirLookupGet` returns. Listing 8-2 on page 8-21 shows how to use a pair of get/parse functions that have different parameter blocks.

When DoProcessAttributeValues is called, it is passed two parameters, a pointer to the catalog browser and a pointer to the current attribute type. It then calls the DoEnumerateAttributeValues function, passing it parameters to specify the identification of the requester; to identify the current catalog, record and attribute type; to name a callback routine; and to match calls to this routine to particular calls to the DirLookupParse function.

When DoEnumerateAttributeValues returns, DoProcessAttributeValues calls DoSelectNthAttributeValue—which is not shown here—to extract the current attribute value and display its contents.

The DoEnumerateAttributeValues function sets up the parameter block that DirLookupGet uses and then sets the initial values. It includes a do / while loop in which DirLookupGet and DirLookupParse do the work of extracting the attribute value information. If the buffer is too small to hold all the data, the DirLookupParse function returns kOCEMoreData, and the loop repeats. The same parameter block that was passed to DirLookupParse is passed on subsequent calls to DirLookupGet. The reason this works is that when DirLookupParse completes, it returns values in the lastRecordIndex, lastAttributeIndex, and lastAttribute fields that are at the same offsets in the parameter block as the values of the startingRecordIndex, startingAttrTypeIndex, and startingAttribute fields on a subsequent call to the DirLookupGet function. The DirLookupGet function continues retrieving information from the point at which it stopped during its previous invocation.

The DirLookupParse function calls the callback routine MyForEachAttrValue for each attribute value that it finds. The callback routine calls the DoAddAttribueValue function—which is not shown here—passing it the data structure containing the attribute values. The DoAddAttribueValue function stores the values.

Listing 8-1 Listing the attribute values for a catalog

```
/* Enumerate all attribute types for the currently-selected
   catalog. Attribute types are added to the type list as
   they are found. */

static pascal Boolean      MyForEachAttrValue(
      long                    clientData,
      const Attribute         *theAttribute
   );

/* DoProcessAttributeValues is called when a new attribute type
   is selected.

   Globals
   DOC.currentDsRefNum
        Current personal directory RefNum
   DOC.currentRecordID
```

```
         Current record to examine */

void
DoProcessAttributeValues(
      register CatalogBrowserPtr       dbp,
      const AttributeTypePtr           attributeTypePtr
   )
{
      OSErr                            status;

       /* Make sure to start with a clean slate.*/

      ClearAttributeValueList(dbp);
      status = DoEnumerateAttributeValues(
               DOC.userIdentity,
               DOC.currentDsRefNum,
               &DOC.currentRecordID,
               attributeTypePtr,
               MyForEachAttrValue,
               (long) dbp
            );
      LOG(status, "\pDoEnumerateAttributeValues");
      DoSelectNthAttributeValue(dbp, 0);
}

/* MyForEachAttrValue is called by the DirLookupParse function.
   The attribute value is an RString that is put into the list. */

static pascal Boolean
MyForEachAttrValue(
      long                             clientData,
      const Attribute                  *theAttribute
   )
{
      register CatalogBrowserPtr       dbp;
      Boolean                          stopParse;

      dbp = (CatalogBrowserPtr) clientData;
      stopParse = FALSE;
      TRY {
         AddAttributeValue(dbp, theAttribute);
      }
      CATCH {
```

```
         LOG(STATUS, "\pCan't add attribute value");
         NO_PROPAGATE;
         stopParse = TRUE;
      }
      ENDTRY;
      return (stopParse);
}

#ifndef kMyBufferSize
#define kMyBufferSize 4096
#endif

/* DoEnumerateAttributeValues is called when a new attribute type
   is selected. */

OSErr
DoEnumerateAttributeValues(
      AuthIdentity              userIdentity,
      short                     dsRefNum,
      RecordIDPtr               recordIDPtr,
      const AttributeTypePtr    theAttributeType,
      ForEachAttrValue          MyForEachAttrValue,
      long                      clientData
   )
{
      OSErr                     status;
      AttributeTypePtr          attrTypeList[1];
      RecordIDPtr               recordIDList[1];
      DirParamBlock             dirParamBlock;
      Ptr                       myBuffer;
#define GET    (dirParamBlock.lookupGetPB)
#define PARSE  (dirParamBlock.lookupParsePB)

      myBuffer = NewPtr(kMyBufferSize);
      if (myBuffer == NULL)
         status = MemError();
      else {
         CLEAR(dirParamBlock);
         recordIDList[0] = recordIDPtr;
         attrTypeList[0] = theAttributeType;
         GET.identity = userIdentity;
         GET.ioCompletion = NULL;
         GET.dsRefNum = dsRefNum;
         GET.clientData = clientData;
```

```
            GET.aRecordList = recordIDList;
            GET.attrTypeList = attrTypeList;
            GET.recordIDCount = 1;
            GET.attrTypeCount = 1;
            GET.includeStartingPoint = FALSE;
            GET.getBuffer = myBuffer;
            GET.getBufferSize = kMyBufferSize;
            GET.startingRecordIndex = 1;
            GET.startingAttrTypeIndex = 1;
            CLEAR(GET.startingAttribute);
            do {
               status = DirLookupGet(&dirParamBlock, SYNC);
               if (status == noErr || status == kOCEMoreData) {
                  PARSE.eachRecordID = NULL;
                  PARSE.eachAttrType = NULL;
                  PARSE.eachAttrValue = MyForEachAttrValue;
                  status = DirLookupParse(&dirParamBlock, SYNC);
               }
            } while (status == kOCEMoreData);
            DisposePtr(myBuffer);
         }
         return (status);
#undef GET
#undef PARSE
}
```

Getting Attribute Type Information

The routines in Listing 8-2 return the attribute types for a specified catalog. They use the Catalog Manager `DirEnumerateAttributeTypesGet` (page 8-127) and `DirEnumerateAttributeTypesParse` (page 8-130) functions. As the example shows, if the buffer is too small to hold all the data returned by `DirEnumerateAttributeTypesGet`, it can be called again in a loop, using the last attribute type parameter that `DirEnumerateAttributeTypesParse` passed to the callback routine. Listing 8-1 on page 8-17 shows how a different get / parse pair work together.

The structure `CallBackData` is used to hold data including the current attribute type.

The `DoEnumerateAttributeTypes` function is called by a higher-level routine and is passed parameters to authenticate the user; identify the catalog, the record, and the current attribute; and to match calls to this routine to particular calls to the `DirEnumerateAttributeTypesParse` function. It then allocates a buffer and sets up the parameter block with initial values.

The DoEnumerateAttributeTypes function contains a do/while loop that enables DirEnumerateAttributeTypesGet and DirEnumerateAttributeTypesParse to extract the attribute type information. For each attribute type extracted, DirEnumerateAttributeTypesParse calls the MyEnumerateEachAttributeType callback routine.

If the buffer is too small to hold all the information returned by DirEnumerateAttributeTypesGet, the loop repeats. The DirEnumerateAttributeTypesGet function uses as its starting attribute type, the last attribute type that DirEnumerateAttributeTypesParse passed to the callback routine.

The callback routine, MyEnumerateEachAttributeType, provides a data type to store the attribute type information extracted by DirEnumerateAttributeTypesParse. It also stores the last attribute type that it received from DirEnumerateAttributeTypesParse in case it needs to pass it back to DirEnumerateAttributeTypesGet for another run through the loop.

Listing 8-2 Listing the attribute types for a catalog

```
 /* Enumerate all attribute types for the specified catalog. The caller
    provides a callback function (which takes the same parameters as the AOCE
    DirEnumerateAttributeTypesParse function) that is called with each
    returned attribute type. */

#ifndef kMyBufferSize
#define kMyBufferSize4096
#endif

static pascal Boolean               MyEnumerateEachAttributeType(
      long                             clientData,
      const AttributeType              *aType
   );

/* This data is passed to MyEnumerateEachAttributeType */
 typedef struct CallBackData {
   ForEachAttrType                  eachAttrType;
   long                             clientData;
   AttributeType                    currentAttrType;
} CallBackData, *CallBackDataPtr;

/* DoEnumerateAttributeTypes is called when a new record is selected. It
   calls a user function for each attribute type stored in that record. */

OSErr
```

```
DoEnumerateAttributeTypes(
      AuthIdentity                  userIdentity,
      short                         dsRefNum,
      RecordIDPtr                   recordIDPtr,
      ForEachAttrType               eachAttrType,
      long                          clientData
   )
{
      OSErr                         status;
      Boolean                       first;
      CallBackData                  callBackData;
      DirParamBlock                 dirParamBlock;
      Ptr                           myBuffer;
#define GET    (dirParamBlock.enumerateAttributeTypesGetPB)
#define PARSE  (dirParamBlock.enumerateAttributeTypesParsePB)

      myBuffer = NewPtr(kMyBufferSize);
      if (myBuffer == NULL)
         status = MemError();
      else {
         callBackData.eachAttrType = eachAttrType;
         callBackData.clientData = clientData;
         CLEAR(callBackData.currentAttrType);
         CLEAR(dirParamBlock);
         first = TRUE;
         do {
            GET.identity = userIdentity;
            GET.dsRefNum = dsRefNum;
            GET.clientData = (long) &callBackData;
            GET.aRecord = recordIDPtr;
            if (first) {
               GET.startingAttrType = NULL;
               first = FALSE;
            }
            else {
               /* This is the last attribute type that was fetched
                  by the parser callback. */
               GET.startingAttrType = &callBackData.currentAttrType;
            }
            GET.includeStartingPoint = FALSE;
            GET.getBuffer = myBuffer;
            GET.getBufferSize = kMyBufferSize;
            status = DirEnumerateAttributeTypesGet(&dirParamBlock, SYNC);
```

```
            if (status != kOCEMoreData)
               LOG(status, "\pDirEnumerateAttributeTypesGet");
            if (status == noErr || status == kOCEMoreData) {

                /* There is a record, or there is a record and more
                   data to read. Parse the data: this will call
                   the callback function.*/
               PARSE.eachAttrType = MyEnumerateEachAttributeType;
               status = DirEnumerateAttributeTypesParse(&dirParamBlock,
                        SYNC);
               if (status != kOCEMoreData)
                  LOG(status, "\pDirEnumerateAttributeTypesParse");
            }
         } while (status == kOCEMoreData);
         DisposePtr(myBuffer);
      }
      return (status);
#undef GET
#undef PARSE
}

/* MyEnumerateEachAttributeType is called by the
   DirEnumerateAttributeTypesParse function. Remember the attribute type for
   the next call and call the application handler. */

static pascal Boolean
MyEnumerateEachAttributeType(
      register long                    clientData,
      const AttributeType              *aType
   )
{
      Boolean                             stopParse;
#define CALLBACK(*((CallBackDataPtr) clientData))

       /* Grab a copy of the attribute type for the next "get more
          data" call. */
      OCECopyRString(
         (const RStringPtr) aType,
         (RStringPtr) &CALLBACK.currentAttrType,
         kAttributeTypeMaxBytes
      );
```

```
    stopParse = CALLBACK.eachAttrType(CALLBACK.clientData, aType);
    return (stopParse);
}
```

Getting Extended Catalog Information

The `DirGetExtendedDirectoriesInfo` function (page 8-54) returns extended information about a catalog. The `DirGetExtendedDirectoriesInfo` function returns a packed structure that you must unpack. The sample routines in Listing 8-3 show how to call the `DirGetExtendedDirectoriesInfo` function and how to examine the information it returns.

The sample routines make use of the structure type, `MyExtendedInfoType`, which can hold the extended information for a single catalog.

The `DoProcessExtendedCatalogInfo` routine declares two pointers: `myBufferPtr`, the pointer to the data buffer that will be passed to the `DirGetExtendedDirectoriesInfo` function, and `extendedInfoPtr`, a pointer to an extended information structure (`MyExtendedInfo`). It sets both pointers to `nil`.

Next, the function calls the `DoGetExtendedCatalogInfo` routine to get the extended information from the Catalog Manager. If the routine returns successfully, `DoProcessExtendedCatalogInfo` allocates enough memory to store the extended information for all of the catalogs on which the Catalog Manager has returned information.

Then `DoProcessExtendedCatalogInfo` calls the `DoUnpackExtendedCatalogInfo` routine to extract the extended information from the data buffer and put it in the array of extended information structures. The `DoUtilizeExtendedCatalogInformation` routine, not shown here, acts on the extended information. You would include a similar routine to do whatever is appropriate to your application. Finally, `DoProcessExtendedCatalogInfo` disposes of the memory that has been allocated before it returns.

Unlike some Catalog Manager functions, `DirGetExtendedDirectoriesInfo` cannot be called to retrieve a portion of the information and then called again to retrieve more. It always attempts to return all of the information at once, and it completes with the `kOCEMoreData` result code if the buffer you pass is too small. The purpose of the `DoGetExtendedCatalogInfo` routine is to call the `DirGetExtendedDirectoriesInfo` function with a buffer that is large enough to hold all the information that `DirGetExtendedDirectoriesInfo` will return. The `DoGetExtendedCatalogInfo` routine allocates a buffer of `kWorkBufferSize` bytes and then calls the `DirGetExtendedDirectoriesInfo` function. If `DirGetExtendedDirectoriesInfo` function returns `kOCEMoreData`, `DoGetExtendedCatalogInfo` disposes of the buffer and allocates a larger one. It does this repeatedly, increasing the size of the buffer in increments of `kWorkBufferSize` bytes until the buffer is large enough to contain all the information `DirGetExtendedDirectoriesInfo` can return. If `DirGetExtendedDirectoriesInfo` completes successfully,

DoGetExtendedCatalogInfo passes back via its actualEntriesPtr parameter the number of catalogs for which extended information now exists in the buffer. Otherwise, it disposes of the buffer.

The DoUnpackExtendedCatalogInfo routine extracts the extended information about each catalog from the buffer and stores the information in MyExtendedInfo structures. It does the same thing as the "parse" routines in Listing 8-1 on page 8-17 and Listing 8-2 on page 8-21, but you have to write this routine yourself because DirGetExtendedDirectoriesInfo has no corresponding "Parse" routine. Note that variables in an extended information structure point to data in the packed buffer.

Because the data in the buffer is of variable length, the sizeof function is required to determine the length of the data. The INCR macro aligns the data on a word boundary.

Listing 8-3 Getting extended information for a catalog

```
typedef struct MyExtendedInfo {
   PackedRLIPtr          pRLIPtr;      /* Catalog's packed RLI     */
   unsigned short        pRLILength;   /* Length of packed RLI     */
   OSType                entnType;     /* Catalog address type     */
   long                  hasMailSlot;  /* Nonzero if mail slot    */
   RStringPtr            realName;     /* Catalog's true name      */
   RStringPtr            comment;      /* More info for display    */
   long                  dataLength;   /* Additional data length   */
   Ptr                   dataPtr;      /* Additional information   */
} MyExtendedInfoType, *MyExtendedInfoPtrType;

OSErr DoProcessExtendedCatalogInfo(void) {
   OSErr                   status;
   Ptr                     myBufferPtr;
   MyExtendedInfoPtrType   extendedInfoPtr;
   unsigned long           actualEntries;

   myBufferPtr = nil;
   extendedInfoPtr = nil;
   status = DoGetExtendedCatalogInfo(&myBufferPtr, &actualEntries);
   if (status == noErr) {
      extendedInfoPtr = (MyExtendedInfoPtrType) NewPtrClear(actualEntries *
                        sizeof(MyExtendedInfo));
      status = MemError();
   }
   if (status == noErr) {
      status = DoUnpackExtendedCatalogInfo(myBufferPtr, extendedInfoPtr,
                                           actualEntries);
```

```
      status = DoUtilizeExtendedCatalogInformation(extendedInfoPtr,
                                                   actualEntries);
   }

   if (extendedInfoPtr != nil)
      DisposePtr((Ptr) extendedInfoPtr);
   if (myBufferPtr != nil)
      DisposePtr((Ptr) myBufferPtr);
   return (status);
}
OSErr DoGetExtendedCatalogInfo(
   Ptr               *resultBuffer,       /* address of ptr to buffer */
   unsigned long     *actualEntriesPtr)

{
#define kWorkBufferSize (512)
   OSErr             status;
   DirParamBlock     myParamBlock;
   unsigned long     bufferLength;
#define GET    (myParamBlock.getExtendedDirectoriesInfoPB)
   bufferLength = 0;
   *resultBuffer = nil;
   do {
      if (*resultBuffer != nil)
         DisposePtr(*resultBuffer);
      bufferLength += kWorkBufferSize;
      *resultBuffer = NewPtr(bufferLength);
      if ((status = MemError()) != noErr)
         break;
      ClearMemory(&myParamBlock, sizeof myParamBlock);
      GET.identity = gUserIdentity;
      GET.buffer = *resultBuffer;
      GET.bufferSize = bufferLength;
      status = DirGetExtendedDirectoriesInfo(&myParamBlock, false);
   } while (status == kOCEMoreData);
   if (status == noErr)
      *actualEntriesPtr = GET.actualEntries;
   else if (*resultBuffer != nil) {
      DisposePtr(*resultBuffer);
      *resultBuffer = nil;
   }
   return (status);
#undef GET
}
```

```
OSErr DoUnpackExtendedCatalogInfo(
   register Ptr                     bufPtr,          /* pointer to buffer
                                                        containing packed
                                                        extended catalog
                                                        information */
   register MyExtendedInfoPtrType   extendedInfoPtr,  /* pointer to array
                                            of MyExtendedInfo structures */
   unsigned long                    actualEntries)
{
   unsigned long  dataLength;       /* working value  */
   unsigned long  i;                /* current entry count */

      /* Scan through the buffer to extract the extended catalog
         information. The INCR macro increments bufPtr by some amount,
         making sure that it is aligned on a word boundary. Its argument
         must not have side-effects.*/
#define INCR(v)   (bufPtr += ((v) + ((v) & 0x01)))
#define RESULT    (*extendedInfoPtr)

   for (i = 0; i < actualEntries; i++, extendedInfoPtr++) {
      RESULT.pRLIPtr = (PackedRLIPtr) bufPtr;
      RESULT.pRLILength = pRLIPtr->dataLength;
      INCR(pRLILength + sizeof (ProtoPackedRLI));
      RESULT.entnType = *((OSType *) bufPtr);
      INCR(sizeof (OSType));
      RESULT.hasMailSlot = *((long *) bufPtr);
      INCR(sizeof (long));
      RESULT.realName = (RStringPtr) bufPtr;
      dataLength = RESULT.realName->dataLength;
      INCR(dataLength + sizeof (ProtoRString));
      RESULT.comment = (RStringPtr) bufPtr;
      dataLength = RESULT.comment->dataLength;
      INCR(dataLength + sizeof (ProtoRString));
      RESULT.dataLength = *((long *) bufPtr);
      INCR(sizeof (long));
      RESULT.dataPtr = (Ptr) bufPtr;
      INCR(RESULT.dataLength);          /* Step over the rest */
   }
#undef INCR
#undef RESULT
}
```

Catalog Manager Reference

This section describes the feature flag bit array, and the data types and functions provided by the Catalog Manager.

Feature Flag Bit Array

Each catalog provides information so that you can determine which features it supports. This information is specified in a feature flag bit array. The bits are defined next.

Bit name

```
kSupportsDNodeNumberBit
kSupportsRecordCreationIDBit
kSupportsAttributeCreationIDBit
kSupportsMatchAllBit

kSupportsBeginsWithBit
kSupportsExactMatchBit
kSupportsEndsWithBit
kSupportsContainsBit
kSupportsOrderedEnumerationBit
kCanSupportNameOrderBit

kCanSupportTypeOrderBit
kSupportSortBackwardsBit
kSupportIndexRatioBit
kSupportsEnumerationContinueBit
kSupportsLookupContinueBit

kSupportsEnumerateAttributeTypeContinueBit
kSupportsEnumeratePseudonymContinueBit
kSupportsAliasesBit
kSupportsPseudonymsBit

kSupportsPartialPathNamesBit
kSupportsAuthenticationBit
kSupportsProxiesBit
kSupportsFindRecordBit
```

Bit descriptions

`kSupportsDNodeNumberBit`
: If this bit is set, you can reference a dNode by using a dNode number in the `RLI` data structure and setting the pathname pointer to `nil`. If this bit is not set, you can reference a dNode only by specifying its pathname information in the `RLI` data structure; in this case, you must set the dNode number to 0.

`kSupportsRecordCreationIDBit`
: If this bit is set, you can reference a record by specifying its record creation ID for most Catalog Manager functions. If this bit is not set, you must reference a record by specifying its record name and record type in its record ID.

`kSupportsAttributeCreationIDBit`
: If this bit is set, you can reference an attribute value by specifying its attribute creation ID and attribute type.

The next five bits indicate what combination of browsing, finding, and matching capabilities a catalog supports when you enumerate the contents of a dNode in that catalog.

`kSupportsMatchAllBit`
: If this bit is set, a catalog supports browsing of record names and record types. When you call the `DirEnumerateGet` function, such a catalog can accept an enumeration specification with the `matchNameHow` and `matchTypeHow` fields set to `kMatchAll`, in which case, a search matches any record name or record type.

`kSupportsBeginsWithBit`
: If this bit is set, a catalog supports finding record names and record types beginning with a certain string. When you call the `DirEnumerateGet` function, such a catalog can accept an enumeration specification with the `matchNameHow` and `matchTypeHow` fields set to `kBeginsWith`; in this case, a search matches any record name or record type that begins with the string pointed to by the `recordName` or `typesList` field, respectively.

`kSupportsExactMatchBit`
: If this bit is set, a catalog supports finding a record based on an exact match with a record name or record type. When you call the `DirEnumerateGet` function, such a catalog can accept an enumeration specification with the `matchNameHow` and `matchTypeHow` fields set to `kMatchExact`; in this case, a search matches only the record name or record type pointed to by the `recordName` or `typesList` field, respectively.

`kSupportsEndsWithBit`
: If this bit is set, a catalog supports finding record names and record types ending with a certain string. When you call the `DirEnumerateGet` function, such a catalog can accept an enumeration specification with the `matchNameHow` and `matchTypeHow` fields set to `kEndingWith`; in this case, a search matches any record name or record type that ends with the string pointed to by the `recordName` or `typesList` field, respectively.

`kSupportsContainsBit`
: If this bit is set, a catalog supports finding record names and record types that contain a certain string. When you call the `DirEnumerateGet` function, such a catalog can accept an enumeration specification with the `matchNameHow` and `matchTypeHow` fields set to `kContaining`; in this case, a search matches any record name or record type that contains the string pointed to by the `recordName` or `typesList` field, respectively.

`kSupportsOrderedEnumerationBit`
: If this bit is set, a catalog returns requested information in a sorted order when you call the `DirEnumerateGet` function. It may return the information sorted by name or by type, in which case one of the two following bits will also be set. The catalog may also return the information in an unspecified sorted order.

`kCanSupportNameOrderBit`
: If this bit is set, a catalog supports the sorting by name option in the `DirEnumerateGet` function.

`kCanSupportTypeOrderBit`
: If this bit is set, a catalog supports the sorting by type option in the `DirEnumerateGet` function.

`kSupportsSortBackwardsBit`
: If this bit is set, a catalog supports the backward sort direction option in the `DirEnumerateGet` function.

`kSupportIndexRatioBit`
: If this bit is set, a catalog supports the index ratio feature in the `DirEnumerateGet` function. That is, the catalog can return the approximate position of a record among all records that match the search criteria in a dNode.

`kSupportsEnumerationContinueBit`
: If this bit is set, a catalog supports the continue feature in the `DirEnumerateGet` function.

`kSupportsLookupContinueBit`
: If this bit is set, a catalog supports the continue feature in the `DirLookupGet` function.

`kSupportsEnumerateAttributeTypeContinueBit`
: If this bit is set, a catalog supports the continue feature in the `DirEnumerateAttributeTypesGet` function.

`kSupportsEnumeratePseudonymContinueBit`
: If this bit is set, a catalog supports the continue feature in the `DirEnumeratePseudonymGet` function.

`kSupportsAliasesBit`
: If this bit is set, a catalog supports the `DirAddAlias` function. It also supports deleting an alias with the `DirDeleteRecord` function and enumerating aliases with the `DirEnumerateGet` function.

kSupportsPseudonymsBit
: If this bit is set, a catalog supports the `DirAddPseudonym`, `DirDeletePseudonym`, and `DirEnumeratePseudonymGet` functions. It also supports enumerating pseudonyms with the `DirEnumerateGet` function.

kSupportsPartialPathnamesBit
: If this bit is set, you 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
: If this bit is set, a catalog supports all Authentication Manager functions except those that relate to proxies. Support for proxies is specified by a separate bit.

kSupportsProxiesBit
: If this bit is set, a catalog supports the Authentication Manager functions that relate to proxies.

kSupportsFindRecordBit
: If this bit is set, a catalog supports the `DirFindRecordGet` and `DirFindRecordParse` functions.

You can use the following mask values to set the bits in a variable that specifies the features supported by a given catalog. Such variables are 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
};
```

Data Types

This section describes the data types that are specific to the Catalog Manager. See the chapter "AOCE Utilities" for descriptions of other data types that you use to provide information to or obtain information from Catalog Manager functions.

The Parameter Block Header

Every Catalog Manager routine takes a pointer to a `DirParamBlock` parameter block as input. The `DirParamBlock` parameter block defines a union of substructures, each of which is a parameter block for one of the Catalog Manager routines. Each routine description in "Catalog Manager Routines" starting on page 8-38 lists the fields of that routine's parameter block. Each parameter block contains the following header.

```
#define  AuthDirParamHeader \
   Ptr            qLink;         /* reserved */\
   long           reserved_H1;   /* reserved */\
   long           reserved_H2;   /* reserved */\
   ProcPtr        ioCompletion;  /* your completion routine */\
   OSErr          ioResult;      /* result code */\
   unsigned long  saveA5;        /* reserved */\
   short          reqCode;       /* CSAM request code*/\
   long           reserved[2];   /* reserved */\
   AddrBlock      serverHint;    /* PowerShare server's AppleTalk address */\
   short          dsRefNum;      /* personal catalog reference number */\
   unsigned long  callID;        /* reserved */\
   AuthIdentity   identity;      /* requester's authentication identity */\
   long           gReserved1;    /* reserved */\
   long           gReserved2;    /* reserved */\
   long           gReserved3;    /* reserved */\
   long           clientData;    /* you define this field */
```

Field descriptions

`qLink`	Reserved.
`reserved_H1`	Reserved.
`reserved_H2`	Reserved.

`ioCompletion`	A pointer to a completion routine that you can provide. When a Catalog Manager function that you called asynchronously completes execution, it calls your completion routine. Set this field to `nil` if you do not wish to 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 routine has been queued for execution. When the function completes execution, it sets this field to the actual result code.
`saveA5`	Reserved.
`reqCode`	This field is reserved when you call a Catalog Manager function. However, when the Catalog Manager passes a `DirParamBlock` parameter block to a CSAM, the `reqCode` field contains a constant that identifies which member of the `DirParamBlock` `union` type is being passed
`reserved[2]`	Reserved.
`serverHint`	The AppleTalk address of the PowerShare server to which you want to direct your request. Normally, you specify `nil` for all fields of this structure and the Catalog Manager directs the request to an appropriate PowerShare server. However, PowerShare server administration software (PowerShare Admin) may need to specify a particular server, and the `DirAddADAPDirectory` function requires a specific PowerShare server address. You can obtain the AppleTalk address of a PowerShare server from the `NBPLookup` function. The `AddrBlock` data structure is defined in *Inside Macintosh: Networking*.
`dsRefNum`	The reference number of the personal catalog to which the request applies. The `DirOpenPersonalDirectory` function returns this reference number when you open a personal catalog. If you are not addressing a personal catalog, set this field to 0.
`callID`	Reserved.
`identity`	The authentication identity of the requester. The authentication identity can be either the local identity of the owner of the computer or a specific identity. Typically, you set this field to the local identity to gain transparent access to all installed catalogs. You may also set this field to a specific identity. You can obtain the local identity from the Authentication Manager's `AuthGetLocalIdentity` function and a specific identity from the `AuthBindIdentity` function. See the chapter "Authentication Manager" in this book for more information about obtaining identities. Specify 0 for this field for guest access; that is, no identity. Functions that fail due to an insufficient level of access privilege return either the `kOCEReadAccessDenied` or `kOCEWriteAccessDenied` result code.
`gReserved1`	Reserved.
`gReserved2`	Reserved.
`gReserved3`	Reserved.

`clientData` Reserved for your use. The Catalog Manager passes the value in this field to your callback routines. If you have the same callback or completion routine processing more than one asynchronous request, your routine can use the `clientData` field to determine for which request it is processing results.

The dNode ID

A dNode ID consists of a dNode number that uniquely identifies a dNode within a catalog plus the name of the dNode. A dNode ID is defined by the `DNodeID` data structure. In the Catalog Manager API, it is not used as a stand-alone data structure; it is a member of the union part of the `DirEnumSpec` data structure, described on page 8-35.

```
struct DNodeID {
   DNodeNum    dNodeNumber;   /* dNode number  */
   long        reserved1;     /* reserved */
   RStringPtr  name;          /* name of the dNode */
   long        reserved2;     /* reserved */
};
```

The Enumeration Choice Type

The bits in a variable of type `DirEnumChoices` indicate types of entities. You use a variable of type `DirEnumChoices` to specify the type of entities about which you want information when you call the `DirEnumerateGet` function.

```
typedef unsigned long DirEnumChoices;
```

The bits in the `DirEnumChoices` data type are defined as follows:

```
enum {
   kEnumDistinguishedNameBit,
   kEnumAliasBit,
   kEnumPseudonymBit,
   kEnumDNodeBit,
   kEnumInvisibleBit
};
```

You can use the following values to set and test the bits in a variable of type `DirEnumChoices`.

```
enum {         /* values of DirEnumChoices */
   kEnumDistinguishedNameMask = 1L<<kEnumDistinguishedNameBit,
   kEnumAliasMask             = 1L<<kEnumAliasBit,
   kEnumPseudonymMask         = 1L<<kEnumPseudonymBit,
```

```
    kEnumDNodeMask              = 1L<<kEnumDNodeBit,
    kEnumInvisibleMask          = 1L<<kEnumInvisibleBit
};

#define kEnumAllMask (kEnumDistinguishedNameMask | kEnumAliasMask |
                      kEnumPseudonymMask | kEnumDNodeMask |
                      kEnumInvisibleMask)
```

Descriptions

`kEnumDistinguishedNameMask`
This setting specifies a record.

`kEnumAliasMask`
This setting specifies an alias.

`kEnumPseudonymMask`
This setting specifies a pseudonym.

`kEnumDNodeMask`
This setting specifies a dNode.

`kEnumInvisibleMask`
As an input, this setting specifies all dNodes, records, aliases, and pseudonyms, both visible and invisible. As an output, it is set in conjunction with either `kEnumDistinguishedNameMask`, `kEnumAliasMask`, `kEnumPseudonymMask`, or `kEnumDNodeMask`, and indicates that the specified entity is invisible.

`kEnumAllMask` As an input, this setting specifies all visible records, aliases, pseudonyms, and dNodes. It is not used as an output.

The Enumeration Specification

The `DirEnumSpec` data structure contains information about either a record, an alias, a pseudonym, or a dNode. The value of the `enumFlag` field indicates the type of entity to which the rest of the information applies as well as the format of that information.

When you want to enumerate the contents of a dNode starting from a specific dNode, record, alias, or pseudonym, you provide a `DirEnumSpec` structure to the `DirEnumerateGet` function that specifies the record, alias, pseudonym, or dNode at which you want the `DirEnumerateGet` function to start the enumeration. The `DirEnumerateParse` function passes a `DirEnumSpec` structure to your callback routine for each record, alias, pseudonym, or dNode that it finds in the buffer.

```
struct DirEnumSpec {
   DirEnumChoices enumFlag;      /* type of entity */
   unsigned short indexRatio;    /* approximate record position */
   union {
      LocalRecordID  recordIdentifier; /* record information */
```

```
      DNodeID        dnodeIdentifier;  /* dNode info */
   }u;
};
```

Field descriptions

`enumFlag` A value that indicates the type of entity about which information is provided in the `u` field. The following constants indicate whether the information applies to a record, an alias, a pseudonym, or a dNode: `kEnumDistinguishedNameMask`, `kEnumAliasMask`, `kEnumPseudonymMask`, or `kEnumDNodeMask`. The `kEnumInvisibleMask` constant indicates whether the entity is invisible or visible.

`indexRatio` The approximate position, expressed as a percentile ranging from 1 to 100, of a record among all records in a dNode. This is a hint that can be used with a scroll box (or some other mechanism) to show how far you have moved through a list of records. If a catalog does not support this feature, it sets this field to 0.

`u.recordIdentifier`
If the `enumFlag` field is set to `kEnumDistinguishedNameMask`, `kEnumAliasMask` or `kEnumPseudonymMask`, this field contains a `LocalRecordID` data structure. The local record ID specifies a record's name, type, and creation ID.

`u.dnodeIdentifier`
If the `enumFlag` field is set to `kEnumDNodeMask`, this field contains a `DNodeID` data structure. A dNode ID specifies a dNode's name and its dNode number. If a catalog does not support dNode numbers, the `dNodeNumber` field is set to 0.

The Script Structure

The script structure `SLRV`, returned by the `DirEnumerateGet` function, identifies the script, language, and region that the function uses to sort the entries in your buffer.

```
struct SLRV {
   ScriptCode  script;     /*  script code in which entries are sorted */
   short       language;   /*  language code in which entries are sorted */
   short       regionCode; /*  region code in which entries are sorted */
   short       version;    /*  version of AOCE sorting software */
};
```

Field descriptions

`script`	The script code identifies the script that the `DirEnumerateGet` function uses in sorting.
`language`	The language code identifies the language that the `DirEnumerateGet` function uses in sorting.
`regionCode`	The region code identifies the region that the `DirEnumerateGet` function uses in sorting.
`version`	The constant `kCurrentOCESortVersion`. It identifies the version of AOCE sorting software that the `DirEnumerateGet` function uses.

The Matching Criteria Type

You use the `DirMatchWith` data type to indicate a matching mode when you enumerate the contents of a dNode. You always use a variable of type `DirMatchWith` in conjunction with a search string. The `DirMatchWith` variable specifies the criteria that the `DirEnumerateGet` function uses to determine when it has found a match with your search string.

```
typedef unsigned char DirMatchWith;
```

The possible values of the `DirMatchWith` data type are defined as follows:

```
   enum {            /* values of DirMatchWith */
      kMatchAll,
      kExactMatch,
      kBeginsWith,
      kEndingWith,
      kContaining
};
```

Descriptions

`kMatchAll`	Match any string.
`kExactMatch`	Match only those strings that are exactly the same as the search string.
`kBeginsWith`	Match any string that begins with the search string.
`kEndingWith`	Match any string that ends with the search string.
`kContaining`	Match any string that contains the search string.

Catalog Manager Functions

This section describes the Catalog Manager functions that provide services such as getting information about catalogs and dNodes, managing the PowerTalk Setup catalog, managing records and attribute values and types, and controlling access to dNodes, records, and attribute types.

All of the Catalog Manager functions take a pointer to a catalog parameter block as input. Each routine description includes a list of the fields in the parameter block that are used by the function. Each list of parameter block fields has four columns. See the Preface to this book for a description of the type of information that each column contains.

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.

Getting Information About Catalogs

You can use the functions in this section to get a variety of information about the catalogs that are listed in the PowerTalk Setup catalog. The `DirEnumerateDirectoriesGet` and `DirEnumerateDirectoriesParse` functions work together to provide the catalog name, discriminator value, and feature flags for some or all of the catalogs listed in the PowerTalk Setup catalog. You can discover the features that a specific catalog supports by calling the `DirGetDirectoryInfo` function. The `DirGetLocalNetworkSpec` function provides you with the name of the network on which a PowerShare catalog is located. You can get information about the icons that represent a catalog by calling the `DirGetDirectoryIcon` function. The `DirGetExtendedDirectoriesInfo` function provides additional information about an external catalog that is specific to that catalog.

DirEnumerateDirectoriesGet

The `DirEnumerateDirectoriesGet` function returns information about catalogs that are listed in the PowerTalk Setup catalog.

```
pascal OSErr DirEnumerateDirectoriesGet
                                    (DirParamBlockPtr paramBlock,
                                       Boolean async);
```

paramBlock
Pointer to a parameter block.

async A Boolean value that specifies whether the function is to be executed asynchronously. Set this parameter 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
→	directoryKind	OCEDirectoryKind	Catalog type
→	startingDirectoryName	DirectoryNamePtr	Starting catalog name
→	startingDirDiscriminator	DirDiscriminator	Starting discriminator value
→	includeStartingPoint	Boolean	Begin enumeration with starting point?
↔	getBuffer	Ptr	Your buffer
→	getBufferSize	unsigned long	Size of your buffer

See “The Parameter Block Header” on page 8-32 for descriptions of the ioCompletion, ioResult, and clientData fields.

Field descriptions

directoryKind A value that indicates the type of catalog about which you are requesting information. Use the constant kDirADAPKind to request information about PowerShare catalogs. Use the constant kDirDSAMKind to request information about external catalogs. To request information about both PowerShare and external catalogs, use the constant kDirAllKinds. You can also supply a specific signature value to get information on catalogs having that signature. The function does not return information about personal catalogs.

startingDirectoryName
A pointer to the name of the catalog at which you want the DirEnumerateDirectoriesGet function to begin the enumeration. Set this field to nil to start with the first catalog. If the DirEnumerateDirectoriesGet function completes with the kOCEMoreData result code, set this field to the value of the last dirName parameter passed to your callback routine by the DirEnumerateDirectoriesParse function to continue the enumeration. You must coordinate the value you provide in this field with the value you provide in the startingDirDiscriminator field; that is, both values are required, and both must apply to the same catalog.

startingDirDiscriminator
The discriminator value of the catalog at which you want the DirEnumerateDirectoriesGet function to begin the enumeration. Set the fields of this structure to 0 to start with the first catalog. If the DirEnumerateDirectoriesGet function completes with the kOCEMoreData result code, set this field to the value of the last discriminator parameter passed to your

callback routine by the `DirEnumerateDirectoriesParse` function to continue the enumeration. You must coordinate the value you provide in this field with the value you provide in the `startingDirectoryName` field; that is, both values are required, and both must apply to the same catalog

`includeStartingPoint`
A Boolean value that tells the `DirEnumerateDirectoriesGet` function how to interpret the `startingDirectoryName` and `startingDirDiscriminator` fields. Set this field to `true` if you want the `DirEnumerateDirectoriesGet` function to return information about catalogs beginning with the one specified by the `startingDirectoryName` and `startingDirDiscriminator` fields. If you set this field to `false`, the function returns information starting with the catalog immediately *after* the one specified by the `startingDirectoryName` and `startingDirDiscriminator` fields.

`getBuffer` A pointer to the buffer in which the function stores the name, the discriminator value, and the capability flags for each catalog listed in the PowerTalk Setup catalog. You provide this buffer.

`getBufferSize` The number of bytes in the buffer.

DESCRIPTION

You call the `DirEnumerateDirectoriesGet` function to obtain information about PowerShare catalogs and external catalogs that are listed in the PowerTalk Setup catalog. You can request information about either PowerShare catalogs or external catalogs, or about both. You can also request information about catalogs that share a specific signature that you specify. For example, if there are several X.500 catalogs listed in the PowerTalk Setup catalog and they used the same signature value, you could request information about that set of catalogs.

For each catalog about which you have requested information, the function places the catalog's name, its discriminator value, and its feature flags in the buffer you provide. If your buffer is not large enough to contain all of the information you requested, the function places as many sets of catalog name, discriminator value, and feature flags as will fit in your buffer and returns the `kOCEMoreData` result code.

When the function completes with either the `noErr` or `kOCEMoreData` result codes, you can provide a pointer to your buffer to the `DirEnumerateDirectoriesParse` function, which extracts the catalog information from the buffer and passes it to a callback routine that you provide.

If your buffer is too small to hold all of the information you requested, you can continue to obtain information by calling the `DirEnumerateDirectoriesGet` function again, after calling the `DirEnumerateDirectoriesParse` function. For the values of the `startingDirectoryName` and `startingDirDiscriminator` fields, use the values that the `DirEnumerateDirectoriesParse` function last passed to the `dirName` and

`discriminator` parameters of your callback routine. The `DirEnumerateDirectoriesGet` function will continue the enumeration starting with the next catalog as determined by the value of the `includeStartingPoint` field.

Because personal catalogs are not listed in the PowerTalk Setup catalog, the `DirEnumerateDirectoriesGet` function does not return information about them. To obtain the name, discriminator value, and feature flags of a personal catalog, locate the catalog using the routines in the Standard File Package; open the catalog by calling the `DirOpenPersonalDirectory` function, and call the `DirGetDirectoryInfo` function to get the information.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$011A

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `DirEnumerateDirectoriesParse` function is described next.

The `DirGetDirectoryInfo` function is described on page 8-48.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesGet` function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" on page 8-20.

DirEnumerateDirectoriesParse

The `DirEnumerateDirectoriesParse` function parses the data returned by the `DirEnumerateDirectoriesGet` function and returns information about catalogs, one catalog at a time, by repeatedly calling your callback routine.

```
pascal OSErr DirEnumerateDirectoriesParse
                                    (DirParamBlockPtr paramBlock,
                                    Boolean async);
```

`paramBlock`
Pointer to a parameter block.

async 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
→	clientData	long	You define this field
→	eachDirectory	ForEachDirectory	Your callback routine
→	getBuffer	Ptr	Your buffer
→	getBufferSize	unsigned long	Size of your buffer

See "The Parameter Block Header" on page 8-32 for descriptions of the ioCompletion, ioResult, and clientData fields.

Field descriptions

eachDirectory A pointer to your callback routine. The function declaration for this routine is described on page 8-153.

getBuffer A pointer to the buffer containing the information to parse. Use the same buffer that you provided to the DirEnumerateDirectoriesGet function.

getBufferSize The number of bytes in the buffer. Use the same value that you provided to the DirEnumerateDirectoriesGet function.

DESCRIPTION

You call the DirEnumerateDirectoriesParse function to extract the catalog information placed in your buffer by the DirEnumerateDirectoriesGet function. You must provide a callback routine that the DirEnumerateDirectoriesParse function calls for each set of catalog information that it finds in the buffer. Each time it calls your callback routine, the function passes it the name, discriminator value, and the feature flags of a catalog.

The DirEnumerateDirectoriesParse function completes when it has finished parsing the contents of your buffer or when your callback routine returns true. The function returns the kOCEMoreData result code if it reaches the end of the buffer and finds that the DirEnumerateDirectoriesGet function did not return all the data requested. To continue the enumeration, call the DirEnumerateDirectoriesGet function again. Get the values of the dirName and discriminator parameters that the DirEnumerateDirectoriesParse function last passed to your callback routine. In your next call to the DirEnumerateDirectoriesGet function, use these as the values of the startingDirectoryName and startingDirDiscriminator fields.

If your callback routine returns true, the DirEnumerateDirectoriesParse function completes with the noErr result code.

Because the `DirEnumerateDirectoriesGet` function returns information about PowerShare and external catalogs only, the `DirEnumerateDirectoriesParse` function can retrieve information only about these types of catalogs. To obtain the name, discriminator value, and feature flags of a personal catalog, locate the catalog using the routines in the Standard File Package; open the catalog by calling the `DirOpenPersonalDirectory` function, and call the `DirGetDirectoryInfo` function to get the information.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0106

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEMoreData`	–1623	More data available

SEE ALSO

The function declaration for your callback routine is described on page 8-153.

The `DirGetDirectoryInfo` function is described on page 8-48.

The `DirEnumerateDirectoriesGet` function is described on page 8-38.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesParse` function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" on page 8-20.

DirFindRecordGet

The `DirFindRecordGet` function returns information about the records, aliases, and pseudonyms contained in a catalog that you specify.

```
pascal OSErr DirFindRecordGet (DirParamBlockPtr paramBlock,
                               Boolean async);
```

`paramBlock`
Pointer to a parameter block.

`async` 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
→	serverHint	AddrBlock	AppleTalk address of the PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
→	startingPoint	DirEnumSpec*	Starting point for enumeration
→	nameMatchString	RStringPtr	Name of record, alias, or pseudonym you want returned
→	typesList	RStringPtr*	List of types you want returned
→	typeCount	unsigned long	Number of types in the list
→	matchNameHow	DirMatchWith	Match criteria for names
→	matchTypeHow	DirMatchWith	Match criteria for types
↔	getBuffer	Ptr	Your buffer
→	getBufferSize	unsigned long	Size of your buffer
→	directoryName	DirectoryNamePtr	Catalog name
→	discriminator	Discriminator	Discriminator value

See "The Parameter Block Header" on page 8-32 for descriptions of the ioCompletion, ioResult, serverHint, dsRefNum, identity, and clientData fields.

Field descriptions

`startingPoint` A pointer to the record, alias, or pseudonym at which you want the function to start the enumeration. Set this field to `nil` when you call the `DirFindRecordGet` function for the first time. If the function completes with the `kOCEMoreData` result code, you can set this field to the value of the last `enumSpec` parameter passed to your callback routine by the `DirFindRecordParse` function to continue the enumeration from the next record, alias, or pseudonym.

`nameMatchString` A pointer to the name of the record, alias, or pseudonym about which you want information. You specify the mode in which you want the function to match the name in the `matchNameHow` field. If you specify `kMatchAll` in the `matchNameHow` field, the function ignores this field. The `DirFindRecordGet` function returns only records, aliases, or pseudonyms whose names match the value that you specify according to the match criteria that you specify.

`typesList` A pointer to an array of pointers. Each element in the array points to a record type about which you want information. Your array may include both AOCE-defined record types and record types that you define. You specify the mode in which you want the function to match the type in the `matchTypeHow` field. If you specify `kMatchAll` in the `matchTypeHow` field, the function ignores this field.

`typeCount` The number of pointers to record types in your array of types.

matchNameHow	A value that specifies the matching mode by which the function determines matches with the name you provide in the nameMatchString field. The possible values for exact and wildcard matching are described on page 8-37.
matchTypeHow	A value that specifies the matching mode by which the function determines matches with the values you provide in the typesList field. The possible values for exact and wildcard matching are described on page 8-37. If you specify kMatchAll, the function returns information on each record, alias, or pseudonym whose name matches the value pointed to by the nameMatchString field.
getBuffer	A pointer to the buffer in which the function stores the requested information. You provide this buffer.
getBufferSize	The number of bytes in the buffer.
directoryName	A pointer to the name of the catalog whose records you want to enumerate. You provide the name buffer.
discriminator	A unique value associated with a catalog that distinguishes it from other catalogs with the same name.

DESCRIPTION

You call the DirFindRecordGet function to obtain a list of records, aliases, pseudonyms, or all of these for a catalog that you specify. This function allows you to specify matching criteria for both names and types.

The sort order of the information returned by the function is undefined.

The DirFindRecordGet function places a local record ID for each record, alias, or pseudonym that it finds in your buffer. The function provides only whole units of information for each entity. That is, it will not provide the creation ID for a record without also providing its name and type. If your buffer is not large enough to contain all of the information requested, the DirFindRecordGet function provides complete information on as many records, aliases, or pseudonyms as will fit and returns the kOCEMoreData result code.

When the function completes with either the noErr or kOCEMoreData result codes, you use a pointer to your buffer as input to the DirFindRecordParse function, which extracts the information from the buffer.

If the DirFindRecordGet function returned the kOCEMoreData result code, you can request additional information by calling it again after calling the DirFindRecordParse function. Get the value of the enumSpec parameter that the DirFindRecordParse function last passed to your callback routine. When you call DirFindRecordParse again, use this value in the startingPoint field. Use the same values for the nameMatchString and typesList fields that you used in your original call to the DirFindRecordGet function. The DirFindRecordGet function will continue the enumeration starting with the next record, alias, or pseudonym.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0140

RESULT CODES

noErr	0	No error
kOCEMoreData	–1623	More data available

SEE ALSO

The DirFindRecordParse function is described next.

For an example of continuing the enumeration (using the DirEnumerateAttributeTypesGet function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" on page 8-20.

To obtain the value for the discriminator field, call the DirGetDirectoryInfo function on page 8-48.

DirFindRecordParse

The DirFindRecordParse function parses the data returned by the DirFindRecordGet function and returns information on each record, alias, or pseudonym by repeatedly calling your callback routine.

```
pascal OSErr DirFindRecordParse (DirParamBlockPtr paramBlock,
                                 Boolean async);
```

paramBlock
Pointer to a parameter block.

async — 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
→	serverHint	AddrBlock	AppleTalk address of the PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
→	startingPoint	DirEnumSpec *	Starting point for enumeration
→	nameMatchString	RStringPtr	Name of record, alias, or pseudonym you want returned
→	typesList	RStringPtr *	List of types you want returned

→	typeCount	unsigned long	Number of types in the list
→	matchNameHow	DirMatchWith	Match criteria for names
→	matchTypeHow	DirMatchWith	Match criteria for types
→	getBuffer	Ptr	Your buffer
→	getBufferSize	unsigned long	Size of your buffer
→	directoryName	DirectoryNamePtr	Catalog name
→	discriminator	Discriminator	Discriminator value
→	forEachRecordFunc	ForEachRecord	Your callback routine

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`startingPoint` Use the value you provided in the `startingPoint` field of the `DirFindRecordGet` function.

`nameMatchString`
Use the value you provided in the `nameMatchString` field of the `DirFindRecordGet` function.

`typesList` Use the value you provided in the `typesList` field of the `DirFindRecordGet` function.

`typeCount` Use the value you provided in the `typeCount` field of the `DirFindRecordGet` function.

`matchNameHow` Use the value you provided in the `matchNameHow` field of the `DirFindRecordGet` function.

`matchTypeHow` Use the value you provided in the `matchTypeHow` field of the `DirFindRecordGet` function.

`getBuffer` Use the value you provided in the `getBuffer` field of the `DirFindRecordGet` function.

`getBufferSize` Use the value you provided in the `getBufferSize` field of the `DirFindRecordGet` function.

`directoryName` Use the value you provided in the `directoryName` field of the `DirFindRecordGet` function.

`discriminator` Use the value you provided in the `discriminator` field of the `DirFindRecordGet` function.

`forEachRecordFunc`
A pointer to your callback routine.

DESCRIPTION

You call the `DirFindRecordParse` function to extract the information that the `DirFindRecordGet` function placed in your buffer. You must provide a callback routine that the `DirFindRecordParse` function calls for each record, alias, or pseudonym about which there is information in the buffer. The `DirFindRecordParse` function provides a local record ID for each record, alias, or pseudonym. See the description of your callback routine on page 8-159 for more information.

The `DirFindRecordParse` function completes when it has finished parsing the contents of your buffer or when your callback routine returns `true`. The function returns

the kOCEMoreData result code if it reaches the end of the buffer and finds that the DirFindRecordGet function did not return all the data requested. To continue the enumeration, call the DirFindRecordGet function again. In your next call to the DirFindRecordGet function, for the value of the startingPoint field, use the value that your callback routine last received in the enumSpec parameter.

If your callback routine returns true, the DirFindRecordParse function completes with the noErr result code.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0141

RESULT CODES

noErr	0	No error
kOCEParamErr	–50	Invalid parameter
kOCEMoreData	–1623	More data available

SEE ALSO

The function declaration for your callback routine is described on page 8-159.

The DirFindRecordGet function is described on page 8-43.

For an example of continuing the enumeration (using the DirEnumerateAttributeTypesParse function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" on page 8-20.

DirGetDirectoryInfo

The DirGetDirectoryInfo function returns information about a catalog that you specify.

```
pascal OSErr DirGetDirectoryInfo (DirParamBlockPtr paramBlock,
                                  Boolean async);
```

paramBlock
: Pointer to a parameter block.

async
: 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
→	serverHint	AddrBlock	AppleTalk address of the PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
↔	directoryName	DirectoryNamePtr	The name of the catalog
↔	discriminator	DirDiscriminator	Discriminator value
←	features	DirGestalt	Feature flags

See "The Parameter Block Header" on page 8-32 for descriptions of the ioCompletion, ioResult, serverHint, dsRefNum, identity, and clientData fields.

Field descriptions

directoryName — A pointer to the name of the catalog about which you want information. You provide the name buffer. You specify the catalog name unless you are requesting information about a personal catalog. In that case, you may provide either the personal catalog's name and discriminator value or its reference number. If you specify its reference number in the dsRefNum field, the function returns, in the buffer supplied for the directoryName field, the volume name on which the personal catalog resides. To obtain the file specification for the personal catalog, call the DirMakePersonalDirectoryRLI function first. Then call OCEExtractAlias using the record location information you obtained to extract the File Manager alias for the personal catalog.

discriminator — A unique value that distinguishes a catalog from other catalogs with the same name. You specify this field unless you are requesting information about a personal catalog. In that case, you may provide either the personal catalog's name and discriminator value or its reference number. If you specify its reference number in the dsRefNum field, the function returns the discriminator value in this field.

features — A set of bit flags that describe the features that a catalog supports. The function returns these flags for the catalog that you specify.

DESCRIPTION

You call the DirGetDirectoryInfo function to determine the features that a catalog supports before calling other Catalog Manager functions that address that catalog.

In addition to returning a catalog's feature flags, the DirGetDirectoryInfo function may also return the name and discriminator value for a catalog. The function first examines the dsRefNum field. If you specify a nonzero value for the dsRefNum field (that is, if your target catalog is a personal catalog), the DirGetDirectoryInfo function returns the name, the discriminator value, and the feature flags for the personal

catalog that you identified. If the `dsRefNum` field is set to 0, the function examines the `serverHint` field. A special case arises when you request information about a PowerShare catalog and you specify the AppleTalk address of a server for that catalog in the `serverHint` field. In this case, you do not need to provide the catalog name and discriminator. The function returns those values as well the feature flags.

To test the bits in the `features` field, you can use the mask values shown on page 8-31.

Note

The `DirEnumerateDirectoriesGet` function also returns the name, discriminator value, and feature flags for PowerShare and external catalogs. Unlike the `DirGetDirectoryInfo` function, which requires that you know some information about a specific catalog before you can request additional information about that catalog, the `DirEnumerateDirectoriesGet` function returns catalog information without you needing to provide any. However, the `DirEnumerateDirectoriesGet` function returns information only about the PowerShare and external catalogs listed in the PowerTalk Setup catalog. u

SPECIAL CONSIDERATIONS

The `DirFindADAPDirectoryByNetSearch` and `DirAddADAPDirectory` functions allow you to make a catalog available for your use without adding it to the PowerTalk Setup catalog. You can call the `DirGetDirectoryInfo` function for any catalog that you have made privately available.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0119

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available

SEE ALSO

The `DirEnumerateDirectoriesGet` function is described on page 8-38.

You obtain a reference number for a personal catalog from the `DirOpenPersonalDirectory` function, which is described on page 8-84.

DirGetLocalNetworkSpec

The `DirGetLocalNetworkSpec` function returns the name of the network on which a PowerShare catalog resides.

```
pascal OSErr DirGetLocalNetworkSpec (DirParamBlockPtr paramBlock,
                                     Boolean async);
```

`paramBlock`
Pointer to a parameter block.

`async` 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
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`directoryName`	`DirectoryNamePtr`	Catalog name
→	`discriminator`	`DirDiscriminator`	Discriminator value
↔	`networkSpec`	`NetworkSpecPtr`	Network name

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`directoryName` A pointer to the name of the PowerShare catalog to which the request applies.

`discriminator` The discriminator value of the PowerShare catalog to which the request applies. A catalog discriminator differentiates between two or more catalogs with the same name.

`networkSpec` A pointer to a buffer in which the function places the name of the network in which the catalog resides. You provide this buffer. The buffer should be big enough to hold a maximum size `NetworkSpec` data structure.

DESCRIPTION

You call the `DirGetLocalNetworkSpec` function when you want to know the name of the network on which a specific ADAP catalog resides. You provide the catalog name and discriminator value. The function returns in the `networkSpec` field a pointer to the name of the network. The information that this function provides may be useful in an environment containing multiple interconnected networks.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0124

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available

SEE ALSO

The `NetworkSpec` data structure is described in the chapter "AOCE Utilities" in this book.

DirGetDirectoryIcon

The `DirGetDirectoryIcon` function returns information about an icon representing a catalog that you specify.

```
pascal OSErr DirGetDirectoryIcon (DirParamBlockPtr paramBlock,
                                  Boolean async);
```

`paramBlock`
Pointer to a parameter block.

`async` 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
→	serverHint	AddrBlock	AppleTalk address of the PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
→	pRLI	PackedRLIPtr	Target catalog
→	iconType	OSType	The type of icon
↔	iconBuffer	Ptr	Your buffer
↔	bufferSize	unsigned long	Size of buffer on input; data bytes in buffer on output

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`pRLI` A pointer to packed record location information for the catalog whose icon you want to obtain. The function ignores this field when you provide a nonzero value in the `dsRefNum` field to specify a personal catalog.

`iconType` The type of icon about which you want information. Specify one of the following: '`ICN#`', '`icl8`', '`icl4`', '`ics8`' '`ics4`', or '`ics#`'.

`iconBuffer` A pointer to the buffer in which the function stores the icon data. You provide this buffer.

`bufferSize` On input, you set this field to the size of the buffer pointed to by the `iconBuffer` field. On output, the function sets this field to the size of the icon it placed in your buffer. If the function completes with the `kOCEBufferTooSmall` result code, it sets this field to the size of the icon.

DESCRIPTION

You call the `DirGetDirectoryIcon` function to get icon information for a catalog so that you may display the icon.

This function is not supported by PowerShare and personal catalogs. A catalog service access module may support this function for its catalog.

If your buffer is not large enough to hold the icon you requested, the function returns the `kOCEBufferTooSmall` result code. In that case, the `bufferSize` field contains the size of the icon. You should increase the size of your buffer to the icon size and call the function again.

SPECIAL CONSIDERATIONS

Apple Computer, Inc., does not publish the size of icon resources. They are subject to change.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0121

RESULT CODES

noErr	0	No error
kOCEBufferTooSmall	–1503	Buffer too small for data requested

SEE ALSO

You can create a PackedRLI data structure with the OCEPackRLI utility routine. It is described in the chapter "AOCE Utilities" in this book.

For information about the different icon types and the format of the data associated with those types, see the chapter "Finder Interface" in *Inside Macintosh: Macintosh Toolbox Essentials*.

DirGetExtendedDirectoriesInfo

The DirGetExtendedDirectoriesInfo function returns extended information about catalogs.

```
pascal OSErr DirGetExtendedDirectoriesInfo
                                    (DirParamBlockPtr paramBlock,
                                    Boolean async);
```

paramBlock
Pointer to a parameter block.

async 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
→	serverHint	AddrBlock	AppleTalk address of the PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
↔	buffer	Ptr	Your output buffer
→	bufferSize	unsigned long	Size of buffer;
←	totalEntries	unsigned long	Number of catalogs found
←	actualEntries	unsigned long	Number of entries returned

See “The Parameter Block Header” on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`buffer` A pointer to your buffer in which the function stores the information you request.

`bufferSize` The number of bytes in your buffer. You set this field to the size of your buffer in bytes.

`totalEntries` The total number of external catalogs that the `DirGetExtendedDirectoriesInfo` function found listed in the PowerTalk Setup catalog.

`actualEntries` The number of catalogs about which the function has returned information in your buffer.

DESCRIPTION

You call the `DirGetExtendedDirectoriesInfo` function to get information about catalogs. The function provides more information than is available from the `DirEnumerateDirectoriesGet` function. For example, it might return information on the addressing scheme used by an external catalog. Typically, an AOCE address template calls this function to help construct an address for a messaging service access module. Unlike the `DirEnumerateDirectoriesGet` function, `DirGetExtendedDirectoriesInfo` has no associated parse routine. Thus, you must parse the contents of the buffer yourself.

For each catalog, the `DirGetExtendedDirectoriesInfo` function stores information in your buffer in the following format:

```
struct EachDirectoryData {
   PackedRLI      pRLI;          /* packed RLI for catalog */

   OSType         entnType;      /* address type */
   long           hasMailSlot;   /* catalog has mail slot? */
   ProtoRString   realName;      /* real name */
   ProtoRString   comment;       /* comment for display */
   long           length;        /* data length */
   char           data[length];  /* data */
};
```

Field descriptions

`pRLI` Packed record location information that identifies the catalog.

`entnType` The address type.

`hasMailSlot` The `DirGetExtendedDirectoriesInfo` function sets this field to 1 if the catalog is associated with a mail slot. Otherwise, it sets this field to 0.

`realName` The name of the catalog in its native environment. This may differ from its catalog name within an AOCE system. It is word aligned.

`comment`	Information that the catalog provider stores in its record in the PowerTalk Setup catalog for display to a user. Typically, this information further identifies and describes the catalog to the user. For example, it might say "This catalog is located in Paris, France. You are connected to it via a public packet-switched network." It is word aligned.
`length`	The number of bytes in the `data` field.
`data`	Information about the catalog, padded to an even boundary.

Your buffer must be large enough to accommodate the total number of entries that the function finds. If your buffer is not large enough to hold all of the information, the function completes with the `kOCEMoreData` result code. In that case, use the value of the `totalEntries` field as a guide in allocating a bigger buffer and then call the function again. Because the function returns data that is of variable length for each catalog, this is a trial-and-error method.

Note that there is no way to have the `DirGetExtendedDirectoriesInfo` function return only data it has not previously returned. It always attempts to return information on every catalog that it finds.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0136

RESULT CODES

`noErr`	0	No error
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `DirEnumerateDirectoriesGet` function is described on page 8-38.

For information on messaging service access modules, see the chapter "Messaging Service Access Modules" in *Inside Macintosh: AOCE Service Access Modules*.

The chapter "Service Access Module Setup" in *Inside Macintosh: AOCE Service Access Modules* describes address templates.

For an example of using the `DirEnumerateDirectoriesGet` function, see "Getting Extended Catalog Information" beginning on page 8-24.

Getting Information About dNodes

You can use the functions in this section to get a variety of information about dNodes. The `DirEnumerateGet` and `DirEnumerateParse` functions work together to provide information about the contents of a dNode. You can detect changes in a dNode by calling the `DirGetDNodeMetaInfo` function which indicates whether a specific dNode is a leaf

node in a catalog tree. If you know the pathname information for a dNode, you can obtain its dNode number and vice versa by using the functions `DirMapDNodeNumberToPathName` and `DirMapPathNameToDNodeNumber`.

DirEnumerateGet

The `DirEnumerateGet` function returns information about the contents of a dNode that you specify. The contents of a dNode include records, aliases, pseudonyms, and dNodes.

```
pascal OSErr DirEnumerateGet (DirParamBlockPtr paramBlock,
                              Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRLI`	`PackedRLIPtr`	Target dNode
→	`startingPoint`	`DirEnumSpec*`	Starting point for enumeration
→	`sortBy`	`DirSortOption`	Return data in name or type order
→	`sortDirection`	`DirSortDirection`	Search forward or backward for info
→	`nameMatchString`	`RStringPtr`	Name of record, alias, pseudonym, or dNode you want returned
→	`typesList`	`RStringPtr*`	List of types you want returned
→	`typeCount`	`unsigned long`	Number of types in the list
→	`enumFlags`	`DirEnumChoices`	Types of entities about which you want information
→	`includeStartingPoint`	`Boolean`	Begin enumeration with starting point?
→	`matchNameHow`	`DirMatchWith`	Match criteria for names
→	`matchTypeHow`	`DirMatchWith`	Match criteria for types
↔	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer
←	`responseSLRV`	`SLRV`	Script information

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRLI` A pointer to packed record location information that identifies the dNode for which you want a list of records, aliases, pseudonyms, or dNodes. You use the `enumFlags` field to specify the type of entity about which you want information. The function ignores the `aRLI` field when you provide a nonzero value in the `dsRefNum` field to specify a personal catalog.

`startingPoint` A pointer to the record, alias, pseudonym, or dNode at which you want the function to start the enumeration. You specify the type of entity in the `enumFlag` field of the `DirEnumSpec` data structure and provide either a `LocalRecordID` or a `DNodeID` data structure to identify the specific entity from which you want the function to start returning information. Set this field to `nil` to start with the first record, alias, pseudonym, or dNode in the dNode. If the `DirEnumerateGet` function completes with the `kOCEMoreData` result code, you can continue the enumeration as follows: Set the `startingPoint` field to the value of the last `enumSpec` parameter passed to your callback routine by the `DirEnumerateParse` function.

`sortBy` A constant that specifies whether the function returns the records and dNodes sorted by name or sorted by type. Set this field to the constant `kSortByName` if you want the data ordered alphabetically by name. Set this field to the constant `kSortByType` if you want the data ordered alphabetically by type.

`sortDirection` A constant that specifies whether the function returns the information you requested in forward sort order or reverse sort order. Set this field to the constant `kSortForwards` if you want your data in forward sort order. Set it to the constant `kSortBackwards` if you want your data in reverse sort order.

`nameMatchString`
A pointer to the name of the record, alias, pseudonym, or dNode about which you want information. Use the `matchNameHow` field to specify the mode in which you want the function to match the name. If you specify `kMatchAll` in the `matchNameHow` field, the function ignores this field. The `DirEnumerateGet` function returns only records, aliases, pseudonyms, or dNodes whose names match the value that you specify according to the match criteria that you specify.

`typesList` A pointer to an array of pointers. Each element in the array points to a record type about which you want information. Your array may include both AOCE-defined record types and record types that you define. In the `matchTypeHow` field, specify the mode in which you want the function to match the type. If you specify `kMatchAll` in the `matchTypeHow` field, the function ignores this field.

`typeCount` The number of pointers to record types in your array of types.

`enumFlags` A mask value that specifies whether you want the `DirEnumerateGet` function to return information about records, aliases, pseudonyms, dNodes, or some combination of these. The mask constants that you can specify are described in the section

"The Enumeration Choice Type" on page 8-34. With the `enumFlag` field of the `DirEnumSpec` data structure and with either a `LocalRecordID` or a `DNodeID` data structure that you provide in that data structure, you identify the specific entity from which you want the function to start returning information.

`includeStartingPoint`
: A Boolean value that tells the function how to interpret the `startingPoint` field. Set `includeStartingPoint` to `true` if you want `DirEnumerateGet` to return information beginning with the entity specified by the `startingPoint` field. Set this field to `false` if you want the `DirEnumerateGet` function to return information beginning with the entity immediately after the entity specified by the `startingPoint` field.

`matchNameHow`
: A value that specifies the matching mode used to determine matches with the name specified by `nameMatchString`. The possible values for exact and wildcard matching are described on page 8-37.

`matchTypeHow`
: A value that specifies the matching mode used to determine matches with the values specified by `typesList`. The possible values for matching are described in "The Matching Criteria Type" on page 8-37. If you specify `kMatchAll`, the function returns information on each instance of a target entity whose name matches the value pointed to by the `nameMatchString` field. (You specify target entities in the `enumFlags` field.)

`getBuffer`
: A pointer to the buffer in which the function stores the requested information. You provide this buffer.

`getBufferSize`
: The number of bytes in the buffer.

`responseSLRV`
: A structure in which the function returns the script code, language code, and region code of the character set that the function used to sort the entries in your buffer.

DESCRIPTION

You call the `DirEnumerateGet` function to obtain a list of records, aliases, pseudonyms, dNodes, or any combination of these for a dNode that you specify. This function allows you to specify a starting point for the enumeration, a sort indicator (by name or by type), and a sort direction, as well as matching criteria for both names and types.

Note that a given catalog may not support the sort indicator that you specify. For example, a catalog may support an ordered enumeration by creation times, but not a sort by name or by type. Your results would come back in an unspecified order. (A catalog indicates its sorting capabilities through its feature flags. See "Feature Flag Bit Array" beginning on page 8-28 for more information.)

The sort order of the data returned to you is determined by the target catalog's sorting capabilities and the value you provide in the `sortDirection` field. If the catalog supports sorting by name or sorting by type, the data is sorted in alphabetical or reverse-alphabetical order. If the catalog supports an unspecified ordered enumeration, the catalog determines the meaning of a forward or backward order. For example, if a

catalog supports only an ordered enumeration by creation times, it may return the data in a most recent first or oldest first order.

PowerShare and personal catalogs do not provide secondary sorting. If you specify sorting by name and there are several entities with the same name, those entities are not additionally sorted by type. Similarly, if you specify sorting by type, entities of the same type are not additionally sorted by name. Some external catalogs may have a secondary sort capability; however, the `DirEnumerateGet` function does not provide a way for you to specify a secondary sort order.

Note

The `enumFlags` field indicates the type of entity about which you want information. You can set it to any combination of the mask constants `kEnumDistinguishedNameMask`, `kEnumAliasMask`, `kEnumPseudonymMask`, and `kEnumDNodeMask` to request information about records, aliases, pseudonyms, and dNodes, respectively. If you want information about all visible entities, set the mask to `kEnumAllMask`. If you want information about all entities, visible and invisible, set the mask to `kEnumInvisibleMask`. u

If the `DirEnumerateGet` function is enumerating dNodes, it obtains a dNode name and number for each dNode and places the names and numbers in your buffer. If the `DirEnumerateGet` function is enumerating records, aliases, or pseudonyms, it obtains a local record ID for each record, alias, or pseudonym and places these IDs in your buffer. The function provides only whole units of information for each entity. That is, it will not provide the name for a dNode without also providing the dNode number. Similarly, it will not provide the creation ID for a record without also providing its name and type. If your buffer is not large enough to contain all of the information requested, the `DirEnumerateGet` function provides complete information on as many records, aliases, pseudonyms, or dNodes as will fit and returns the `kOCEMoreData` result code.

When the function completes with either the `noErr` or `kOCEMoreData` result codes, you use a pointer to your buffer as input to the `DirEnumerateParse` function, which extracts the information from the buffer.

If the `DirEnumerateGet` function returns the `kOCEMoreData` result code, you can request additional information by calling it again after calling the `DirEnumerateParse` function. In your next call to the `DirEnumerateGet` function, for the value of the `startingPoint` field, use the value that your callback routine last received in the `enumSpec` parameter. Use the same values for the `aRLI`, `nameMatchString`, and `typesList` fields that you used in your original call to the `DirEnumerateGet` function. The `DirEnumerateGet` function continues the enumeration starting with the next entity as determined by the value of the `includeStartingPoint` field.

To enumerate the contents of the root node of a PowerShare or external catalog, construct a `PackedRLI` data structure in which the dNode number is set to `kRootDNodeNumber` and the pointer to the pathname is set to `nil`. Then set the `aRLI` field to point to your `PackedRLI` data structure.

SPECIAL CONSIDERATIONS

If you target a PowerShare or personal catalog and you specify sorting by type, you can provide only one type in the types list. If you provide more than one type, the function returns an error.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x111

RESULT CODES

`noErr`	0	No error
`kOCEReadAccessDenied`	–1540	Identity lacks read access privileges
`kOCEUnknownID`	–1567	Authentication identity is not valid
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can't find specified dNode
`kOCEMoreData`	–1623	More data available
`kOCEStreamCreationErr`	–1625	Error in creating connection to server

SEE ALSO

The `PackedRLI` data structure is described in the chapter "AOCE Utilities" in this book.

To create a `PackedRLI` data structure use the `OCEPackRLI` utility routine, also described in the chapter "AOCE Utilities."

The `DirEnumSpec` data structure is described on page 8-35.

The `DirEnumerateParse` function is described next.

Feature flags are described in "Feature Flag Bit Array" beginning on page 8-28.

The enumeration mask constants are described in the section "The Enumeration Choice Type" on page 8-34.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesGet` function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" beginning on page 8-20.

DirEnumerateParse

The `DirEnumerateParse` function parses the data returned by the `DirEnumerateGet` function and returns information on each record, alias, pseudonym, or dNode by repeatedly calling your callback routine.

```
pascal OSErr DirEnumerateParse (DirParamBlockPtr paramBlock,
                                Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRLI`	`PackedRLIPtr`	Target dNode
→	`eachEnumSpec`	`ForEachDirEnumSpec`	Your callback routine
→	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRLI` The pointer to the dNode for which you want a list of records, aliases, pseudonyms, or dNodes. Use the same value that you provided to the associated `DirEnumerateGet` function.

`eachEnumSpec` The pointer to your callback routine. The function declaration for this routine is described on page 8-157.

`getBuffer` A pointer to the buffer containing the information to parse. Use the same buffer that you provided to the `DirEnumerateGet` function.

`getBufferSize` The number of bytes in the buffer. Use the same value that you provided to the associated `DirEnumerateGet` function.

DESCRIPTION

You call the `DirEnumerateParse` function to extract the information placed in your buffer by the `DirEnumerateGet` function. You must provide a callback routine that the `DirEnumerateParse` function calls for each record, alias, pseudonym, or dNode about which there is information in the buffer. The `DirEnumerateParse` function provides the dNode name and number if the entity about which it returns information is a dNode. It provides a local record ID if the entity is a record, an alias, or a pseudonym. See the description of your callback routine on page 8-157 for more information.

The `DirEnumerateParse` function completes when it has finished parsing the contents of your buffer or when your callback routine returns `true`. The function returns the `kOCEMoreData` result code if it reaches the end of the buffer and finds that the `DirEnumerateGet` function did not return all the data requested. To continue the enumeration, call the `DirEnumerateGet` function again. For the value of the `startingPoint` field, use the value that your callback routine last received in the `enumSpec` parameter.

If your callback routine returns `true`, the `DirEnumerateParse` function completes with the `noErr` result code.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x101

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEMoreData`	–1623	More data available

SEE ALSO

The function declaration for your callback routine is described on page 8-157.

The `DirEnumerateGet` function is described on page 8-57.

The `PackedRLI` data structure is described in the chapter “AOCE Utilities” in this book.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesParse` function) when the buffer is too small to hold all the information you requested, see “Getting Attribute Type Information” beginning on page 8-20.

DirGetDNodeMetaInfo

The `DirGetDNodeMetaInfo` function returns a numeric value that you can use to determine whether a dNode has changed since you last called this function.

```
pascal OSErr DirGetDNodeMetaInfo (DirParamBlockPtr paramBlock,
                                  Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`pRLI`	`PackedRLIPtr`	Target dNode
←	`metaInfo`	`DirMetaInfo`	Comparison value

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`pRLI` A pointer to packed record location information that identifies the dNode to which the request applies. The function ignores this field when you provide a non-zero value in the `dsRefNum` field to specify a personal catalog.

`metaInfo` A numeric value that the `DirGetDNodeMetaInfo` function returns. The Catalog Manager updates this value when a catalog node changes. You use it to determine if the catalog node has changed.

DESCRIPTION

You call the `DirGetDNodeMetaInfo` function to find out if there has been a change in the content of a dNode that you specify. The function returns the `metaInfo` value associated with the dNode. You must call the function once to get an initial value. When you call the function again, compare the initial value with the new value. If the values match, the dNode has not changed since your previous call to the `DirGetDNodeMetaInfo` function. Any change in the information associated with that

dNode causes the value of the `metaInfo` field to change. Records, aliases, pseudonyms, or dNodes may have been added, deleted or renamed. Attribute types or attribute values may have been added, deleted, or changed. Access controls for the dNode, its records, or attribute types may have changed.

If you detect a change in a dNode, you should do whatever is appropriate in your application to update the information you need. For example, you can call the `DirEnumerate` function to retrieve current information for the dNode. If your application is displaying information about the dNode, you can refresh your window.

The `metaInfo` field contains the following structure:

```
struct DirMetaInfo {
   unsigned long  info[4];
};
```

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x118

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can't find specified dNode

SEE ALSO

The `PackedRLI` data structure is described in the chapter "AOCE Utilities" in this book.

You can create a `PackedRLI` data structure with the `OCEPackRLI` utility routine. It is also described in the chapter "AOCE Utilities."

DirMapDNodeNumberToPathName

The `DirMapDNodeNumberToPathName` function returns pathname information for a dNode that you specify.

```
pascal OSErr DirMapDNodeNumberToPathName (DirParamBlockPtr
                                          paramBlock,Boolean async);
```

paramBlock
Pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	serverHint	AddrBlock	AppleTalk address of the PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
→	directoryName	DirectoryNamePtr	Catalog name
→	discriminator	DirDiscriminator	Discriminator value
→	dNodeNumber	DNodeNum	The dNode number
↔	path	PackedPathNamePtr	Your buffer
↔	lengthOfPathName	unsigned short	Length of your buffer, pathname

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

directoryName A pointer to the name of the catalog in which the target dNode resides.

discriminator The discriminator value of the catalog in which the dNode resides. This value differentiates two or more catalogs with the same name.

dNodeNumber The dNode number whose pathname you want to obtain.

path A pointer to a buffer in which the function stores packed pathname information. You must provide a buffer big enough to hold all of the path information that the function returns. A buffer size of `kPathNameMaxBytes` can hold any packed pathname. Before you can read the packed pathname information, you must unpack it with the `OCEUnpackPathName` routine.

lengthOfPathName
This field is used for both input and output. You set this field to the size of your buffer in bytes before you call the `DirMapDNodeNumberToPathName` function. The function sets this field to the number of bytes in the pathname information that you requested. If the function completes successfully, this field represents the number of bytes that the function placed in your buffer. If your buffer is too small to hold the entire pathname, the function returns a `kOCEMoreData` result code and does not store any information in your buffer. If this occurs, the value in this field represents the minimum size of a buffer capable of holding the packed pathname information. You must increase the size of your buffer to at least the minimum size and call the function again.

DESCRIPTION

You call the `DirMapDNodeNumberToPathName` function when you know a dNode number and want to obtain the corresponding full pathname. If the catalog you specify does not support dNode numbers (this includes all personal catalogs), the function returns the `kOCENoSuchDNode` error.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x123

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEUnknownID`	–1567	Authentication identity is not valid
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can't find specified dNode
`kOCEMoreData`	–1623	Buffer too small
`kOCEStreamCreationErr`	–1625	Error in creating connection to server

SEE ALSO

The `OCEUnpackPathName` routine is described in the chapter "AOCE Utilities" in this book.

The `PackedPathName` data structure is also described in the chapter "AOCE Utilities."

To obtain the dNode number when you know the pathname, use the `DirMapPathNameToDNodeNumber` function, described next.

DirMapPathNameToDNodeNumber

The `DirMapPathNameToDNodeNumber` function returns the dNode number for a dNode identified by a pathname and catalog that you specify.

```
pascal OSErr DirMapPathNameToDNodeNumber
                    (DirParamBlockPtr paramBlock, Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	serverHint	AddrBlock	AppleTalk address of the PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
→	directoryName	DirectoryNamePtr	Catalog name
→	discriminator	DirDiscriminator	Discriminator value
←	dNodeNumber	DNodeNum	DNode number
→	path	PackedPathNamePtr	Pathname

See "The Parameter Block Header" on page 8-32 for descriptions of the ioCompletion, ioResult, serverHint, dsRefNum, identity, and clientData fields.

Field descriptions

directoryName	The name of the catalog containing the dNode whose dNode number you want to obtain.
discriminator	The discriminator value of the catalog containing the dNode whose dNode number you want to obtain. This value differentiates two or more catalogs with the same name.
dNodeNumber	A number that uniquely identifies a dNode within a catalog. The function returns this number.
path	A pointer to the buffer that contains the packed pathname for the dNode whose dNode number you want to obtain. You create a packed pathname with the OCEPackPathName utility routine.

DESCRIPTION

You call the DirMapPathNameToDNodeNumber function when you know the path of a particular dNode and you want to obtain its dNode number. If the catalog you specify does not support dNode numbers (this includes all personal catalogs), the function returns the kOCENoSuchDNode error.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	0x122

RESULT CODES

noErr	0	No error
kOCEParamErr	–50	Invalid parameter
kOCETargetDirectoryInaccessible	–1613	Target catalog is not currently available
kOCENoSuchDNode	–1615	Can't find specified dNode

SEE ALSO

The OCEPackPathName routine is described in the chapter "AOCE Utilities" in this book.

The PackedPathName data structure is also described in the chapter "AOCE Utilities."

To obtain the pathname when you know the DNode number, use the DirMapDNodeNumberToPathName function, described on page 8-65.

DirGetDNodeInfo

The DirGetDNodeInfo function indicates whether a dNode that you specify can contain records and whether it is a foreign node.

```
pascal OSErr DirGetDNodeInfo (DirParamBlockPtr paramBlock,
                              Boolean async);
```

paramBlock
Pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	serverHint	AddrBlock	AppleTalk address of the PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
→	pRLI	PackedRLIPtr	Target dNode
←	descriptor	DirNodeKind	DNode descriptor
↔	networkSpec	NetworkSpecPtr	Network name

See "The Parameter Block Header" on page 8-32 for descriptions of the ioCompletion, ioResult, serverHint, dsRefNum, identity, and clientData fields.

Field descriptions

`pRLI`	A pointer to packed record location information that identifies the catalog and dNode to which the request applies. The function ignores this field when you provide a nonzero value in the `dsRefNum` field to specify a personal catalog.
`descriptor`	A value that the function returns by which you can determine whether the dNode you specified can contain records and whether it is a foreign node. Use the mask `kCanContainRecords` to determine whether the dNode can contain records. To find out if the dNode you specified is a foreign node, use the mask `kForeignNode`.
`networkSpec`	A pointer to the name of the network in which the dNode resides. The function sets this field only if the dNode can contain records.

DESCRIPTION

The `DirGetDNodeInfo` function is usually called by PowerShare Admin software. Most applications do not need to use this function. However, messaging service access modules may call it to determine if a dNode is a foreign dNode. Foreign dNodes represent external messaging systems that are connected to an AOCE system.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x125

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can't find specified dNode
`kOCEStreamCreationErr`	–1625	Error in creating connection to server

SEE ALSO

The `NetworkSpec` data structure is described in the chapter "AOCE Utilities" in this book.

The `PackedRLI` data structure is also described in the chapter "AOCE Utilities."

You can create a `PackedRLI` data structure with the `OCEPackRLI` utility routine. It is also described in the chapter "AOCE Utilities."

Maintaining the PowerTalk Setup Catalog

A catalog that is listed in the PowerTalk Setup catalog is available for use by any application that uses the Catalog Manager. Setup templates use the `DirAddADAPDirectory` and `DirRemoveDirectory` routines to add and remove records that represent PowerShare catalogs from the PowerTalk Setup catalog. The `DirRemoveDirectory` function also removes records that represent external catalogs. For information on adding records that represent external catalogs, see the chapter "Access Module Setup" in *Inside Macintosh: AOCE Service Access Modules*.

Note

A shorthand way of saying that a record representing a catalog is added or removed from the PowerTalk Setup catalog is to say that the *catalog* is added or removed from the PowerTalk Setup catalog. However, a catalog itself is never added or removed from the PowerTalk Setup catalog; only records that represent catalogs are added or removed. u

The `DirNetSearchADAPDirectoryGet` and `DirNetSearchADAPDirectoryParse` routines work together to provide information about all of the PowerShare catalogs on a network.

If you know a PowerShare catalog's name and discriminator value, you can call the `DirFindADAPDirectoryByNetSearch` function to locate a catalog and add it to the PowerTalk Setup catalog if you choose.

The `DirAddADAPDirectory` and `DirFindADAPDirectoryByNetSearch` functions provide the option of making a PowerShare catalog temporarily available for the PowerTalk Key Chain's use, without adding it to the PowerTalk Setup catalog. This condition of private availability lasts only until the computer is restarted.

The `DirGetOCESetupRefnum` function provides the reference number of the PowerTalk Setup catalog.

To get information about all of the catalogs that are listed in the PowerTalk Setup catalog, you can call the `DirEnumerateDirectoriesGet` function. It is described in the section "Getting Information About Catalogs" beginning on page 8-38.

DirAddADAPDirectory

The `DirAddADAPDirectory` function makes a PowerShare catalog that you specify available for use with other Catalog Manager functions. At your option, it also adds the catalog to the PowerTalk Setup catalog.

```
pascal OSErr DirAddADAPDirectory (DirParamBlockPtr paramBlock,
                                  Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`clientData`	`long`	You define this field
→	`directoryName`	`DirectoryNamePtr`	Name of the catalog
→	`discriminator`	`DirDiscriminator`	Discriminator value
→	`addToOCESetup`	`Boolean`	Add to PowerTalk Setup?
←	`directoryRecordCID`	`CreationID`	Creation ID of catalog

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, and `clientData` fields.

Field descriptions

`directoryName` A pointer to the name of the PowerShare catalog that you want to use.

`discriminator` A value that differentiates two or more catalogs with the same name.

`addToOCESetup` A Boolean value that specifies whether you want to add the catalog to the PowerTalk Setup catalog. Set this field to `true` if you want to add the catalog to the PowerTalk Setup catalog.

`directoryRecordCID`
The creation ID of the record representing the PowerShare catalog that you specify in the `directoryName` and `discriminator` fields. The function creates a record for the catalog, adds it to the PowerTalk Setup catalog, and returns the record creation ID only when you set `addToOCESetup` to `true`.

DESCRIPTION

You call the `DirAddADAPDirectory` function when you want to make a PowerShare catalog that is not listed in the PowerTalk Setup catalog available for use with other Catalog Manager functions. You must specify a valid AppleTalk address in the `serverHint` field in the parameter block header for this function. If the `serverHint` field is set to `nil` or does not point to a PowerShare server for that catalog, the `DirAddADAPDirectory` function returns an error.

Note

The PowerTalk Key Chain uses this function. In general, there is no reason for an application to use this function. u

When the function completes successfully, you can use the catalog with other Catalog Manager functions. If you set the `addToOCESetup` field to `true`, the function adds the catalog to the PowerTalk Setup catalog. All catalogs listed in the PowerTalk Setup catalog are visible to the `DirEnumerateDirectories` function and thus available to any application using the services of the Catalog Manager. Furthermore, the catalogs listed in the PowerTalk Setup catalog remain available until they are explicitly removed by the `DirRemoveDirectory` function.

If you set `addToOCESetup` to `false`, the `DirAddADAPDirectory` function makes the catalog available to you privately, and you may specify it when you call other Catalog Manager functions. This availability lasts until the computer is restarted. Once the computer is restarted, the catalog is no longer available to you. A catalog that you do not add to the PowerTalk Setup catalog is not visible to the `DirEnumerateDirectories` function; therefore, it is not available to other applications nor is it visible to a user.

If you want to use a PowerShare catalog that is not listed in the PowerTalk Setup catalog, but you do not know the address of a PowerShare server for that catalog, you can call the `DirFindADAPDirectoryByNetSearch` function.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x137

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEAlreadyExists`	–1510	The catalog being added already exists
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available

SEE ALSO

The `DirEnumerateDirectoriesGet` function is described on page 8-38.

The `DirFindADAPDirectoryByNetSearch` function is described next.

The `DirRemoveDirectory` function is described on page 8-79.

For more information on the PowerTalk Setup catalog, see “Identities and the PowerTalk Setup Catalog” on page 8-8.

DirFindADAPDirectoryByNetSearch

The `DirFindADAPDirectoryByNetSearch` function locates a PowerShare catalog that you specify on a network and makes it available for use with other Catalog Manager functions. At your option, it also adds the catalog to the PowerTalk Setup catalog.

```
pascal OSErr DirFindADAPDirectoryByNetSearch
                                    (DirParamBlockPtr paramBlock,
                                    Boolean async);
```

`paramBlock`
Pointer to a parameter block.

`async` A Boolean value that specifies whether the function is to be executed asynchronously. Set this parameter 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
→	`directoryName`	`DirectoryNamePtr`	Catalog name
→	`discriminator`	`DirDiscriminator`	Discriminator value
→	`addToOCESetup`	`Boolean`	Add to setup list?
←	`directoryRecordCID`	`CreationID`	Creation ID of catalog record

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, and `clientData` fields.

Field descriptions

`directoryName` The name of the PowerShare catalog that you want to find.

`discriminator` The discriminator value for the named catalog. This value differentiates two or more catalogs with the same name.

`addToOCESetup` A Boolean value that indicates whether you want to add the catalog to the PowerTalk Setup catalog. Set this field to `true` if you want to the catalog to the PowerTalk Setup catalog.

`directoryRecordCID`
The creation ID of the record representing the catalog that you specify in the `directoryName` and `discriminator` fields. The function creates a record for the catalog, adds it to the PowerTalk Setup catalog, and returns the record creation ID only when you set `addToOCESetup` to `true`.

DESCRIPTION

You call the `DirFindADAPDirectoryByNetSearch` function when you want to use a PowerShare catalog and the catalog is not listed in the PowerTalk Setup catalog. You must provide the catalog name and discriminator value. The function searches the network for the catalog.

Note

The PowerTalk Key Chain uses this function. In general, there is no reason for an application to use this function. u

If the function finds the catalog, you can use it with other Catalog Manager functions. If you set the `addToOCESetup` field to `true`, the function adds the catalog to the PowerTalk Setup catalog. All catalogs listed in the PowerTalk Setup catalog are visible to the `DirEnumerateDirectories` function and thus are available to any application using the services of the Catalog Manager. Furthermore, the catalogs listed in the PowerTalk Setup catalog remain available until they are explicitly removed by the `DirRemoveDirectory` function.

If you set `addToOCESetup` to `false`, the `DirFindADAPDirectoryByNetSearch` function makes the catalog available to you privately and you may specify it when you call other Catalog Manager functions. This availability lasts until the computer is restarted. Once the computer is restarted, the catalog is no longer available to you. Catalogs that you do not choose to add to the PowerTalk Setup catalog are not visible to the `DirEnumerateDirectories` function; therefore, they are not available to other applications nor are they visible to a user.

SPECIAL CONSIDERATIONS

The `DirFindADAPDirectoryByNetSearch` function makes a networkwide search for the PowerShare catalog that you specify. Because this function consumes expensive network resources, you should use it very sparingly. If you know a catalog's name, discriminator value, and the Apple talk address of a PowerShare server for the catalog, you should use the `DirAddADAPDirectory` function. It too, makes a catalog available for your use and can add it to the PowerTalk Setup catalog.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x107

RESULT CODES

`noErr`	0	No error
`kOCEAlreadyExists`	–1510	The catalog being added already exists
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available

SEE ALSO

The `DirAddADAPDirectory` function is described on page 8-71.

For more information on the PowerTalk Setup catalog, see "Identities and the PowerTalk Setup Catalog" on page 8-8.

The `DirRemoveDirectory` function is described on page 8-79.

The `DirNetSearchADAPDirectoriesGet` function, described next, retrieves the return address of a PowerShare catalog on a network. By saving and using this address you can eliminate the need to search for a particular catalog with the `DirFindADAPDirectoryByNetSearch` function each time the computer is rebooted.

DirNetSearchADAPDirectoriesGet

The `DirNetSearchADAPDirectoriesGet` function returns information about the PowerShare catalogs on a network.

```
pascal OSErr DirNetSearchADAPDirectoriesGet
                                    (DirParamBlockPtr paramBlock,
                                    Boolean async);
```

`paramBlock`
Pointer to a parameter block.

`async` A Boolean value that specifies whether the function is to be executed asynchronously. Set this parameter 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
↔	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, and `clientData` fields.

Field descriptions

`getBuffer`	A pointer to a buffer in which the function stores information about each PowerShare catalog on the network: its name, discriminator value, feature flags, and the AppleTalk address of its server. You provide this buffer.
`getBufferSize`	The number of bytes in the buffer.

DESCRIPTION

You call the `DirNetSearchADAPDirectoriesGet` function to obtain a list of the PowerShare catalogs on a network.

If the buffer you provide is not large enough to contain all of the information, the `DirNetSearchADAPDirectoriesGet` function returns the `kOCEMoreData` result code.

If your buffer is too small to hold all of the information you requested, you must allocate a bigger buffer and call the `DirNetSearchADAPDirectoriesGet` function again to get it all. At each call, the function attempts to return all the information you have requested, starting from the beginning. Therefore, you will get duplicate information on subsequent calls.

When the function completes with either the `noErr` or `kOCEMoreData` result codes, you use a pointer to your buffer as input to the `DirNetSearchADAPDirectoriesParse` function, which extracts the catalog information from the buffer.

SPECIAL CONSIDERATIONS

The `DirNetSearchADAPDirectoriesGet` function makes a networkwide search for PowerShare catalogs. Because this search consumes expensive network resources, you should use this function very sparingly.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x108

RESULT CODES

`noErr`	0	No error
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `DirNetSearchADAPDirectoriesParse` function is described next.

DirNetSearchADAPDirectoriesParse

The `DirNetSearchADAPDirectoriesParse` function parses the data returned by the `DirNetSearchADAPDirectoriesGet` function and returns information on each PowerShare catalog by repeatedly calling your callback routine.

```
pascal OSErr DirNetSearchADAPDirectoriesParse
                                    (DirParamBlockPtr paramBlock,
                                    Boolean async);
```

`paramBlock`
Pointer to a parameter block.

`async` A Boolean value that specifies whether the function is to be executed asynchronously. Set this parameter 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
→	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer
→	`eachADAPDirectory`	`ForEachADAPDirectory`	Your callback routine

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, and `clientData` fields.

Field descriptions

`getBuffer` A pointer to the buffer containing the information to parse. Use the same buffer that you provided to the `DirNetSearchADAPDirectoriesGet` function.

`getBufferSize` The number of bytes in the buffer. Use the same value that you provided to the `DirNetSearchADAPDirectoriesGet` function.

`eachADAPDirectory`
A pointer to your callback routine. The function declaration for this routine is described on page 8-160.

DESCRIPTION

You call the `DirNetSearchADAPDirectoriesParse` function to extract the information about PowerShare catalogs placed in your buffer by the `DirNetSearchADAPDirectoriesGet` function. You must provide a callback routine that the `DirNetSearchADAPDirectoriesParse` function calls for each set of catalog information in the buffer. The `DirNetSearchADAPDirectoriesParse` function passes your callback routine the following information about each catalog: a catalog name and discriminator value, its feature flags, and the AppleTalk address of a PowerShare server for that catalog.

The DirNetSearchADAPDirectoriesParse function completes when it has finished parsing the contents of your buffer or when your callback routine returns true. The function returns the kOCEMoreData result code if it reaches the end of the buffer and finds that the DirNetSearchADAPDirectoriesGet function did not return all the data requested.

If your callback routine returns true, the DirNetSearchADAPDirectoriesParse function completes with the noErr result code.

Once you have the name and discriminator value for a PowerShare catalog, you can call the DirAddADAPDirectory function to make the catalog available for use and, if you choose, to add it to the PowerTalk Setup catalog.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	0x105

RESULT CODES

noErr	0	No error
kOCEMoreData	–1623	More data available

SEE ALSO

The function declaration for your callback routine is described on page 8-160.

The DirNetSearchADAPDirectoriesGet function is described on page 8-76.

The DirAddADAPDirectory function is described on page 8-71.

DirRemoveDirectory

The DirRemoveDirectory function removes a record that represents a catalog from the PowerTalk Setup catalog.

```
pascal OSErr DirRemoveDirectory (DirParamBlockPtr paramBlock,
                                 Boolean async);
```

paramBlock
: Pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	serverHint	AddrBlock	AppleTalk address of the PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
→	directoryRecordCID	CreationID	Creation ID of catalog

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`directoryRecordCID`
The creation ID of a record in the PowerTalk Setup catalog. This record represents the catalog that you want to remove.

DESCRIPTION

You call the `DirRemoveDirectory` function to remove an external or PowerShare catalog that you specify from the PowerTalk Setup catalog.

A catalog that you remove from the PowerTalk Setup catalog is no longer visible to the `DirEnumerateDirectoriesGet` function. You cannot specify it in calls to other Catalog Manager functions until you again add it to the PowerTalk Setup catalog or make it available privately to your application through the `DirAddADAPDirectory` or the `DirFindADAPDirectoryByNetSearch` function.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	0x135

RESULT CODES

noErr	0	No error
kOCEDirectoryNotFoundErr	–7945	Can't find specified catalog

SEE ALSO

Use the `DirAddDSAMDirectory` function, which is described in the chapter "Catalog Service Access Modules" in *Inside Macintosh: AOCE Service Access Modules*, to add a catalog to the PowerTalk Setup catalog.

You can also use the `DirAddADAPDirectory` function, which is described on page 8-71, to add a catalog to the PowerTalk Setup catalog.

The DirFindADAPDirectoryByNetSearch function is described on page 8-74.

The DirEnumerateDirectoriesGet function is described on page 8-38.

DirGetOCESetupRefnum

The DirGetOCESetupRefnum function returns the reference number of the PowerTalk Setup catalog.

```
pascal OSErr DirGetOCESetupRefnum (DirParamBlockPtr paramBlock,
                                   Boolean async);
```

paramBlock
Pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
←	dsRefNum	short	PowerTalk Setup catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
←	oceSetupRecordCID	CreationID	Creation ID of the record identifying the PowerTalk Setup catalog

See "The Parameter Block Header" on page 8-32 for descriptions of the ioCompletion, ioResult, dsRefNum, identity, and clientData fields.

Field descriptions

oceSetupRecordCID
The creation ID of the record identifying the PowerTalk Setup catalog.

DESCRIPTION

You call the DirGetOCESetupRefnum function if you need to read from or write to the PowerTalk Setup catalog. The function returns the dsRefNum value for the PowerTalk Setup catalog. You need this value to perform operations on the PowerTalk Setup catalog.

The function also returns the creation ID of the record that contains summary information about the contents of the PowerTalk Setup catalog.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	0x128

RESULT CODES

noErr	0	No error

SEE ALSO

For more information on the PowerTalk Setup catalog and local identity, see "Identities and the PowerTalk Setup Catalog" on page 8-8 as well as the chapter "Authentication Manager" in this book.

Creating, Opening, and Closing Personal Catalogs

A personal catalog is a Hierarchical File System (HFS) file. You can use the functions in this section to create new personal catalogs as well as to open and close existing personal catalogs. In addition, the `DirMakePersonalDirectoryRLI` function provides information you can use to locate a personal catalog that you opened even if it has been closed, moved, or renamed.

You can use File Manager functions to browse for a personal catalog. Use the constants `kPersonalDirectoryFileType` and `kPersonalDirectoryFileCreator` to specify the file type and file creator, respectively, for a personal catalog.

DirCreatePersonalDirectory

The `DirCreatePersonalDirectory` function creates a new personal catalog.

```
pascal OSErr DirCreatePersonalDirectory
                                    (DirParamBlockPtr paramBlock);
```

`paramBlock`
Pointer to a parameter block.

Parameter block

←	ioResult	OSErr	Result code
→	fsSpec	FSSpecPtr	File system specification
→	fdType	OSType	File type
→	fdCreator	OSType	File creator

See "The Parameter Block Header" on page 8-32 for a description of the `ioResult` field.

Field descriptions

`fsSpec`	A pointer to the file system specification record that identifies the personal catalog you want to create. You can obtain the file system specification record from the `FSMakeFSSpec` function.
`fdType`	The file type for the new personal catalog. If you want to create an ordinary personal catalog, set this field to the constant `kPersonalDirectoryFileType`. If you want to create an information card, set this field to the constant `kBusinessCardFileType`.
`fdCreator`	The file creator for the new personal catalog. Set this field to the constant `kPersonalDirectoryFileCreator`, for both an ordinary personal catalog and an information card.

DESCRIPTION

You call the `DirCreatePersonalDirectory` function to create a personal catalog.

You can provide values for the file creator and file type other than those specified in the field descriptions above. However, if you do so, the Finder and AOCE software will not be able to display the icons that represent the personal catalog or information card to the user.

To open the new personal catalog, use the `DirOpenPersonalDirectory` function, described next.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x11F

RESULT CODES

`noErr`	0	No error
`dupFNErr`	–48	Filename already exists
`kOCEParamErr`	–50	Invalid parameter
`dirNFErr`	–120	Catalog not found

SEE ALSO

An information card is a personal catalog containing a single record. For more information about information cards, see the section “Introduction to AOCE Catalogs” beginning on page 8-4.

For information about file system specification records, see the chapter “File Manager” in *Inside Macintosh: Files*.

DirOpenPersonalDirectory

The `DirOpenPersonalDirectory` function opens a personal catalog and returns a reference number for it.

```
pascal OSErr DirOpenPersonalDirectory
                                    (DirParamBlockPtr paramBlock);
```

`paramBlock`
Pointer to a parameter block.

Parameter block

←	`ioResult`	`OSErr`	Result code
←	`dsRefNum`	`short`	Personal catalog reference number
→	`fsSpec`	`FSSpecPtr`	File system specification
→	`accessRequested`	`char`	Permissions requested
←	`accessGranted`	`char`	Permissions granted
←	`features`	`DirGestalt`	Feature flags

See "The Parameter Block Header" on page 8-32 for a description of the `ioResult` and `dsRefNum` fields.

Field descriptions

`fsSpec` A pointer to a file system specification record for the personal catalog that you want to open.

`accessRequested`
The access that you are requesting for this personal catalog. Set this field to `fsRdPerm` if you are requesting permission to read the personal catalog. If you also want permission to write to the personal catalog, set this field to `fsRdWrPerm`.

`accessGranted`
The catalog access that the Catalog Manager grants. The function returns either `fsRdPerm` or `fsRdWrPerm` in this field, granting you read-only or read / write access, respectively.

`features` A set of bit flags indicating the features that the personal catalog supports. The bit flags are described in "Feature Flag Bit Array" beginning on page 8-28.

DESCRIPTION

You call the `DirOpenPersonalDirectory` function to open a personal catalog (including information cards, which are a type of personal catalog). In the `dsRefNum` field of the parameter block header, the function returns the reference number that uniquely identifies the personal catalog. You must use this reference number in all subsequent Catalog Manager requests directed to this personal catalog.

The function also returns the access that you have to the personal catalog file and a set of bit flags that specify what features the personal catalog supports.

SPECIAL CONSIDERATIONS

If the user moves a personal catalog to a computer whose operating system uses a different script system from the one last used to sort the catalog, the personal catalog must be resorted before the Catalog Manager can open it. If the `DirOpenPersonalDirectory` function returns the error `kOCEVersionErr`, you must call the `SDPSortPersonalDirectory` function to resort the personal catalog and then call the `DirOpenPersonalDirectory` function again to open the catalog.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x11E

RESULT CODES

`noErr`	0	No error
`tmfoErr`	–42	Too many files open
`fnfErr`	–43	File not found
`kOCEParamErr`	–50	Invalid parameter
`permErr`	–54	Permissions error
`dirNFErr`	–120	Catalog not found
`kOCEVersionErr`	–1504	Need to sort personal catalog

SEE ALSO

For a description of catalog feature flags, see "Feature Flag Bit Array" beginning on page 8-28.

To close a personal catalog that you have opened, use the `DirClosePersonalDirectory` function, described next.

The `SDPSortPersonalDirectory` function is described in the chapter "Standard Catalog Package" in this book.

For information about file system specifications, see the chapter "File Manager" in *Inside Macintosh: Files*.

DirClosePersonalDirectory

The `DirClosePersonalDirectory` function closes an open personal catalog.

```
pascal OSErr DirClosePersonalDirectory
                                    (DirParamBlockPtr paramBlock);
```

`paramBlock`
Pointer to a parameter block.

Parameter block

←	ioResult	OSErr	Result code
→	dsRefNum	short	Reference number

Field descriptions

ioResult	The result of the function.
dsRefNum	The catalog reference number that identifies the personal catalog to be closed. After this function successfully completes execution, that reference number is no longer valid.

DESCRIPTION

You call the DirClosePersonalDirectory function to close any personal catalog that has been opened by the DirOpenPersonalDirectory function. This includes information cards, which are a type of personal catalog.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	0x131

RESULT CODES

noErr	0	No error
kOCERefNumBad	–1624	Reference number is not valid

SEE ALSO

The DirOpenPersonalDirectory function is described on page 8-84.

DirMakePersonalDirectoryRLI

The DirMakePersonalDirectoryRLI function provides you with packed record location information for a personal catalog that you specify.

```
pascal OSErr DirMakePersonalDirectoryRLI
                                    (DirParamBlockPtr paramBlock);
```

paramBlock
Pointer to a parameter block.

Parameter block

←	ioResult	OSErr	Result code
→	dsRefNum	short	Reference number
→	fromFSSpec	FSSpecPtr	Catalog to search
→	pRLIBufferSize	unsigned short	Size of your buffer
←	pRLISize	unsigned short	Size of the PackedRLI data structure
↔	pRLI	PackedRLIPtr	Your buffer

See "The Parameter Block Header" on page 8-32 for a description of the ioResult and dsRefNum fields.

Field descriptions

fromFSSpec A pointer to a file system specification record. It specifies the folder within which the personal catalog must reside for the Alias Manager to find it. Set this field to nil if you do not want to limit the Alias Manager's search.

pRLIBufferSize

The size, in bytes, of the buffer that you provide for the packed record location information.

pRLISize The length of the packed record location information. If the function returns the noErr result code, this is the number of bytes of data that the function placed in your buffer. If the function returns the kOCEMoreData result code, you can use the value of this field to determine how large a buffer is required, allocate a buffer of that size, and call the function again.

pRLI A pointer to the buffer in which the function stores the packed record location information. You provide this buffer.

DESCRIPTION

You call the DirMakePersonalDirectoryRLI function to obtain record location information for a personal catalog. You identify the personal catalog about which you want record location information by setting the dsRefNum field in the parameter block header to the personal catalog's reference number. You obtain the reference number from the DirOpenPersonalDirectory function.

You can use the record location information to find the personal catalog if it has been closed, moved, or renamed. For example, if you are developing an electronic mail application, you might have to handle the following sequence of events. A user may open a personal catalog and copy an address from it to a letter being prepared. The user may then close the personal catalog and send the letter at a later time. To send the letter, you may need additional information from the personal catalog. You can locate it using the record location information that the DirMakePersonalDirectoryRLI function returns to you.

To make sure that you can locate a personal catalog even if it has been moved, renamed, or closed, call the `DirMakePersonalDirectoryRLI` function after opening the personal catalog. The function actually creates an alias for the personal catalog and returns record location information for the alias. To find the personal catalog, pass the `PackedRLI` data structure returned by the `DirMakePersonalDirectoryRLI` function to the `OCEExtractAlias` utility routine, which returns an alias record. Pass that alias record to the `ResolveAlias` function, which locates the personal catalog and returns its file system specification. You can then open the personal catalog with the `DirOpenPersonalDirectory` function using the `FSSpec` data structure returned by the `ResolveAlias` function.

If you provided a file system specification record in the `fromFSSpec` field, the `ResolveAlias` routine looks for the catalog only in that folder and any folders enclosed within it. This results in a speedier search. However, if the personal catalog has been moved elsewhere, the `ResolveAlias` routine cannot find it and returns an error.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	0x132

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEMoreData`	–1623	More data available
`kOCERefNumBad`	–1624	Reference number is not valid

SEE ALSO

The `PackedRLI` data structure is described in the chapter “AOCE Utilities” in this book.

You can create a `PackedRLI` data structure with the `OCEPackRLI` utility routine. It is also described in the chapter “AOCE Utilities.”

The `OCEExtractAlias` utility routine is also described in the chapter “AOCE Utilities.”

The `ResolveAlias` routine is described in the chapter “Alias Manager” in *Inside Macintosh: Files*.

The `DirOpenPersonalDirectory` function is described on page 8-84.

Managing Records

The functions described in this section provide the following services:

- adding and deleting records
- adding and deleting pseudonyms
- listing the pseudonyms for a record
- detecting a change in a record
- setting and obtaining a record's name and type
- adding an alias for a record

You can also list all of the records, pseudonyms, and aliases within a dNode. To do this, use the `DirEnumerateGet` and `DirEnumerateParse` functions described in "Getting Information About dNodes" beginning on page 8-56.

DirAddRecord

The `DirAddRecord` function adds a new record to a dNode in a catalog that you specify.

```
pascal OSErr DirAddRecord (DirParamBlockPtr paramBlock,
                           Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
↔	`aRecord`	`RecordIDPtr`	Target record
→	`allowDuplicate`	`Boolean`	Allow duplicate record name?

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a partially specified record ID for the record that you want to add. If you want to add a record to a personal catalog, you must provide the record's name and type. If you want to add a record to a PowerShare or external catalog, you must specify everything in the record ID except the `cid` field. The function places the creation ID for the new record in the `cid` field. If a catalog does not support creation IDs, the function sets the `cid` field to 0.

`allowDuplicate`
A Boolean value specifying whether the function should create a record if another record, alias, or pseudonym with the same name and type already exists. Set this field to `true` if you want the function to create the new record without checking the dNode for a duplicate name and type. If you set the `allowDuplicate` field to `false`, the function checks the name and type of all records, aliases, and pseudonyms in the dNode and returns the `kOCENoDupAllowed` result code if it finds a duplicate name.

DESCRIPTION

You call the `DirAddRecord` function to add a record to a dNode.

SPECIAL CONSIDERATIONS

If you set the `allowDuplicate` field to `false`, the function will not add the record if a record with the same name and type already exists. However, this does not guarantee that a duplicate record will not be created by a requester who sets the `allowDuplicate` field to `true`. The prohibition on duplicates applies only at the time you call this function; it does not guarantee that the record name and record type will be unique at a later time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0109

RESULT CODES

`noErr`	0	No error
`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

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

DirDeleteRecord

The `DirDeleteRecord` function deletes the record that you specify.

```
pascal OSErr DirDeleteRecord (DirParamBlockPtr paramBlock,
                              Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record you want to delete. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.

DESCRIPTION

You call `DirDeleteRecord` to delete a record within a catalog. The function also deletes any pseudonyms for the record. The function does not automatically delete aliases that point to the record you want to delete.

Note
Although, you can call the `DirDeleteRecord` function to delete a record that is an alias for another record, there is no way to automatically identify any aliases that point to a record you have deleted. The situation is much the same as for HFS files. When a file is deleted, its aliases remain intact, but of course the aliases return an error if someone attempts to use them. u

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$010A

RESULT CODES

`noErr`	0	No error
`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
`kOCEBadRecordId`	–1617	Record name or record type doesn't match creation ID
`kOCENoSuchRecord`	–1618	Can't find specified record

SEE ALSO

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

DirGetRecordMetaInfo

The `DirGetRecordMetaInfo` function returns a numeric value that you can use to determine if a record has changed since you last called this function.

```
pascal OSErr DirGetRecordMetaInfo (DirParamBlockPtr paramBlock,
                                   Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	serverHint	AddrBlock	AppleTalk address of the PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
→	aRecord	RecordIDPtr	Target record
←	metaInfo	DirMetaInfo	Comparison value

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record to which the request applies. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.

`metaInfo` A numeric value returned by `DirGetRecordMetaInfo`. The Catalog Manager updates this value when the content of a record changes. You use it to determine if the record has changed.

DESCRIPTION

You call the `DirGetRecordMetaInfo` function to find out if there has been a change in the contents of a record that you specify. The function returns the `metaInfo` value associated with the record. You must call the function once to get an initial value. When you call the function again, compare the initial value with the new value. If the values match, the record has not changed since your previous call to the `DirGetRecordMetaInfo` function. Any change to the value of the `metaInfo` field indicates a change in the information associated with that record. Attribute types may have been added or deleted; attribute values may have been added, deleted, or changed; or access controls for the records or its attribute types may have changed.

If you detect a change in a record, you should do whatever is appropriate in your application to update the information you need. For example, you can call the `DirEnumerateAttributeTypes` function, followed by the `DirLookupGet` and `DirLookupParse` functions, to retrieve current information about attribute types and values in the record. If your application is displaying information about the record, you can refresh your window.

The `metaInfo` field contains the following structure:

```
struct DirMetaInfo {
   long  info[4];
};
```

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0116

RESULT CODES

noErr	0	No error
kOCEParamErr	–50	Invalid parameter
kOCETargetDirectoryInaccessible	–1613	Target catalog is not currently available
kOCENoSuchDNode	–1615	Can't find specified dNode
kOCEBadRecordID	–1617	Record name or record type doesn't match creation ID
kOCENoSuchRecord	–1618	Can't find specified record

SEE ALSO

The RecordID data structure is described in the chapter "AOCE Utilities" in this book.

DirGetNameAndType

The DirGetNameAndType function returns a record's name and type.

```
pascal OSErr DirGetNameAndType (DirParamBlockPtr paramBlock,
                                Boolean async);
```

paramBlock
: Pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	serverHint	AddrBlock	AppleTalk address of the PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
↔	aRecord	RecordIDPtr	Target record

See "The Parameter Block Header" on page 8-32 for descriptions of the ioCompletion, ioResult, serverHint, dsRefNum, identity, and clientData fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record whose name and type you are requesting. You must provide the record creation ID. Unless the catalog containing the record is a personal catalog, you must also provide packed record location information. The `name` and `type` buffers that you provide must be large enough to hold a maximum-length `RString` data structure. If the function is successful, it places the record's name and type in these buffers.

DESCRIPTION

If you know the creation ID of a record and the catalog and dNode in which it resides, you can use the `DirGetNameAndType` function to obtain its name and type. A record's name and type may change, but its creation ID always remains the same. You can store the record creation ID as an always valid value and use it as needed to retrieve the changeable name and type.

You may also prefer to store only the record creation ID because it requires less memory and use this function to retrieve the record name and type when you need it.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0114

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEReadAccessDenied`	–1540	Identity lacks read access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can't find specified dNode
`kOCENoSuchRecord`	–1618	Can't find specified record

SEE ALSO

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `RString` data structure is also described in the chapter "AOCE Utilities."

You use the `DirSetNameAndType` function to change a record's name and type. It is described next.

DirSetNameAndType

The `DirSetNameAndType` function changes the name, the type, or both the name and type of a record that you specify.

```
pascal OSErr DirSetNameAndType (DirParamBlockPtr paramBlock,
                                Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`allowDuplicate`	`Boolean`	Are duplicate name and type OK?
→	`newName`	`RStringPtr`	New record name
→	`newType`	`RStringPtr`	New record type

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record whose name or type you want to change.You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.

`allowDuplicate`
A Boolean value that specifies whether you want to change the name and type even if this change results in a duplicate name and type. If you set this field to `true`, the function does not check the dNode for a duplicate name and type. It simply locates the record and executes the change.

`newName` A pointer to a buffer that contains the new name for the record. You provide this buffer.

`newType` A pointer to a buffer that contains the new type for the record. You provide this buffer.

DESCRIPTION

If `allowDuplicate` is set to `false`, the `DirSetNameAndType` function returns the `kOCENoDupAllowed` result code if it finds another record with the same name and the same type as the new name and type that you specified.

To change the record name without changing the type, set the value of the new type to the current type. To change the record type without changing the name, set the value of the new name to the current name.

If either the `newName` or `newType` field is set to `nil`, the function returns the `kOCEParamErr` result code.

SPECIAL CONSIDERATIONS

If you set the `allowDuplicate` field to `false`, the function will not set the new name and type if a record with the same name and type already exists. However, this does not guarantee that a duplicate record will not be created by a requester who sets the `allowDuplicate` field to `true`. The prohibition on duplicates applies only at the time you call this function; it does not guarantee that the record name and record type will be unique at a later time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0115

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEWriteAccessDenied`	–1541	Identity lacks write access privileges
`kOCEReadAccessDenied`	–1540	Identity lacks read access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can’t find specified dNode
`kOCEBadRecordID`	–1617	Record name or record type doesn’t match creation ID
`kOCENoSuchRecord`	–1618	Can’t find specified record
`kOCEOperationNotSupported`	–1626	Specified catalog does not support this operation
`kOCENoDupAllowed`	–1641	Duplicate name and type

SEE ALSO

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

DirAddPseudonym

The `DirAddPseudonym` function adds an alternative name and type for a record that you specify.

```
pascal OSErr DirAddPseudonym (DirParamBlockPtr paramBlock,
                              Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`pseudonymName`	`RStringPtr`	Alternative name
→	`pseudonymType`	`RStringPtr`	Alternative type
→	`allowDuplicate`	`Boolean`	Are duplicate name and type OK?

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record for which you want to add a pseudonym. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.

`pseudonymName` A pointer to the alternative name that you want to add.

`pseudonymType` A pointer to the alternative type that you want to add.

`allowDuplicate`
A Boolean value that indicates whether the function will add a name and type if another record, alias, or pseudonym with the same name and type already exists in the dNode. Set this field to `true` if you want the function to add the new pseudonym without checking for a duplicate name and type. If you set the `allowDuplicate` field to `false`, the function checks the name and type fields of all records, aliases, and pseudonyms in the dNode and returns the `kOCENoDupAllowed` result code if it finds a duplicate.

DESCRIPTION

You call the `DirAddPseudonym` function when you want to add an alternative name and type for a record. You can discover all of the existing pseudonyms for a record by calling the `DirEnumeratePseudonymGet` and `DirEnumeratePseudonymParse` functions.

Pseudonyms are automatically deleted when the target record is deleted.

You must specify both a name and a type. If either the `pseudonymName` or `pseudonymType` field is set to `nil`, the function returns the `kOCEParamErr` result code.

SPECIAL CONSIDERATIONS

If you set the `allowDuplicate` field to `false`, the function will not add the pseudonym if a pseudonym with the same name and type already exists. However, this does not guarantee that a duplicate pseudonym will not be created by a requester who sets the `allowDuplicate` field to `true`. The prohibition on duplicates applies only at the time you call this function; it does not guarantee that the pseudonym name and pseudonym type will be unique at a later time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$010F

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEWriteAccessDenied`	–1541	Identity lacks write access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCEBadRecordID`	–1617	Record name or record type doesn't match creation ID
`kOCENoSuchRecord`	–1618	Can't find specified record
`kOCEStreamCreationErr`	–1625	Error in connection to server
`kOCENoDupAllowed`	–1641	Duplicate name and type

SEE ALSO

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `DirEnumeratePseudonymGet` and `DirEnumeratePseudonymParse` functions are described on page 8-101 and page 8-104 respectively.

You can use the `DirEnumerateGet` and `DirEnumerateParse` functions, described on page 8-57 and page 8-62, respectively, to enumerate all of the pseudonyms that exist in a dNode.

To remove a pseudonym that you have added, use the `DirDeletePseudonym` function, described next.

DirDeletePseudonym

The `DirDeletePseudonym` function deletes an alternative name and type of a record that you specify.

```
pascal OSErr DirDeletePseudonym (DirParamBlockPtr paramBlock,
                                 Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`pseudonymName`	`RStringPtr`	Alternative name
→	`pseudonymType`	`RStringPtr`	Alternative type

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record whose alternative name and type you want to delete. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.

`pseudonymName` A pointer to the alternative name that you want to delete.

`pseudonymType` A pointer to the alternative type that you want to delete.

DESCRIPTION

If you no longer want to refer to a record by an alternate name or type, you can call this function to delete the name and the type.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0110

RESULT CODES

noErr	0	No error
kOCEParamErr	–50	Invalid parameter
kOCEWriteAccessDenied	–1541	Identity lacks write access privileges
kOCETargetDirectoryInaccessible	–1613	Target catalog is not currently available
kOCEBadRecordID	–1617	Record name or record type doesn't match creation ID
kOCENoSuchRecord	–1618	Can't find specified record
kOCENoSuchPseudonym	–1620	Can't find specified pseudonym
kOCEStreamCreationErr	–1625	Error in creating connection to server

SEE ALSO

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

To add a pseudonym to a record, use the `DirAddPseudonym` function, described on page 8-98.

DirEnumeratePseudonymGet

The `DirEnumeratePseudonymGet` function returns information about the pseudonyms for a record that you specify.

```
pascal OSErr DirEnumeratePseudonymGet
                                    (DirParamBlockPtr paramBlock,
                                    Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	serverHint	AddrBlock	AppleTalk address of the PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
→	aRecord	RecordIDPtr	Target record
→	startingName	RStringPtr	Name to start enumeration from
→	startingType	RStringPtr	Type to start enumeration from
→	includeStartingPoint	Boolean	Begin enumeration with starting point?
↔	getBuffer	Ptr	Your buffer
→	getBufferSize	unsigned long	Size of your buffer

See "The Parameter Block Header" on page 8-32 for descriptions of the ioCompletion, ioResult, serverHint, dsRefNum, identity, and clientData fields.

Field descriptions

aRecord — A pointer to a record ID that identifies the record for which you want to obtain pseudonyms. You must provide the record location information unless the record is in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.

startingName — A pointer to the alternative name from which you want the function to begin the enumeration. Set this field to nil to start with the first alternative name for the record. If the DirEnumeratePseudonymGet function completes with the kOCEMoreData result code, you can continue the enumeration by setting this field to the value of the name field in the last recordID parameter passed to your callback routine from the DirEnumeratePseudonymParse function. You must coordinate the value you provide in this field with the value you provide in the startingType field; that is, both values are required, and both must belong to the same pseudonym.

startingType — A pointer to the alternative type from which you want the function to begin the enumeration. Set this field to nil to start with the first alternative type for the record. If the DirEnumeratePseudonymGet function completes with the kOCEMoreData result code, you can continue the enumeration by setting this field to the value of the type field in the last recordID parameter passed to your callback routine from the DirEnumeratePseudonymParse function. You must coordinate the value you provide in this field with the value you provide in the startingName field; that is, both values are required, and both must belong to the same pseudonym.

includeStartingPoint
: A Boolean value that tells the function how to interpret the startingName and startingType fields. Set this field to true if you want the DirEnumeratePseudonymGet function to return information about pseudonyms beginning with the one specified by the startingName and startingType fields. If you set this field to false, the function returns information starting with the pseudonym after the one specified by the startingName and startingType fields.

getBuffer
: A pointer to the buffer in which the function stores the list of pseudonyms that you requested. You provide this buffer.

getBufferSize
: The number of bytes in the buffer.

DESCRIPTION

You call the DirEnumeratePseudonymGet function to obtain a list of the pseudonyms for a record that you specify.

If the buffer you provide is not large enough to contain all of the information you requested, the DirEnumeratePseudonymGet function returns the kOCEMoreData result code.

When the function completes with either the noErr or kOCEMoreData result codes, you use a pointer to your buffer as input to the DirEnumeratePseudonymParse function, which extracts the pseudonyms from the buffer.

If the DirEnumeratePseudonymGet function returns the kOCEMoreData result code, you can request additional information by calling the DirEnumeratePseudonymGet function again, after calling the DirEnumeratePseudonymParse function. As the values of the startingName and startingType fields, use the values of the name and type fields in the last recordID parameter passed to your callback routine from the DirEnumeratePseudonymParse function. The DirEnumeratePseudonymGet function will continue the enumeration starting with the next record as determined by the value of the includeStartingPoint field.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0113

RESULT CODES

noErr	0	No error
kOCEParamErr	–50	Invalid parameter
kOCEReadAccessDenied	–1540	Identity lacks read access privileges
kOCETargetDirectoryInaccessible	–1613	Target catalog is not currently available
kOCENoSuchDNode	–1615	Can't find specified dNode
kOCENoSuchRecord	–1618	Can't find specified record
kOCEMoreData	–1623	More data available

SEE ALSO

The RecordID data structure is described in the chapter "AOCE Utilities" in this book.

The DirEnumeratePseudonymParse function is described next.

For an example of continuing the enumeration (using the DirEnumerateAttributeTypesGet function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" beginning on page 8-20.

DirEnumeratePseudonymParse

The DirEnumeratePseudonymParse function parses the data returned by the DirEnumeratePseudonymGet function and returns a pointer to each pseudonym by repeatedly calling your callback routine.

```
pascal OSErr DirEnumeratePseudonymParse
                                    (DirParamBlockPtr paramBlock,
                                    Boolean async);
```

paramBlock
: Pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	serverHint	AddrBlock	AppleTalk address of the PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
→	aRecord	RecordIDPtr	Target record
→	eachRecordID	ForEachRecordID	Your callback routine
→	getBuffer	Ptr	Your buffer
→	getBufferSize	unsigned long	Size of your buffer

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record for which you want to obtain pseudonyms. Use the same value that you provided to the `DirEnumeratePseudonymGet` function.

`eachRecordID` A pointer to your callback routine. The function declaration for this routine is described on page 8-151.

`getBuffer` A pointer to the buffer containing the information to parse. Use the same buffer that you provided to the `DirEnumeratePseudonymGet` function.

`getBufferSize` The number of bytes in the buffer. Use the same value that you provided to the `DirEnumeratePseudonymGet` function.

DESCRIPTION

You call the `DirEnumeratePseudonymParse` function to extract the pseudonyms placed in a buffer by the `DirEnumeratePseudonymGet` function. You must provide a callback routine that the `DirEnumeratePseudonymParse` function calls for each pseudonym it finds in the buffer.

The `DirEnumeratePseudonymParse` function completes when it has finished parsing the contents of your buffer or when your callback routine returns `true`. The function returns the `kOCEMoreData` result code if it reaches the end of the buffer and finds that the `DirEnumeratePseudonymGet` function did not return all the data requested. If you want to continue the enumeration, you can call the `DirEnumeratePseudonymGet` function again. In your next call to the `DirEnumeratePseudonymGet` function, set the `startingName` and `startingType` fields to the values of the `name` and `type` fields of the last `recordID` parameter passed to your callback routine from the `DirEnumeratePseudonymParse` function.

If your callback routine returns `true`, the `DirEnumeratePseudonymParse` function completes with the `noErr` result code.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0104

RESULT CODES

noErr	0	No error
kOCEParamErr	–50	Invalid parameter
kOCEMoreData	–1623	More data available

SEE ALSO

The function declaration for your callback routine is described on page 8-151.

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `DirEnumeratePseudonymGet` function is described on page 8-101.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesParse` function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" beginning on page 8-20.

DirAddAlias

The `DirAddAlias` function adds an alias record to a catalog.

```
pascal OSErr DirAddAlias (DirParamBlockPtr paramBlock,
                          Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	serverHint	AddrBlock	AppleTalk address of the PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
↔	aRecord	RecordIDPtr	Target record
→	allowDuplicate	Boolean	Is duplicate name and type OK?

See “The Parameter Block Header” on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a partially specified record ID for the alias record that you want to add. If you want to add an alias record to a personal catalog, you must provide the alias record’s name and type. If you want to add an alias record to an external catalog, you must specify everything in the record ID except the `cid` field. The function places the creation ID for the new alias record in the `cid` field. If the catalog does not support creation IDs, the function sets the `cid` field to `nil`.

`allowDuplicate`

A Boolean value specifying whether the function should create an alias if another record, alias, or pseudonym with the same name and type already exists. Set this field to `true` if you want the function to create the alias without checking the dNode for a duplicate name and type. If you set the `allowDuplicate` field to `false`, the function checks the name and type of all records, aliases, and pseudonyms in the dNode and returns the `kOCENoDupAllowed` result code if it finds a duplicate.

DESCRIPTION

This function works just like the `DirAddRecord` function in that it adds a record. It also marks the new record as an alias. Your application is responsible for storing the information you need to resolve the alias. You should add to the new alias record an attribute whose type is referenced by the attribute type index `kAliasAttrTypeNum` and whose value is a `DSSpec` structure that points to the record that this is an alias to.

You can enumerate aliases with the `DirEnumerateGet` and `DirEnumerateParse` functions. You can use the `DirDeleteRecord` function to remove an alias record that you added.

The catalog feature bit flag `kSupportsAliasMask` indicates whether a catalog supports the `DirAddAlias` function.

SPECIAL CONSIDERATIONS

If you set the `allowDuplicate` field to `false`, the function will not add the alias if a record, pseudonym, or alias with the same name and type already exists. However, this does not guarantee that a duplicate alias will not be created by a requester who sets the `allowDuplicate` field to `true`. The prohibition on duplicates applies only at the time you call this function; it does not guarantee that the name and type will be unique at a later time.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$011C

RESULT CODES

noErr	0	No error
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	Same name and type already exists

SEE ALSO

The RecordID data structure is described in the chapter "AOCE Utilities" in this book.

The DirDeleteRecord function is described on page 8-91.

For a description of catalog feature flags, see "Feature Flag Bit Array" beginning on page 8-28.

The DirEnumerateGet function is described on page 8-57 and the DirEnumerateParse function is described on page 8-62.

The DirAddRecord function is described on page 8-89.

For more information on aliases and pseudonyms, see "Aliases and Pseudonyms" on page 8-7.

Managing Attribute Types and Values

The functions described in this section provide the following services:

- adding and deleting attribute values
- changing and verifying attribute values
- searching for an occurrence of specific data in an attribute value
- reading the attribute values in a record or records
- deleting an attribute type
- listing the attribute types in a record

DirAddAttributeValue

The `DirAddAttributeValue` function adds an attribute value to an existing record.

```
pascal OSErr DirAddAttributeValue (DirParamBlockPtr paramBlock,
                                   Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
↔	`attr`	`AttributePtr`	Attribute structure

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record to which you want to add an attribute value. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.

`attr` A pointer to an `Attribute` data structure. You must completely specify the `type` and `value` substructures of the `Attribute` data structure. The function returns the attribute creation ID.

DESCRIPTION

You call the `DirAddAttributeValue` function to add an attribute value to a record that you specify. PowerShare and personal catalogs do not check for already existing attributes having the same type and value; they simply add the attribute you specify. Therefore, you may add duplicate attribute values to PowerShare and personal catalog records. The Catalog Manager assigns them unique attribute creation IDs.

If the attribute type that you specify does not already exist within the record, the function first adds the new attribute type and then adds the value.

For PowerShare and personal catalogs, you can specify an attribute value up to `kAttrValueMaxBytes` bytes in length. If you specify an attribute value that is larger than `kAttrValueMaxBytes` bytes, the function returns the `kOCEAttributeValueTooBig` result code. The maximum size for an attribute value stored in an external catalog is undefined.

SPECIAL CONSIDERATIONS

Note that there is no function in the Catalog Manager API that explicitly adds an attribute type. To add a new attribute type to a record, begin by setting all fields of the `value` substructure to 0 or `nil` and the `type` substructure to the attribute type that you want to add. Then call the `DirAddAttributeValue` function. If that attribute type already exists within the record, the function returns without an error result code.

Although the Catalog Manager imposes no restrictions on the number of attribute values of a particular attribute type that you can add to a record, the Finder has limited ability to display multivalued attribute types. See the chapter "AOCE Templates" in this book for more information.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$010B

RESULT CODES

`noErr`	0	No error
`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
`kOCENoSuchRecord`	–1618	Can't find specified record
`kOCEAttributeValueTooBig`	–1621	Attribute value larger than `kAttrValueMaxBytes` bytes

SEE ALSO

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `Attribute` data structure is also described in the chapter "AOCE Utilities."

You can delete an attribute value with the `DirDeleteAttributeValue` function, described next.

You can delete an attribute type within a record with the `DirDeleteAttributeType` function, described on page 8-126.

DirDeleteAttributeValue

The `DirDeleteAttributeValue` function deletes an attribute value from a record that you specify.

```
pascal OSErr DirDeleteAttributeValue (DirParamBlockPtr paramBlock,
                                      Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`attr`	`AttributePtr`	Target attribute value

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record in which the target attribute value is located. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.

`attr` A pointer to an `Attribute` data structure. If you want to delete an attribute value from a catalog that supports attribute creation IDs, you identify the attribute value to be deleted by specifying the attribute creation ID and attribute type. To delete an attribute value from a catalog that does not support attribute creation IDs, specify the attribute type and attribute value.

DESCRIPTION

You call the `DirDeleteAttributeValue` function to delete an attribute value from a record. Deleting the last attribute value of a given attribute type does not delete the attribute type.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$010C

RESULT CODES

noErr	0	No error
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
kOCENoSuchRecord	–1618	Can't find specified record
kOCENoSuchAttributeValue	–1619	Can't find specified attribute value

SEE ALSO

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `Attribute` data structure is also described in the chapter "AOCE Utilities."

You can add an attribute value to a record with the `DirAddAttributeValue` function, described on page 8-109.

You can delete an attribute type with the `DirDeleteAttributeType` function, described on page 8-126.

DirChangeAttributeValue

The `DirChangeAttributeValue` function changes an attribute value that you specify.

```
pascal OSErr DirChangeAttributeValue (DirParamBlockPtr paramBlock,
                                      Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	serverHint	AddrBlock	AppleTalk address of the PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
→	aRecord	RecordIDPtr	Target record
→	currentAttr	AttributePtr	Existing attribute value
→	newAttr	AttributePtr	New attribute value

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record containing the attribute value that you want to change. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.

`currentAttr` A pointer to the `Attribute` data structure that specifies the attribute value that you want to change. For catalogs that support attribute creation IDs, you identify the attribute value to change by providing its attribute creation ID and attribute type; otherwise, you need to provide the attribute value and attribute type.

`newAttr` A pointer to an `Attribute` data structure that contains the new attribute value. In the `value` field, you provide the new attribute value and its length in bytes. You must also provide a type in the `type` field or a `ParamErr` is returned.

DESCRIPTION

You call the `DirChangeAttributeValue` function to change an attribute value without changing its associated attribute creation ID. If you want to assign a new attribute creation ID to the attribute value, use the `DirDeleteAttributeValue` function to delete the old value and the `DirAddAttributeValue` function to add the new value.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$010D

RESULT CODES

noErr	0	No error
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
kOCENoSuchRecord	–1618	Can't find specified record
kOCENoSuchAttributeValue	–1619	Can't find specified attribute value

SEE ALSO

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `Attribute` data structure is also described in the chapter "AOCE Utilities."

DirVerifyAttributeValue

The `DirVerifyAttributeValue` function indicates whether the specified attribute value exists in the record.

```
pascal OSErr DirVerifyAttributeValue (DirParamBlockPtr paramBlock,
                                      Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	serverHint	AddrBlock	AppleTalk address of the PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
→	aRecord	RecordIDPtr	Target record
→	attr	AttributePtr	Target attribute value

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord`	A pointer to a record ID that identifies the record in which the target attribute value resides. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.
`attr`	A pointer to an `Attribute` data structure. You must specify the `type` and `value` fields. You may also provide the attribute creation ID if you know it. Otherwise, set the `cid` field of this structure to 0.

DESCRIPTION

If you provide the attribute creation ID, the `DirVerifyAttributeValue` function verifies that an attribute value having the specified attribute type, attribute creation ID, and actual attribute value exists in the record you specify.

SPECIAL CONSIDERATIONS

If you set the attribute creation ID to 0, the function verifies that an attribute value having the specified actual attribute value and attribute type exists in the record you specify and returns its attribute creation ID in the `cid` field of your `Attribute` data structure. Note that duplicate attribute values may exist in the same record. The attribute creation ID that the function returns may belong to any of the duplicate attribute values.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$010E

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEReadAccessDenied`	–1540	Identity lacks read access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can’t find specified dNode
`kOCENoSuchRecord`	–1618	Can’t find specified record
`kOCENoSuchAttributeValue`	–1619	Can’t find specified attribute value

SEE ALSO

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

The `Attribute` data structure is also described in the chapter “AOCE Utilities.”

DirFindValue

The `DirFindValue` function searches the records in a dNode that you specify for an occurrence of an attribute value.

```
pascal OSErr DirFindValue (DirParamBlockPtr paramBlock,
                           Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRLI`	`PackedRLIPtr`	Target dNode
→	`aRecord`	`LocalRecordIDPtr`	Target record
→	`attrType`	`AttributeTypePtr`	Target attribute type
→	`startingRecord`	`LocalRecordIDPtr`	Record to start search from
→	`startingAttribute`	`AttributePtr`	Attribute value to start search from
↔	`recordFound`	`LocalRecordIDPtr`	Record containing matching data
←	`attributeFound`	`Attribute`	Attribute containing matching data
→	`matchSize`	`unsigned long`	Length of data to match
→	`matchingData`	`Ptr`	Data to match
→	`sortDirection`	`DirSortDirection`	Search forward or backward?

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRLI` A pointer to the dNode in which you want to search for an attribute value. The function ignores this field when you provide a nonzero value in the `dsRefNum` field to specify a personal catalog.

`aRecord` A pointer to a local record ID. If you set this field to a nonzero value, the search for matching data is restricted to the record you specify here. Set this field to `nil` if you do not want to restrict your search to a single record.

`attrType` A pointer to an attribute type. If you set this field to a nonzero value, the search for matching data is restricted to the attribute type that you specify here. Set this field to `nil` if you do not want to restrict your search to a single attribute type.

`startingRecord`
A pointer to a local record ID. Set this field to `nil` if you wish to start the search with the first record in the dNode. If you have already called the `DirFindValue` function and found an occurrence of matching data, you can set this field to the value of the `recordFound` field to search for the next occurrence.

`startingAttribute`
A pointer to an `Attribute` data structure. Set this field to `nil` if you wish to start the search with the first attribute value in the first record to be searched. If you have already called the `DirFindValue` function, you can set this field to the value of the `attributeFound` field to search for the next occurrence.

`recordFound` A pointer to the local record ID that identifies the record in which the `DirFindValue` function found a matching attribute value. You can set the `startingRecord` field to this field in a subsequent call to the `DirFindValue` function.

`attributeFound`
An `Attribute` data structure that specifies the attribute value within which the function found matching data. You can set the `startingAttribute` field to the address of this structure in a subsequent call to the `DirFindValue` function. If you are searching a PowerShare or personal catalog, the function returns only the attribute creation ID in the `Attribute` data structure. If the catalog in which you are searching does not support attribute creation IDs, the function may return a complete attribute value. In that case, you must provide a buffer large enough to hold a maximum size attribute value as part of your `Attribute` data structure.

`matchSize` The number of bytes of data to be matched.

`matchingData` A pointer to a buffer that contains the data to be matched.

`sortDirection` A constant that specifies the search direction. Set this field to `kSortForwards` to have the function search in a forward direction through the dNode and the record for a match. Set this field to `kSortBackwards` to have the function search in a backward direction.

DESCRIPTION

The `DirFindValue` function examines up to the first 32 bytes of data in an attribute value to find a match when it is searching in a PowerShare or personal catalog. The match is to any string that begins with the search string.

The function returns the type and creation ID of the attribute value in which it finds a match. You may call `DirLookupGet` to obtain the complete attribute value.

IMPORTANT

Personal catalogs and the PowerShare catalog server do not support the `DirFindValue` function. An external catalog may or may not support this function, and it may match more or less than 32 characters. s

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0126

RESULT CODES

`noErr`	0	No error

SEE ALSO

The `DirLookupGet` function is described next.

The `PackedRLI` data structure is described in the chapter "AOCE Utilities" in this book.

You can create a `PackedRLI` data structure with the `OCEPackRLI` utility routine. It is described in the chapter "AOCE Utilities."

The `LocalRecordID` data structure is also described in the chapter "AOCE Utilities."

The `Attribute` data structure is also described in the chapter "AOCE Utilities."

DirLookupGet

The `DirLookupGet` function returns the attribute values of the attribute types that you specify for a list of records that you provide.

```
pascal OSErr DirLookupGet (DirParamBlockPtr paramBlock,
                           Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	serverHint	AddrBlock	Address of the PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
→	aRecordList	RecordIDPtr*	List of record IDs
→	attrTypeList	AttributeTypePtr*	List of attribute types
→	recordIDCount	unsigned long	Number of IDs in list
→	attrTypeCount	unsigned long	Number of types in list
→	includeStartingPoint	Boolean	Begin enumeration with starting point?
↔	getBuffer	Ptr	Your buffer
→	getBufferSize	unsigned long	Size of your buffer
→	startingRecordIndex	unsigned long	Record to start from
→	startingAttrTypeIndex	unsigned long	Attribute type to start from
→	startingAttribute	Attribute	Attribute value to start from

See "The Parameter Block Header" on page 8-32 for descriptions of the ioCompletion, ioResult, serverHint, dsRefNum, identity, and clientData fields.

Field descriptions

aRecordList A pointer to an array of pointers to record IDs. The record IDs represent the records in which you want to look up attribute values. You must provide record location information in each record ID unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the record creation ID; otherwise, you must provide the record name and type. The record IDs in your list should be unique.

For PowerShare catalogs, all of the records that you specify must reside in the same dNode. This is not necessarily true for external catalogs.

attrTypeList A pointer to an array of pointers to attribute types. The attribute types are those for which you want to look up attribute values. The attribute types in your list should be unique.

recordIDCount The number of elements in your array of pointers to record IDs.

attrTypeCount The number of elements in your array of pointers to attribute types.

includeStartingPoint
A Boolean value that determines how the DirLookupGet function interprets the startingRecordIndex, startingAttrTypeIndex, and startingAttribute fields. Set this field to true if you want the DirLookupGet function to return information from the record, attribute type, and attribute value specified by the starting fields. If you set this field to false, the function returns information starting with the record, attribute type, and attribute value immediately following the one specified by the starting fields.

`getBuffer` A pointer to the buffer in which the function stores the requested information. You provide this buffer.

`getBufferSize` The number of bytes in the buffer.

`startingRecordIndex`
An index into the array of pointers to record IDs. It represents the record at which the `DirLookupGet` function begins the lookup. To start at the first record specified by the array, set this value to 1. The value of the `startingRecordIndex` field must always be less than or equal to the value of the `recordIDCount` field.

`startingAttrTypeIndex`
An index into the array of pointers to attribute types. It represents the attribute type at which the function begins the lookup. To start at the first attribute type specified by the array, set this value to 1. The value of the `startingAttrTypeIndex` field must always be less than or equal to the value of the `attrTypeCount` field.

`startingAttribute`
An `Attribute` data structure that specifies the attribute value at which the `DirLookupGet` function begins the lookup. If the catalog in which you are requesting the lookup supports creation IDs, the attribute creation ID is a sufficient specification. If you set the attribute creation ID to 0, the function begins the search from the first attribute value of the type specified by the `startingAttrTypeIndex` field. If you specify a nonzero attribute creation ID and the function does not find an attribute value with a matching creation ID, `DirLookupGet` terminates with a `kOCENoSuchAttributeValue` result code. You should not call `DirLookupParse` after getting this error.

If the catalog in which you are requesting the lookup does not support attribute creation IDs, you must specify the entire structure. Set every field in the structure to 0 if you want the function to begin the search from the first attribute value of the type specified by the `startingAttrTypeIndex` field.

DESCRIPTION

You call the `DirLookupGet` function to obtain the attribute values of particular attribute types in records that you specify. You must specify a record, an attribute type, and an attribute value at which you want the function to start the lookup.

The `DirLookupGet` function places the requested attribute values in your buffer. It also provides the local record IDs identifying the records in which the attribute values are located. Last, it provides the attribute type and associated access control information that apply to each attribute value. If the buffer you provide is not large enough to contain all of the information requested, the `DirLookupGet` function returns the `kOCEMoreData` result code.

When the `DirLookupGet` function completes with either the `noErr` or `kOCEMoreData` result codes, call the `DirLookupParse` function to extract the attribute information from your buffer. You can pass `DirLookupParse` the same parameter that you passed to `DirLookupGet`.

If the `DirLookupGet` function completes with the `kOCEMoreData` result code, you may wish to continue the lookup. When the `DirLookupParse` function completes, it returns values in the `lastRecordIndex`, `lastAttributeIndex`, and `lastAttribute` fields. You may use these as the values of the `startingRecordIndex`, `startingAttrTypeIndex`, and `startingAttribute` fields on a subsequent call to the `DirLookupGet` function. Therefore, you can simply pass the same parameter block to the `DirLookupGet` function as you passed to the `DirLookupParse` function and call the `DirLookupGet` function to continue retrieving information from the point at which it stopped during its previous invocation.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0117

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can't find specified dNode
`kOCENoSuchRecord`	–1618	Can't find specified record
`kOCENoSuchAttributeValue`	–1619	Can't find specified attribute value
`kOCEMoreData`	–1623	More data available
`kOCEBadStartingRecord`	–1638	Starting record index out of range
`kOCEBadStartingAttribute`	–1639	Starting attribute index out of range
`kOCERLIsDontMatch`	–1645	RLIs of different records in the record list are not the same

SEE ALSO

The `DirLookupParse` function is described next.

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `Attribute` data structure is also described in the chapter "AOCE Utilities."

For an example of continuing the enumeration using the `DirLookupGet` function when the buffer is too small to hold all the information you requested, see "Getting Attribute Value Information" beginning on page 8-16.

DirLookupParse

The `DirLookupParse` function parses the data returned by the `DirLookupGet` function and returns each attribute value to your application by repeatedly calling your callback routine. It also returns information about record IDs and attribute types when you specify callback routines for these purposes.

```
pascal OSErr DirLookupParse (DirParamBlockPtr paramBlock,
                             Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecordList`	`RecordIDPtr*`	List of record IDs
→	`attrTypeList`	`AttributeTypePtr*`	List of attribute types
→	`eachRecordID`	`ForEachLookupRecordID`	Your callback routine for record information
→	`eachAttrType`	`ForEachAttrTypeLookup`	Your callback routine for attribute type information
→	`eachAttrValue`	`ForEachAttrValue`	Your callback routine for attribute values
→	`recordIDCount`	`unsigned long`	Number of IDs in list
→	`attrTypeCount`	`unsigned long`	Number of types in list
→	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer
←	`lastRecordIndex`	`unsigned long`	Last record ID retrieved
←	`lastAttributeIndex`	`unsigned long`	Last attribute type retrieved
←	`lastAttribute`	`Attribute`	Last attribute value retrieved
←	`attrSize`	`unsigned long`	Length of the attribute value that was too big to fit

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecordList` A pointer to an array of pointers to record IDs. Use the same value that you provided to the corresponding `DirLookupGet` function.

`attrTypeList` A pointer to an array of pointers to attribute types.Use the same value that you provided to the corresponding `DirLookupGet` function.

`eachRecordID` A pointer to your callback routine for record information. The function declaration for this routine is described on page 8-154. The `DirLookupParse` function calls this routine for each record ID in the buffer. If you are looking up attribute values in a single record, you may not want to provide this callback routine. Set this field to `nil` if you do not want to specify this callback routine.

`eachAttrType` A pointer to your callback routine for attribute type information. The `DirLookupParse` function passes your callback routine a pointer to an attribute type and the access control mask that applies to the requester for that attribute type. The attribute type always belongs in the record identified in the most recent call to your `MyForEachRecordID` routine. You may set this field to `nil` if you do not want to specify this callback routine. If you are looking up only one attribute type, or you prefer to read the type from the `Attribute` data structure that the `DirLookupParse` function passes to the `MyForEachAttrValue` routine, you may not want to provide this callback routine. However, it is recommended that you supply this callback routine to get the access control information for a given attribute type. Access controls may prohibit you from reading an attribute value. In that case, the `DirLookupParse` function does not call your `MyForEachAttrValue` callback routine even though the attribute value exists. If you do not supply the `MyForEachAttrType` callback routine, you have no way of knowing whether attribute values of the requested type exist for which you are denied read access. The function declaration for this routine is described on page 8-155.

`eachAttrValue` A pointer to your callback routine for attribute values (`MyForEachAttrValue`). You must provide this callback routine. The function declaration for this routine is described on page 8-156.

`recordIDCount` The number of elements in the array of pointers to record IDs. Use the same value that you provided to the corresponding `DirLookupGet` function.

`attrTypeCount` The number of elements in the array of pointers to attribute types. Use the same value that you provided to the corresponding `DirLookupGet` function.

`getBuffer` A pointer to the buffer containing the information to parse. Use the same buffer that you provided to the `DirLookupGet` function.

`getBufferSize` The number of bytes in the buffer.

`lastRecordIndex` The index value of the last record that the `DirLookupParse` function retrieved from your buffer.

`lastAttributeIndex`
: The index value of the last attribute type that the `DirLookupParse` function retrieved from your buffer.

`lastAttribute`
: An `Attribute` data structure that specifies the last attribute value that the `DirLookupParse` function retrieved from your buffer.

`attrSize`
: The length of an attribute value that is too large to fit in the buffer. If your buffer is too small to hold an attribute value, the `DirLookupParse` function sets this field to the length of the attribute value that cannot fit in your buffer and returns the `kOCEMoreAttrValue` result code. In this case, the value in this field represents the minimum size of a buffer capable of holding the attribute value. You must increase the size of your buffer to at least the minimum size and call the `DirLookupGet` function again. When the function does not return the `kOCEMoreAttrValue` result code, this field is undefined.

DESCRIPTION

You call the `DirLookupParse` function to extract the information on attribute values, attribute types, and records placed in a buffer by the `DirLookupGet` function.

When you provide callback routines for record and attribute type information, the `DirLookupGet` function returns the record IDs and attribute types in the same order as you provided in your list of record IDs and attribute types.

You should provide a callback routine for record information if you request information on more than one record. All of the attribute values that the `DirLookupParse` function passes to your callback routine for attribute values (`MyForEachAttrValue`) belong to the record identified in the most recent call to your callback routine for record information (`MyForEachRecordID`). If you do not provide this routine, you cannot determine the record to which an attribute type or value belongs. In addition, a callback routine for record information allows you to distinguish between the case where a record exists but an attribute type does not exist and the case where a record does not exist.

Although it is optional, you should provide a callback routine for attribute types (`MyForEachAttrType`) because it receives access control information about every attribute type in your buffer. If you do not have read access to an attribute type, the `DirLookupParse` function does not call your callback routine for the corresponding attribute values even though those attribute values are present in your buffer. By providing a callback routine for attribute types, you can detect the presence of attribute values for which you do not have read access.

The `DirLookupParse` function returns the `kOCEMoreData` result code if it reaches the end of the buffer and finds that the `DirLookupGet` function did not return all the data requested. In this case, you can call the `DirLookupGet` function again to retrieve more data. The `DirLookupParse` function sets the values of the `lastRecordIndex`, `lastAttributeIndex`, and `lastAttribute` fields in the parameter block to indicate the last items it retrieved from your buffer. These fields correspond to the `startingRecordIndex`, `startingAttrTypeIndex`, and `startingAttribute` fields in the `DirLookupGet` function's parameter block. You can use the same

parameter block when you call the `DirLookupGet` function again, and it will continue retrieving data at the point where it stopped the first time you called it.

If the `DirLookupParse` function returns the `kOCEMoreAttrValue` result code, you must increase the size of your buffer before calling the `DirLookupGet` function again. There are two conditions in which an attribute value may not fit in your buffer. The first occurs when your buffer already contains some data and the remaining space is insufficient to store the next `Attribute` data structure. In this case, `DirLookupGet` returns the `kOCEMoreData` result code. Such an attribute value will be stored in your buffer the next time you call the `DirLookupGet` function. The second condition occurs when the size of an attribute value exceeds the size of your buffer. Such an attribute value will not fit even when your buffer is empty. In this second case, the `DirLookupGet` function completes with the `kOCEMoreData` result code; the corresponding `DirLookupParse` function call stores the length of the oversized attribute value in the `attrSize` field and returns the `kOCEMoreAttrValue` result code. Before calling the `DirLookupGet` function again, you must increase the size of your buffer to accommodate the oversized attribute value.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0102

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEReadAccessDenied`	–1540	Identity lacks read access privileges
`kOCEMoreData`	–1623	More data available
`kOCEMoreAttrValue`	–1640	Buffer too small for a single attribute value

SEE ALSO

The `DirLookupGet` function is described on page 8-118.

The function declaration for your callback routine that processes record information is described on page 8-154.

The function declaration for your callback routine that processes attribute type information is described on page 8-155.

The function declaration for your callback routine that processes attribute value information is described on page 8-156.

For an example of continuing the enumeration using the `DirLookupParse` function when the buffer is too small to hold all the information you requested, see “Getting Attribute Value Information” beginning on page 8-16.

DirDeleteAttributeType

The `DirDeleteAttributeType` function deletes an attribute type and its associated values from a particular record.

```
pascal OSErr DirDeleteAttributeType (DirParamBlockPtr paramBlock,
                                     Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`attrType`	`AttributeTypePtr`	Target attribute type

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record in which the target attribute type is located. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.

`attrType` A pointer to the attribute type that you want to delete.

DESCRIPTION

You call the `DirDeleteAttributeType` function to delete an existing attribute type. If any attribute values exist for that type, the function first deletes the values and then deletes the attribute type. If you do not have access privileges to delete the attribute type, the function returns the `kOCEWriteAccessDenied` result code.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0130

RESULT CODES

noErr	0	No error
kOCEWriteAccessDenied	–1541	Identity lacks write access privileges
kOCETargetDirectoryInaccessible	–1613	Target catalog is not currently available
kOCENoSuchRecord	–1618	Can't find specified record
kOCENoSuchAttributeType	–1642	Can't find specified attribute type

SEE ALSO

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `AttributeType` data structure is also described in the chapter "AOCE Utilities."

Access controls are discussed in the section "Getting Access Controls" beginning on page 8-11.

Records and attributes are described in the section "Catalog Records and Attributes" beginning on page 8-6.

DirEnumerateAttributeTypesGet

The `DirEnumerateAttributeTypesGet` function returns information about the attribute types in a record.

```
pascal OSErr DirEnumerateAttributeTypesGet
                                    (DirParamBlockPtr paramBlock,
                                    Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	serverHint	AddrBlock	AppleTalk address of PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	clientData	long	You define this field
→	aRecord	RecordIDPtr	Target record
→	startingAttrType	AttributeTypePtr	Attribute type to start from
→	includeStartingPoint	Boolean	Begin enumeration with starting point?
↔	getBuffer	Ptr	Your buffer
→	getBufferSize	unsigned long	Size of your buffer

See "The Parameter Block Header" on page 8-32 for descriptions of the ioCompletion, ioResult, serverHint, dsRefNum, identity, and clientData fields.

Field descriptions

aRecord — A pointer to a record ID that identifies the record about which you are requesting attribute type information. You must provide the record location information unless the record exists in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.

startingAttrType — A pointer to the attribute type within the record at which you want the DirEnumerateAttributeTypesGet function to begin the enumeration. Set this field to nil to start with the first attribute type in the record. If the DirEnumerateAttributeTypesGet function completes with the kOCEMoreData result code, you can continue the enumeration by setting this field to the value of the last attrType parameter passed to your callback routine by the DirEnumerateAttributeTypesParse function.

includeStartingPoint — A Boolean value that determines how this function interprets the startingAttrType field. Set this field to true if you want the DirEnumerateAttributeTypesGet function to return information about attribute types beginning with the one you specify in the startingAttrType field. If you set this field to false, the function returns information starting with the attribute type immediately after the one specified by the startingAttrType field.

getBuffer — A pointer to the buffer in which the function stores the attribute types it found in the specified record. You provide this buffer.

getBufferSize — The number of bytes in the buffer.

DESCRIPTION

You call the `DirEnumerateAttributeTypesGet` function to obtain a list of the attribute types that are present in a particular record.

If your buffer is not large enough to contain all of the information requested, the `DirEnumerateAttributeTypesGet` function returns the `kOCEMoreData` result code.

When the `DirEnumerateAttributeTypesGet` function completes with either the `noErr` or `kOCEMoreData` result codes, you provide a pointer to your buffer to the `DirEnumerateAttributeTypesParse` function, which extracts the attribute type information from the buffer.

If your buffer is too small to hold all of the requested information, you can get additional information by calling the `DirEnumerateAttributeTypesGet` function again. As the value of the `startingAttrType` field, use the value of the last `attrType` parameter passed to your callback routine by the `DirEnumerateAttributeTypesParse` function. The `DirEnumerateAttributeTypesGet` function will continue the enumeration starting with the next attribute type as determined by the value of the `includeStartingPoint` field.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0112

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEReadAccessDenied`	–1540	Identity lacks read access privileges
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchDNode`	–1615	Can’t find specified dNode
`kOCENoSuchRecord`	–1618	Can’t find specified record
`kOCEMoreData`	–1623	More data available
`kOCEDirectoryNotFoundErr`	–1630	Can’t find catalog

SEE ALSO

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

The `AttributeType` data structure is also described in the chapter “AOCE Utilities.”

The `DirEnumerateAttributeTypesParse` function is described next.

For an example of continuing the enumeration using the `DirEnumerateAttributeTypesGet` function when the buffer is too small to hold all the information you requested, see “Getting Attribute Type Information” beginning on page 8-20.

DirEnumerateAttributeTypesParse

The `DirEnumerateAttributeTypesParse` function parses the data returned by the `DirEnumerateAttributeTypesGet` function and returns each attribute type to your application by repeatedly calling your callback routine.

```
pascal OSErr DirEnumerateAttributeTypesParse
                                    (DirParamBlockPtr paramBlock,
                                    Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`eachAttrType`	`ForEachAttrType`	Your callback routine
→	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record about which you want to obtain attribute type information. Use the same value that you provided to the corresponding `DirEnumerateAttributeTypesGet` function.

`eachAttrType` A pointer to your callback routine. The function declaration for this routine is described on page 8-152.

`getBuffer` A pointer to the buffer containing the attribute types to parse. Use the same buffer that you provided to the corresponding `DirEnumerateAttributeTypesGet` function.

`getBufferSize` The number of bytes in the buffer. Use the same value that you provided to the corresponding `DirEnumerateAttributeTypesGet` function.

DESCRIPTION

You call the `DirEnumerateAttributeTypesParse` function to extract the attribute types placed in a buffer by the `DirEnumerateAttributeTypesGet` function. You must provide a callback routine that the `DirEnumerateAttributeTypesParse` function calls for each attribute type that it finds in the buffer.

The `DirEnumerateAttributeTypesParse` function completes when it has finished parsing the contents of your buffer or when your callback routine returns `true`. The function returns the `kOCEMoreData` result code if it reaches the end of the buffer and finds that the `DirEnumerateAttributeTypesGet` function did not return all the data requested. If you want to continue the enumeration, you can call the `DirEnumerateAttributeTypesGet` function again. In your next call to the `DirEnumerateAttributeTypesGet` function, set `startingAttrType` to the value of the last `attrType` parameter passed to your callback routine by the `DirEnumerateAttributeTypesParse` function.

If your callback routine returns `true`, the `DirEnumerateAttributeTypesParse` function completes with the `noErr` result code.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0103

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

The function declaration for your callback routine is described on page 8-152.

The `DirEnumerateAttributeTypesGet` function is described on page 8-127.

For an example of continuing the enumeration using the `DirEnumerateAttributeTypesParse` function when the buffer is too small to hold all the information you requested, see “Getting Attribute Type Information” beginning on page 8-20.

Reading Access Controls for dNodes, Records, and Attribute Types

The functions in this section identify requestor categories and access controls. There are five categories of requestors. You use the `OCEGetAccessControlDSSpec` function to read category masks that identify the type of requestor about which you want information. See the section "Getting Access Controls" beginning on page 8-11 for more information on access controls and requestor categories.

You can use the other functions described in this section to read the access control masks for a dNode, a record, or an attribute type.

OCEGetAccessControlDSSpec

The `OCEGetAccessControlDSSpec` function returns a `DSSpec` that you can use to get access controls.

```
pascal DSSpec *OCEGetAccessControlDSSpec (const CategoryMask
                                          categoryBitMask);
```

`categoryBitMask`
A value indicating the type of `DSSpec` you want `OCEGetAccessControlDSSpec` to return.

DESCRIPTION

Given one of the `categoryBitMask` values, the `OCEGetAccessControlDSSpec` function returns to you a pointer to a `DSSpec` structure that corresponds to the particular `categoryBitMask` value. You can then use, in a `DirGet`*xxx*`AccessControlGet` function, the `DSSpec` that is returned to you. The `categoryBitMask` value identifies a type of requestor, such as owner or guest. For more information on how to use the `categoryBitMask` value to obtain a `DSSpec` structure, see "Types of Requesters" beginning on page 8-11.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`__OCEUtils`	$0345

RESULT CODES

`noErr`	0	No error

SEE ALSO

The `CategoryMask` data type is described in "Getting Access Controls" beginning on page 8-11.

The `DSSpec` data structure is described in the chapter "Utility Manager" in this book.

DirGetDNodeAccessControlGet

The `DirGetDNodeAccessControlGet` function returns access control information for a dNode that you specify.

```
pascal OSErr DirGetDNodeAccessControlGet
                                    (DirParamBlockPtr paramBlock,
                                    Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`pRLI`	`PackedRLIPtr`	Target dNode
→	`forCurrentUserOnly`	`Boolean`	Return only requester's access controls?
→	`startingPoint`	`DSSpec*`	Object to start from
→	`includeStartingPoint`	`Boolean`	Begin enumeration with starting point?
↔	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`pRLI` A pointer to packed record location information that specifies the dNode about which you want access control information. The function ignores this field when you specify a personal catalog in the `dsRefNum` field.

`forCurrentUserOnly` A Boolean value that indicates what access control information the function returns. Set this field to `true` if you want only access control information for the requester specified in the `identity` field. If you set this field to `false`, the function returns access control information for each object on the dNode's access control list.

`startingPoint` A pointer to the object on the access control list from which the function begins to retrieve information. Set this field to `nil` to start with the first object. If the function completes with the `kOCEMoreData` result code, you can set this field to the value that `DirGetDNodeAccessControlParse` last passed to the `dsObj` parameter of your callback routine. Then call `DirGetDNodeAccessControlGet` again to continue to obtain information. The function ignores this field if you set `forCurrentUserOnly` to `true`.

`includeStartingPoint` A Boolean value that determines how the function interprets the `startingPoint` field. Set this field to `true` if you want the function to return access control information beginning with the object specified by the `startingPoint` field. If you set this field to `false`, the function returns information starting with the object after the one specified by the `startingPoint` field. The function ignores this field if you set `forCurrentUserOnly` to `true`.

`getBuffer` A pointer to the buffer in which the function stores the access control information for the dNode you specify. You provide this buffer.

`getBufferSize` The size, in bytes, of your buffer.

DESCRIPTION

You call the `DirGetDNodeAccessControlGet` function to obtain access control information for a dNode that you specify. The information consists of catalog service specifications that identify the objects on the access control list and of the access control masks that apply to these objects. The mask specifies which access privileges the object possesses.

If the buffer you provide is not large enough to contain all of the information requested, the function returns the `kOCEMoreData` result code.

When the function completes with either the `noErr` or `kOCEMoreData` result codes, you use a pointer to your buffer as input to the `DirGetDNodeAccessControlParse` function, which extracts the information from the buffer.

If your buffer is too small to hold all of the requested information, you can get additional information by calling the `DirGetDNodeAccessControlGet` function again, after calling the `DirGetDNodeAccessControlParse` function. Set the `startingPoint` field to the value that `DirGetDNodeAccessControlParse` last passed to the `dsObj` parameter of your callback routine. The `DirGetDNodeAccessControlGet` function will continue to return information starting with the next entry on the dNode's access control list as determined by the value of the `includeStartingPoint` field.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$012A

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchdNode`	–1615	Can't find specified dNode
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `PackedRLI` data structure is described in the chapter "AOCE Utilities" in this book.

You can create a `PackedRLI` data structure with the `OCEPackRLI` utility routine. It is also described in the chapter "AOCE Utilities."

The catalog service specification, defined by the `DSSpec` data structure, is also described in the chapter "AOCE Utilities."

The `DirGetDNodeAccessControlParse` function is described next.

For information on the types of objects and the types of access controls specified in the access control mask, see "Getting Access Controls" beginning on page 8-11.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesGet` function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" beginning on page 8-20.

DirGetDNodeAccessControlParse

The `DirGetDNodeAccessControlParse` function parses the access control information returned by the `DirGetDNodeAccessControlGet` function and returns a pointer to each object and its access control mask by repeatedly calling your callback routine.

```
pascal OSErr DirGetDNodeAccessControlParse
                                    (DirParamBlockPtr paramBlock,
                                    Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`pRLI`	`PackedRLIPtr`	Target dNode
→	`eachObject`	`ForEachDNodeAccessControl`	Your callback routine
→	`forCurrentUserOnly`	`Boolean`	Return only requester's access info?
→	`startingPoint`	`DSSpec*`	Object to start from
→	`includeStartingPoint`	`Boolean`	Begin enumeration with starting point?
→	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`pRLI` A pointer to packed record location information that specifies the dNode about which you want access control information. Use the same value that you provided to the `DirGetDNodeAccessControlGet` function.

`eachObject` A pointer to your callback routine. The Catalog Manager passes your callback routine the value that you provide in the `clientData` field, a pointer to a catalog service specification that identifies the object to which the access control masks apply, the

active access control mask for the target dNode, and the default access control masks for newly created records and attribute types within that dNode. The function declaration for this routine is described on page 8-161.

`forCurrentUserOnly`
Use the same value that you provided to the `DirGetDNodeAccessControlGet` function.

`startingPoint` Use the same value that you provided to the `DirGetDNodeAccessControlGet` function.

`includeStartingPoint`
Use the same value that you provided to the `DirGetDNodeAccessControlGet` function.

`getBuffer` A pointer to the buffer containing the information to parse. Use the same buffer that you provided to the `DirGetDNodeAccessControlGet` function.

`getBufferSize` The size, in bytes, of your buffer. Use the same value that you provided to the `DirGetDNodeAccessControlGet` function.

DESCRIPTION

You call the `DirGetDNodeAccessControlParse` function to extract the access control information placed in your buffer by the `DirGetDNodeAccessControlGet` function. You must provide a callback routine that the `DirGetDNodeAccessControlParse` function calls for each entry it finds in the buffer.

The `DirGetDNodeAccessControlParse` function completes when it has finished parsing the contents of your buffer or when your callback routine returns `true`. The function returns the `kOCEMoreData` result code if it reaches the end of the buffer and finds that the `DirGetDNodeAccessControlGet` function did not return all the data requested. If you want to continue to obtain information, you can call the `DirGetDNodeAccessControlGet` function again. Set the value of the `startingPoint` field to the value that `DirGetDNodeAccessControlParse` last passed to the `dsObj` parameter of your callback routine.

If your callback routine returns `true`, the `DirGetDNodeAccessControlParse` function completes with the `noErr` result code.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$012F

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `PackedRLI` data structure is described in the chapter "AOCE Utilities" in this book.

The `DSSpec` data structure is also described in the chapter "AOCE Utilities."

The `DirGetDNodeAccessControlGet` function is described on page 8-133.

The function declaration for your callback routine is described on page 8-161.

For information on the types of access controls specified in the access control mask, see "Getting Access Controls" beginning on page 8-11.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesParse` function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" beginning on page 8-20.

DirGetRecordAccessControlGet

The `DirGetRecordAccessControlGet` function returns the access controls of a record that you specify.

```
pascal OSErr DirGetRecordAccessControlGet
                                    (DirParamBlockPtr paramBlock,
                                    Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`forCurrentUserOnly`	`Boolean`	Return only requester's access controls?
→	`startingPoint`	`DSSpec*`	Object to start from
→	`includeStartingPoint`	`Boolean`	Begin enumeration with starting point?
↔	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record about which you want access control information. You must provide the record location information unless the record resides in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the record creation ID; otherwise, you must provide the record name and type.

`forCurrentUserOnly` A Boolean value that indicates what access control information the function returns. Set this field to `true` if you want access controls only for the user specified in the `identity` field. If you set this field to `false`, the function returns access controls for each object on the record's access control list.

`startingPoint` A pointer to the object from which the function begins returning information. Set this field to `nil` to start with the first object. If the function completes with the `kOCEMoreData` result code, you can set this field to the value of the last `dsObj` parameter passed to your callback routine by the parse function and call the function again to continue to obtain information. The function ignores this field if you set `forCurrentUserOnly` to `true`.

`includeStartingPoint` A Boolean value that determines how the function interprets the `startingPoint` field. Set this field to `true` if you want the function to return access control information beginning with the object specified by the `startingPoint` field. If you set this field to `false`, the function returns information starting with the object after the one specified by the `startingPoint` field. The function ignores this field if you set `forCurrentUserOnly` to `true`.

`getBuffer` A pointer to the buffer in which the function stores the access control information for the record you specify. You provide this buffer.

`getBufferSize` The size, in bytes, of your buffer.

DESCRIPTION

You call the `DirGetRecordAccessControlGet` function to obtain access control information for a record that you specify. The information consists of catalog service specifications that identify the objects on the access control list and of the access control masks that apply to these objects. The mask specifies which access privileges the object possesses.

If the buffer you provide is not large enough to contain all of the information requested, the function returns the `kOCEMoreData` result code.

When the function completes with either the `noErr` or `kOCEMoreData` result codes, you use a pointer to your buffer as input to the `DirGetRecordAccessControlParse` function, which extracts the information from the buffer.

If your buffer is too small to hold all of the requested information, you can get additional information by calling the `DirGetRecordAccessControlGet` function again, after

calling the `DirGetRecordAccessControlParse` function. Set the value of the `startingPoint` field to the value that `DirGetRecordAccessControlParse` last passed to the `dsObj` parameter of your callback routine. The `DirGetRecordAccessControlGet` function will continue to return information starting with the next entry as determined by the value of the `includeStartingPoint` field.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$012C

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchdNode`	–1615	Can't find specified dNode
`kOCENoSuchRecord`	–1618	Can't find specified record
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `DSSpec` data structure is also described in the chapter "AOCE Utilities."

The `DirGetRecordAccessControlParse` function is described next.

For information on the types of access controls specified in the access control mask, see "Getting Access Controls" beginning on page 8-11.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesGet` function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" beginning on page 8-20.

DirGetRecordAccessControlParse

The `DirGetRecordAccessControlParse` function parses the access control information returned by the `DirGetRecordAccessControlGet` function and returns a pointer to each object and its access control mask by repeatedly calling your callback routine.

```
pascal OSErr DirGetRecordAccessControlParse
                                    (DirParamBlockPtr paramBlock,
                                    Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`eachObject`	`ForEachRecordAccessControl`	Your callback routine
→	`forCurrentUserOnly`	`Boolean`	Return only requester's access info?
→	`startingPoint`	`DSSpec*`	Object to start from
→	`includeStartingPoint`	`Boolean`	Begin enumeration with starting point?
→	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record about which you want access control information. Use the same value that you provided to the `DirGetRecordAccessControlGet` function.

`eachObject` A pointer to your callback routine. The Catalog Manager passes your callback routine the value that you provide in the `clientData` field, a pointer to a catalog service specification that identifies the object to which the access control masks apply, the active access control masks for the target record and the dNode in which it resides, and the default access control mask for newly created attribute types within that record. The function declaration for this routine is described on page 8-162.

`forCurrentUserOnly`
Use the same value that you provided to the `DirGetRecordAccessControlGet` function.

`startingPoint` Use the same value that you provided to the `DirGetRecordAccessControlGet` function.

`includeStartingPoint`
Use the same value that you provided to the `DirGetRecordAccessControlGet` function.

`getBuffer`	A pointer to the buffer containing the information to parse. Use the same buffer that you provided to the `DirGetRecordAccessControlGet` function.
`getBufferSize`	The size, in bytes, of your buffer. Use the same value that you provided to the `DirGetRecordAccessControlGet` function.

DESCRIPTION

You call the `DirGetRecordAccessControlParse` function to extract the access control information placed in your buffer by the `DirGetRecordAccessControlGet` function. You must provide a callback routine that the `DirGetRecordAccessControlParse` function calls for each entry it finds in the buffer.

The `DirGetRecordAccessControlParse` function completes when it has finished parsing the contents of your buffer or when your callback routine returns `true`. The function returns the `kOCEMoreData` result code if it reaches the end of the buffer and finds that the `DirGetRecordAccessControlGet` function did not return all the data requested. If you want to continue to obtain information, you can call the `DirGetRecordAccessControlGet` function again. Set the value of the `startingPoint` field to the value that `DirGetRecordAccessControlParse` last passed to the `dsObj` parameter of your callback routine.

If your callback routine returns `true`, the `DirGetRecordAccessControlParse` function completes with the `noErr` result code.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0134

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

The `DSSpec` data structure is also described in the chapter “AOCE Utilities.”

The `DirGetRecordAccessControlGet` function is described on page 8-138.

The function declaration for your callback routine is described on page 8-162.

For information on the types of access controls specified in the access control mask, see "Getting Access Controls" beginning on page 8-11.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesParse` function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" beginning on page 8-20.

DirGetAttributeAccessControlGet

The `DirGetAttributeAccessControlGet` function returns access control information for an attribute type that you specify.

```
pascal OSErr DirGetAttributeAccessControlGet
                                    (DirParamBlockPtr paramBlock,
                                    Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`aType`	`AttributeTypePtr`	Target attribute type
→	`forCurrentUserOnly`	`Boolean`	Return only requester's access info?
→	`startingPoint`	`DSSpec*`	Object to start from
→	`includeStartingPoint`	`Boolean`	Begin enumeration with starting point?
↔	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record in which the target attribute type resides. You must provide the record location information unless the record resides in a personal catalog. If the catalog in which the record resides supports record creation IDs, you must provide the creation ID; otherwise, you must provide the record name and type.

`aType` A pointer to the attribute type about which you are requesting access control information.

`forCurrentUserOnly` A Boolean value that indicates what access controls the function returns. Set this field to `true` if you want access controls only for the user specified in the `identity` field. If you set this field to `false`, the function returns access controls for each object on the attribute type's access control list.

`startingPoint` A pointer to the object from which the function begins to return information. Set this field to `nil` to start with the first object. If the function completes with the `kOCEMoreData` result code, you can set this field to the value of the last `dsObj` parameter passed to your callback routine by the parse function and call the function again to continue to obtain information. The function ignores this field if you set `forCurrentUserOnly` to `true`.

`includeStartingPoint` A Boolean value that determines how the function interprets the `startingPoint` field. Set this field to `true` if you want the function to return access control information beginning with the object specified by the `startingPoint` field. If you set this field to `false`, the function returns information starting with the object after the one specified by the `startingPoint` field. The function ignores this field if you set `forCurrentUserOnly` to `true`.

`getBuffer` A pointer to the buffer in which the function stores the access control information for the attribute type you specify. You provide this buffer.

`getBufferSize` The size, in bytes, of your buffer.

DESCRIPTION

You call the `DirGetAttributeAccessControlGet` function to obtain access control information for an attribute type that you specify. The information consists of catalog service specifications that identify the objects on the access control list and of the access control masks that apply to these objects. The mask specifies which access privileges the object possesses.

If the buffer you provide is not large enough to contain all of the information requested, the function returns the `kOCEMoreData` result code.

When the function completes with either the `noErr` or `kOCEMoreData` result code, you use a pointer to your buffer as input to the `DirGetAttributeAccessControlParse` function, which extracts the information from the buffer.

If your buffer is too small to hold all of the information requested, you can get additional information by calling the `DirGetAttributeAccessControlGet` function again, after calling the `DirGetAttributeAccessControlParse` function. Set the value of the `startingPoint` field to the value that `DirGetAttributeAccessControlParse` last passed to the `dsObj` parameter of your callback routine. The `DirGetAttributeAccessControlGet` function will continue to return information starting with the next object as determined by the value of the `includeStartingPoint` field.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$012E

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCETargetDirectoryInaccessible`	–1613	Target catalog is not currently available
`kOCENoSuchdNode`	–1615	Can’t find specified dNode
`kOCENoSuchRecord`	–1618	Can’t find specified record
`kOCENoSuchPseudonym`	–1619	Can’t find specified pseudonym
`kOCEMoreData`	–1623	More data available

SEE ALSO

The `RecordID` data structure is described in the chapter “AOCE Utilities” in this book.

The `AttributeType` data structure is also described in the chapter “AOCE Utilities.”

The `DSSpec` data structure is also described in the chapter “AOCE Utilities.”

The `DirGetAttributeAccessControlParse` function is described next.

For information on the types of access controls specified in the access control mask, see “Getting Access Controls” beginning on page 8-11.

For an example of continuing the enumeration (using the `DirEnumerateAttributeTypesGet` function) when the buffer is too small to hold all the information you requested, see “Getting Attribute Type Information” beginning on page 8-20.

DirGetAttributeAccessControlParse

The `DirGetAttributeAccessControlParse` function parses the access control information returned by the `DirGetAttributeAccessControlGet` function and returns a pointer to each object and its access control mask by repeatedly calling your callback routine.

```
pascal OSErr DirGetAttributeAccessControlParse
                                    (DirParamBlockPtr paramBlock,
                                    Boolean async);
```

`paramBlock`
Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`serverHint`	`AddrBlock`	AppleTalk address of the PowerShare server
→	`dsRefNum`	`short`	Personal catalog reference number
→	`identity`	`AuthIdentity`	Requester's authentication identity
→	`clientData`	`long`	You define this field
→	`aRecord`	`RecordIDPtr`	Target record
→	`aType`	`AttributeTypePtr`	Target attribute type
→	`eachObject`	`ForEachAttributeAccessControl`	Your callback routine
→	`forCurrentUserOnly`	`Boolean`	Return only requester's access controls?
→	`startingPoint`	`DSSpec*`	Object to start from
→	`includeStartingPoint`	`Boolean`	Begin enumeration with starting point?
→	`getBuffer`	`Ptr`	Your buffer
→	`getBufferSize`	`unsigned long`	Size of your buffer

See "The Parameter Block Header" on page 8-32 for descriptions of the `ioCompletion`, `ioResult`, `serverHint`, `dsRefNum`, `identity`, and `clientData` fields.

Field descriptions

`aRecord` A pointer to a record ID that identifies the record in which the target attribute type resides. Use the same value that you provided to the `DirGetAttributeAccessControlGet` function.

`aType` A pointer to an attribute type about which you are requesting access control information. Use the same value that you provided to the `DirGetAttributeAccessControlGet` function.

`eachObject` A pointer to your callback routine. The Catalog Manager passes your callback routine the value that you provide in the `clientData` field, a pointer to a catalog service specification that identifies the object to which the access control masks apply, and the active access control masks for the target attribute type as well as those for the dNode and record within which it resides. The function declaration for this routine is described on page 8-161.

`forCurrentUserOnly` Use the same value that you provided to the `DirGetAttributeAccessControlGet` function.

`startingPoint` Use the same value that you provided to the `DirGetAttributeAccessControlGet` function.

`includeStartingPoint` Use the same value that you provided to the `DirGetAttributeAccessControlGet` function.

`getBuffer` A pointer to the buffer containing the information to parse. Use the same buffer that you provided to the `DirGetAttributeAccessControlGet` function.

`getBufferSize` The size, in bytes, of your buffer. Use the same value that you provided to the `DirGetAttributeAccessControlGet` function.

DESCRIPTION

You call the `DirGetAttributeAccessControlParse` function to extract the access control information placed in your buffer by the `DirGetAttributeAccessControlGet` function. You must provide a callback routine that the `DirGetAttributeAccessControlParse` function calls for each entry it finds in the buffer.

The `DirGetAttributeAccessControlParse` function completes when it has finished parsing the contents of your buffer or when your callback routine returns `true`. The function returns the `kOCEMoreData` result code if it reaches the end of the buffer and finds that the `DirGetAttributeAccessControlGet` function did not return all the data requested. If you want to continue to obtain information, you can call the `DirGetAttributeAccessControlGet` function again. Set the value of the `startingPoint` field to the value that `DirGetAttributeAccessControlParse` last passed to the `dsObj` parameter of your callback routine.

If your callback routine returns `true`, the `DirGetAttributeAccessControlParse` function completes with the `noErr` result code.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0138

RESULT CODES

noErr	0	No error
kOCEParamErr	–50	Invalid parameter
kOCEMoreData	–1623	More data available

SEE ALSO

The RecordID data structure is described in the chapter "AOCE Utilities" in this book.

The AttributeType data structure is also described in the chapter "AOCE Utilities."

The DSSpec data structure is also described in the chapter "AOCE Utilities."

The DirGetAttributeAccessControlGet function is described on page 8-143.

The function declaration for your callback routine is described on page 8-161.

For information on the types of access controls specified in the access control mask, see "Getting Access Controls" beginning on page 8-11.

For an example of continuing the enumeration (using the DirEnumerateAttributeTypesParse function) when the buffer is too small to hold all the information you requested, see "Getting Attribute Type Information" beginning on page 8-20.

Cancelling a Catalog Manager Function

You use the function described in this section to cancel a Catalog Manager function that has not completed execution.

DirAbort

The DirAbort function cancels a currently executing Catalog Manager function.

```
pascal OSErr DirAbort (DirParamBlockPtr paramBlock);
```

paramBlock
Pointer to a parameter block.

Parameter block

←	ioResult	OSErr	Result code
→	serverHint	AddrBlock	AppleTalk address of the PowerShare server
→	dsRefNum	short	Personal catalog reference number
→	identity	AuthIdentity	Requester's authentication identity
→	pb	union DirParamBlock*	Function to cancel

See “The Parameter Block Header” on page 8-32 for descriptions of the `ioResult`, `serverHint`, `dsRefNum`, and `identity` fields.

Field descriptions

`pb`	A pointer to the `DirParamBlock` parameter block for the function you want to cancel.

DESCRIPTION

You call the `DirAbort` function to cancel a Catalog Manager function that has not completed execution. If the function that you want to cancel addresses a PowerShare catalog or a personal catalog, the Catalog Manager attempts the cancel operation. If the function that you want to cancel addresses an external catalog, the CSAM driver attempts the cancel operation. If the Catalog Manager or the CSAM driver does not support the `DirAbort` function for the executing function that you specify, the function returns the `kOCEAbortNotSupportedForThisCall` result code.

PowerShare and personal catalogs support the `DirAbort` function for the `DirFindADAPDirectoryByNetSearch` and `DirNetSearchADAPDirectoriesGet` functions only.

IMPORTANT

Because the `DirAbort` function makes references to fields in the parameter block associated with the function that you want to cancel, you must not alter or dispose of that parameter block before the `DirAbort` function has completed. s

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$011B

RESULT CODES

`noErr`	0	No error
`kOCEAbortNotSupportedForThisCall`	–1631	Abort not supported

SEE ALSO

The `DirFindADAPDirectoryByNetSearch` function is described on page 8-74.

The `DirNetSearchADAPDirectoriesGet` function is described on page 8-76.

Application-Defined Functions

This section contains descriptions of the completion routine you can provide when you call the Catalog Manager asynchronously and of the callback routines that you provide to Catalog Manager functions that parse a buffer's contents.

The information on callback routines in this introduction applies to all callback routines and is not repeated in individual routine descriptions.

The Catalog Manager manages all of the buffers associated with pointers that it passes to a callback routine. You must copy the data in these buffers if you want to refer to it after your callback routine completes execution.

When a callback routine returns `false`, the parse function continues parsing the results in your buffer. When a callback routine returns `true`, the parse function completes with a `noErr` result code. If a parse function invokes a callback routine and passes it the last item in the buffer and the callback routine returns `false`, the parse routine completes with either a `noErr` or a `kOCEMoreData` result code, depending on the result code of the corresponding "get" function.

A Catalog Manager function always calls a callback routine at deferred-task time so that it will work properly if the computer is using virtual memory. Because these functions restore the value of your application's A5 register before calling a callback routine, a callback routine has access to your application's global variables. Your callback routine can allocate memory if you make a synchronous call to the function that invokes it.

See "Callback Routines" on page 8-10 for more information about the restrictions that apply to callback routines.

MyCompletionRoutine

You may provide a completion routine when you call a Catalog Manager function asynchronously.

```
void MyCompletionRoutine (DirParamBlockPtr paramBlk);
```

`paramBlk` A pointer to the parameter block that you provided to the Catalog Manager function that is calling your completion routine.

DESCRIPTION

You can provide a completion routine to any Catalog Manager functions that you can call asynchronously by passing a pointer to the completion routine in the `ioCompletion` field of the `DirParamBlock` parameter block. If you provide a completion routine, it executes when the asynchronous request completes execution.

The Catalog Manager saves the value of your A5 register at the time you call a Catalog Manager routine and then restores the A5 value before calling the completion routine.

The Catalog Manager always calls completion routines in deferred-task time. Running at deferred-task time is a safe practice when using 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 MyCompletion(paramBlk: DirParamBlockPtr);
```

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 a Catalog Manager function asynchronously, the function 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, it sets the `ioResult` field to the actual function result. If you poll, you should do so within a loop that calls the `WaitNextEvent` routine so that other processes get execution time. If you poll in a tight loop, you may cause a deadlock condition.

ASSEMBLY-LANGUAGE INFORMATION

If you write it in assembly language, your completion routine gets a pointer to the parameter block in the A0 register and the Catalog Manager function result code in the D0 register. The function result code is also available in the `ioResult` field of the parameter block.

MyForEachRecordID

The `MyForEachRecordID` function is a callback routine you must provide if you call the `DirEnumeratePseudonymParse` function.

```
pascal Boolean MyForEachRecordID (long clientData,
                                  const RecordID *recordID);
```

`clientData`
: The value that you provided in the `clientData` field of the parameter block that you passed to the `DirEnumeratePseudonymParse` function. You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirEnumeratePseudonymParse` function, you can use this parameter to match calls to this routine with a particular call to the `DirEnumeratePseudonymParse` function.

`recordID`
: A pointer to a record ID containing the name, the type, and the creation ID of a pseudonym. The record location information is unspecified because the pseudonym resides in the same catalog and dNode as the target record.

DESCRIPTION

The `DirEnumeratePseudonymParse` function calls your callback routine for each pseudonym it finds in a buffer previously filled by the `DirEnumeratePseudonymGet` function.

SEE ALSO

Read the introduction to "Application-Defined Functions" on page 8-150 for important information that applies to all callback routines.

The `DirEnumeratePseudonymParse` function is described on page 8-104.

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

MyForEachAttrType

The `MyForEachAttrType` function is a callback routine you must provide if you call the `DirEnumerateAttributeTypesParse` function.

```
pascal Boolean MyForEachAttrType (long clientData,
                                  const AttributeType *attrType);
```

`clientData`
: The value that you provided in the `clientData` field of the parameter block that you passed to the `DirEnumerateAttributeTypesParse` function. You can use this field for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirEnumerateAttributeTypesParse` function, you can use this field to match calls to this routine with a particular call to the `DirEnumerateAttributeTypesParse` function.

`attrType`
: A pointer to an `AttributeType` data structure.

DESCRIPTION

The `DirEnumerateAttributeTypesParse` function calls your callback routine for each attribute type that it finds in a buffer previously filled by the `DirEnumerateAttributeTypesGet` function.

SEE ALSO

Read the introduction to "Application-Defined Functions" on page 8-150 for important information that applies to all callback routines.

The `DirEnumerateAttributeTypesParse` function is described on page 8-130

The `AttributeType` data structure is described in the chapter "AOCE Utilities" in this book.

MyForEachDirectory

The `MyForEachDirectory` function is a callback routine you must provide if you call the `DirEnumerateDirectoriesParse` function.

```
pascal Boolean MyForEachDirectory (long clientData,
                                   const DirectoryName *dirName,
                                   const DirDiscriminator *discriminator,
                                   DirGestalt features);
```

`clientData`
: The value that you provided in the `clientData` field of the parameter block that you passed to the `DirEnumerateDirectoriesParse` function. You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirEnumerateDirectoriesParse` function, you can use this parameter to match calls to this routine with a particular call to the `DirEnumerateDirectoriesParse` function.

`dirName`
: A pointer to the name of a catalog.

`discriminator`
: A pointer to a `DirDiscriminator` data structure that differentiates between catalogs that share the same name.

`features`
: A set of flags that indicates the features that the catalog supports.

DESCRIPTION

The `DirEnumerateDirectoriesParse` function calls your callback routine for each catalog entry that it finds in a buffer previously filled by the `DirEnumerateDirectoriesGet` function. Your callback routine receives a pointer to the catalog's name, a pointer to its discriminator value, and the feature flags that indicate what features the catalog supports.

The `DirEnumerateDirectoriesParse` function does not supply information about personal catalogs.

SEE ALSO

Read the introduction to "Application-Defined Functions" on page 8-150 for important information that applies to all callback routines.

The `DirEnumerateDirectoriesParse` function is described on page 8-41.

The `DirDiscriminator` data structure is described in the chapter "AOCE Utilities" in this book.

For a description of catalog feature flags, see "Feature Flag Bit Array" beginning on page 8-28.

MyForEachLookupRecordID

The `MyForEachLookupRecordID` function is a callback routine that you may provide if you call the `DirLookupParse` function and you want to get record information.

```
pascal Boolean MyForEachLookupRecordID (long clientData,
                                        const RecordID *recordID);
```

`clientData`
The value that you provided in the `clientData` field of the parameter block that you passed to the `DirLookupParse` function. You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirLookupParse` function, you can use this field to match calls to this routine with a particular call to the `DirLookupParse` function.

`recordID` A pointer to a record ID.

DESCRIPTION

The `DirLookupParse` function calls your callback routine for each record ID that it finds in a buffer previously filled by the `DirLookupGet` function.

This callback routine is optional. If you look up attribute values only in a single record, you may not want to provide this routine. However, then you cannot distinguish between the case where a record exists but an attribute type does not exist and the case where a record does not exist.

If you look up attribute values in multiple records, you need to provide this routine to associate attribute types and attribute values with the record to which they belong.

SEE ALSO

Read the introduction to "Application-Defined Functions" on page 8-150 for important information that applies to all callback routines.

The `RecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `DirLookupParse` function is described on page 8-121.

MyForEachAttrTypeLookup

The `MyForEachAttrTypeLookup` function is a callback routine which you may provide if you call the `DirLookupParse` function and you want to retrieve attribute type information.

```
pascal Boolean MyForEachAttrTypeLookup (long clientData,
                                        const AttributeType *attrType,
                                        AccessMask myAttrAccMask);
```

`clientData`
: The value that you provided in the `clientData` field of the parameter block that you passed to the `DirLookupParse` function. You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirLookupParse` function, you can use this parameter to match calls to this routine with a particular call to the `DirLookupParse` function.

`attrType`
: A pointer to an `AttributeType` data structure.

`myAttrAccMask`
: The requester's access control mask for this attribute type. If the `myAttrAccMask` parameter indicates that you do not have read access permission for this attribute type, the `DirLookupParse` function does not call your `MyForEachAttrValue` callback routine for this attribute type.

DESCRIPTION

The `DirLookupParse` function calls this callback routine for each attribute type that it finds in a buffer previously filled by the `DirLookupGet` function.

If you provided a callback routine for record ID information (`MyForEachLookupRecordID`), you can associate the attribute type that the function passes here with the record ID that the `DirLookupParse` function most recently passed to your `MyForEachLookupRecordID` callback routine.

This callback routine is optional. However, it provides access control information about each attribute type that you requested. If you do not have read access to an attribute type that you requested, you can still detect the presence of attribute values of that type in a record. However, you cannot read those attribute values because the `DirLookupParse` function does not call your attribute value callback routine (`MyForEachAttrValue`) when you lack read access to the attribute type. If you do not provide this routine, you have no way of knowing that attribute values that you requested exist in a record when you lack read access to their attribute type.

SEE ALSO

Read the introduction to "Application-Defined Functions" on page 8-150 for important information that applies to all callback routines.

The `AttributeType` data structure is described in the chapter "AOCE Utilities" in this book.

The values of the access mask are described in "Getting Access Controls" beginning on page 8-11.

The `DirLookupParse` function is described on page 8-121.

The `DirLookupGet` function is described on page 8-118.

The `MyForEachLookupRecordID` routine is described on page 8-154.

The `MyForEachAttrValue` routine is described next.

MyForEachAttrValue

The `MyForEachAttrValue` function is a callback routine you must provide if you call the `DirLookupParse` function.

```
pascal Boolean MyForEachAttrValue (long clientData,
                                   const Attribute *attribute);
```

`clientData`
The value that you provided in the `clientData` field of the parameter block that you passed to the `DirLookupParse` function. You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirLookupParse` function, you can use this parameter to match calls to this routine with a particular call to the `DirLookupParse` function.

`attribute` A pointer to an `Attribute` data structure that specifies an attribute value. Note that this attribute value has the attribute type specified in the most recent call to your `MyForEachAttrTypeLookup` callback routine, and it is contained in the record specified by the most recent call to your `MyForEachLookupRecordID` callback routine.

DESCRIPTION

The `DirLookupParse` function calls this callback routine for each attribute value that it finds in a buffer previously filled by the `DirLookupGet` function.

If you provided a callback routine for record ID information (`MyForEachLookupRecordID`), you can associate the attribute value that the function passes here with the local record ID that the `DirLookupParse` function most recently passed to your `MyForEachLookupRecordID` callback routine. If you did not provide a callback routine for record ID information and you are looking up attribute values in

multiple records, you must devise your own system of matching attribute values with the records to which they belong.

If you provided a callback routine for attribute type information (`MyForEachAttrTypeLookup`), you can associate the attribute value that the function passes here with the access controls that the `DirLookupParse` function most recently passed to your `MyForEachAttrTypeLookup` callback routine. If you did not provide a callback routine for attribute type information, you cannot detect the presence of attribute values in a record when you lack read access to their attribute types. The `DirLookupParse` function does not call this callback routine for an attribute value if you do not have read access to its attribute type.

SEE ALSO

Read the introduction to "Application-Defined Functions" on page 8-150 for important information that applies to all callback routines.

The `DirLookupParse` function is described on page 8-121.

The `DirLookupGet` function is described on page 8-118.

The `Attribute` data structure is described in the chapter "AOCE Utilities" in this book.

The `MyForEachLookupRecordID` routine is described on page 8-154.

The `MyForEachAttrTypeLookup` routine is described on page 8-155.

MyForEachDirEnumSpec

The `MyForEachDirEnumSpec` function is a callback routine you must provide if you call the `DirEnumerateParse` function.

```
pascal Boolean MyForEachDirEnumSpec (long clientData,
                                     const DirEnumSpec *enumSpec);
```

`clientData`
: The value that you provided in the `clientData` field of the parameter block that you passed to the `DirEnumerateParse` function.You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirEnumerateParse` function, you can use this parameter to match calls to this routine with a particular call to the `DirEnumerateParse` function.

`enumSpec`
: A pointer to an enumeration specification data structure. The value of the `enumFlag` field of the `DirEnumSpec` data structure indicates the type of entity about which information is being returned. Use the mask constants `kEnumDistinguishedNameMask`, `kEnumAliasMask`, `kEnumPseudonymMask`, and `kEnumDNodeMask` to determine if the

entity is a record, alias, pseudonym, or dNode, respectively. Use the mask constant `kEnumInvisibleMask` to determine if the entity is visible or invisible.

If the `DirEnumerateParse` function is returning information about a dNode or an invisible dNode, the `u` field of the `DirEnumSpec` structure contains a `DNodeID` data structure. The dNode ID consists of the name of the dNode and its dNode number. If the catalog does not support dNode numbers, the dNode number is set to 0.

If the `DirEnumerateParse` function is returning information about a record, an alias, or a pseudonym, the `u` field of the `DirEnumSpec` structure contains a `LocalRecordID` data structure. The local record ID consists of the record's creation ID, name, and type. If the catalog does not support creation IDs, the creation ID is set to 0.

DESCRIPTION

The `DirEnumerateParse` function calls your callback routine for each record, alias, pseudonym, and dNode about which it finds information in a buffer previously filled by the `DirEnumerateGet` function.

Invisible dNodes are typically foreign dNodes, that is, they represent external messaging systems within an AOCE system. Invisible records are typically those that are used in administering an AOCE system. Usually, you would not display information about these to a user. It is up to your application to consider how to handle information about invisible entities.

SEE ALSO

Read the introduction to "Application-Defined Functions" on page 8-150 for important information that applies to all callback routines.

The `DirEnumerateParse` function is described on page 8-62.

The `DirEnumerateGet` function is described on page 8-57.

The `LocalRecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `DirEnumSpec` data structure is described on page 8-35.

The `DNodeID` data structure is described on page 8-34.

MyForEachRecord

The `MyForEachRecord` function is a callback routine you must provide if you call the `DirFindRecordParse` function.

```
pascal Boolean MyForEachRecord (long clientData,
                                const DirEnumSpec *enumSpec,
                                pRLI PackedRLIPtr);
```

`clientData`
: The value that you provided in the `clientData` field of the parameter block that you passed to the `DirFindRecordParse` function.You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirFindRecordParse` function, you can use this parameter to match calls to this routine with a particular call to the `DirFindRecordParse` function.

`enumSpec`
: A pointer to an enumeration specification data structure. The value of the `enumFlag` field of the `DirEnumSpec` data structure indicates the type of entity about which information is being returned. Use the mask constants `kEnumDistinguishedNameMask`, `kEnumAliasMask`, and `kEnumPseudonymMask` to determine if the entity is a record, alias, or pseudonym, respectively.

`pRLI`
: A pointer to packed record location information that specifies the dNode within which the record, alias, or pseudonym is located.

DESCRIPTION

The `DirFindRecordParse` function calls your callback routine for each record, alias, and pseudonym about which it finds information in a buffer previously filled by the `DirFindRecordGet` function.

SEE ALSO

Read the introduction to "Application-Defined Functions" on page 8-150 for important information that applies to all callback routines.

The `DirFindRecordParse` function is described on page 8-46.

The `DirFindRecordGet` function is described on page 8-43.

The `LocalRecordID` data structure is described in the chapter "AOCE Utilities" in this book.

The `DirEnumSpec` data structure is described on page 8-35.

MyForEachADAPDirectory

The `MyForEachADAPDirectory` function is a callback routine you must provide if you call the `DirNetSearchADAPDirectoriesParse` function.

```
pascal Boolean MyForEachADAPDirectory (long clientData,
                            const DirectoryName *directoryName,
                            const DirDiscriminator *discriminator,
                            DirGestalt features,
                            AddrBlock serverHint);
```

`clientData`
The value that you provided in the `clientData` field of the parameter block that you passed to the `DirNetSearchADAPDirectoriesParse` function. You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirNetSearchADAPDirectoriesParse` function, you can use this parameter to match calls to this routine with a particular call to the `DirNetSearchADAPDirectoriesParse` function.

`directoryName`
A pointer to the name of the catalog.

`discriminator`
A pointer to the value that differentiates two or more catalogs with the same name.

`features` A set of feature bit flags for the catalog.

`serverHint`
The AppleTalk address of a PowerShare server that serves the catalog specified in the `directoryName` and `discriminator` fields.

DESCRIPTION

The `DirNetSearchADAPDirectoriesParse` function calls your callback routine for each PowerShare catalog that it finds in a buffer previously filled by the `DirNetSearchADAPDirectoriesGet` function.

SEE ALSO

Read the introduction to "Application-Defined Functions" on page 8-150 for important information that applies to all callback routines.

The `DirNetSearchADAPDirectoriesParse` function is described on page 8-78.

The `DirDiscriminator` data structure is described in the chapter "AOCE Utilities" in this book.

For a description of catalog feature flags, see "Feature Flag Bit Array" beginning on page 8-28.

The `AddrBlock` data structure is described in the header file AppleTalk.h.

MyForEachDNodeAccessControl

The `MyForEachDNodeAccessControl` function is a callback routine you must provide when you call the `DirGetDNodeAccessControlParse` function.

```
pascal Boolean MyForEachDNodeAccessControl (long clientData,
                        const DSSpec *dsObj,
                        AccessMask activeDnodeAccMask,
                        AccessMask defaultRecordAccMask,
                        AccessMask defaultAttributeAccMask);
```

`clientData`
: The value that you provided in the `clientData` field of the parameter block that you passed to the `DirGetDNodeAccessControlParse` function. You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirGetDNodeAccessControlParse` function, you can use this parameter to match calls to this routine with a particular call to the `DirGetDNodeAccessControlParse` function.

`dsObj`
: A pointer to the object to which the access control mask applies.

`activeDnodeAccMask`
: A mask that specifies the access controls that apply to the object in relation to the dNode.

`defaultRecordAccMask`
: A mask that specifies the default access controls that apply to new records within the dNode.

`defaultAttributeAccMask`
: A mask that specifies the default access controls that apply to new attribute types within records in the dNode.

DESCRIPTION

The `DirGetDNodeAccessControlParse` function calls your callback routine for each entry that it finds in a buffer previously filled by the `DirGetDNodeAccessControlGet` function.

SEE ALSO

Read the introduction to "Application-Defined Functions" on page 8-150 for important information that applies to all callback routines.

The `DirGetDNodeAccessControlGet` function is described on page 8-133.

The `DirGetDNodeAccessControlParse` function is described on page 8-136.

The values of the access mask are described in "Getting Access Controls" beginning on page 8-11.

MyForEachRecordAccessControl

The `MyForEachRecordAccessControl` function is a callback routine you must provide when you call the `DirGetRecordAccessControlParse` function.

```
pascal Boolean MyForEachRecordAccessControl (long clientData,
                        const DSSpec *dsObj,
                        AccessMask activeDnodeAccMask,
                        AccessMask activeRecordAccMask,
                        AccessMask defaultAttributeAccMask);
```

`clientData`
The value that you provided in the `clientData` field of the parameter block that you passed to the `DirGetRecordAccessControlParse` function. You can use this parameter for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirGetRecordAccessControlParse` function, you can use this parameter to match calls to this routine with a particular call to the `DirGetRecordAccessControlParse` function.

`dsObj` A pointer to the object to which the access control mask applies.

`activeDnodeAccMask`
A mask that specifies the access controls that apply to the object in relation to the dNode that contains the record.

`activeRecordAccMask`
A mask that specifies the access controls that apply to the object in relation to the record.

`defaultAttributeAccMask`
A mask that specifies the default access controls that apply to new attribute types within the record.

DESCRIPTION

The `DirGetRecordAccessControlParse` function calls your callback routine for each entry that it finds in a buffer previously filled by the `DirGetRecordAccessControlGet` function.

Read the introduction to "Application-Defined Functions" on page 8-150 for important information that applies to all callback routines.

SEE ALSO

The `DirGetRecordAccessControlGet` function is described on page 8-138.

The `DirGetRecordAccessControlParse` function is described on page 8-140.

The values of the access mask are described in "Getting Access Controls" beginning on page 8-11.

MyForEachAttributeAccessControl

The `MyForEachAttributeAccessControl` function is a callback routine you must provide when you call the `DirGetAttributeAccessControlParse` function.

```
pascal Boolean MyForEachAttributeAccessControl (long clientData,
                        const DSSpec *dsObj,
                        AccessMask activeDnodeAccMask,
                        AccessMask activeRecordAccMask,
                           AccessMask activeAttributeAccMask);
```

`clientData`
: The value that you provided in the `clientData` field of the parameter block that you passed to the `DirGetAttributeAccessControlParse` function. You can use this field for whatever purpose you choose. For example, if you make multiple asynchronous calls to the `DirGetAttributeAccessControlParse` function, you can use this field to match calls to this routine with a particular call to the `DirGetAttributeAccessControlParse` function.

`dsObj`
: A pointer to the object to which the access control mask applies.

`activeDnodeAccMask`
: A mask that specifies the access controls that apply to the object in relation to the dNode containing the record that contains the attribute type.

`activeRecordAccMask`
: A mask that specifies the access controls that apply to the object in relation to the record that contains the attribute type.

`activeAttributeAccMask`
: A mask that specifies the access controls that apply to the object in relation to the attribute type.

DESCRIPTION

The `DirGetAttributeAccessControlParse` function calls your callback routine for each entry that it finds in a buffer previously filled by the `DirGetAttributeAccessControlGet` function.

Read the introduction to "Application-Defined Functions" on page 8-150 for important information that applies to all callback routines.

SEE ALSO

The `DirGetAttributeAccessControlGet` function is described on page 8-143.

The `DirGetAttributeAccessControlParse` function is described on page 8-146.

The values of the access mask are described in "Getting Access Controls" beginning on page 8-11.

Summary of the Catalog Manager

C Summary

Constants and Data Types

```
enum {
   kThisRecordOwnerBit           =  0,
   kFriendsBit                   =  1,
   kAuthenticatedInDNodeBit      =  2,
   kAuthenticatedInDirectoryBit  =  3,
   kGuestBit                     =  4,
   kMeBit                        =  5
};

enum {         /* Values of CategoryMask */
   kThisRecordOwnerMask          = (1L << kThisRecordOwnerBit),
   kFriendsMask                  = (1L << kFriendsBit),
   kAuthenticatedInDNodeMask     = (1L << kAuthenticatedInDNodeBit),
   kAuthenticatedInDirectoryMask = (1L << kAuthenticatedInDirectoryBit),
   kGuestMask                    = (1L << kGuestBit),
   kMeMask                       = (1L << kMeBit)

typedef unsigned long CategoryMask;

};enum {
      kEnumDistinguishedNameBit,
      kEnumAliasBit,
      kEnumPseudonymBit,
      kEnumDNodeBit,
      kEnumInvisibleBit
};

enum {
      /* values of DirEnumChoices */
      kEnumDistinguishedNameMask = 1L<<kEnumDistinguishedNameBit,
      kEnumAliasMask             = 1L<<kEnumAliasBit,
      kEnumPseudonymMask         = 1L<<kEnumPseudonymBit,
```

```
      kEnumDNodeMask             = 1L<<kEnumDNodeBit,
      kEnumInvisibleMask         = 1L<<kEnumInvisibleBit
};

#define kEnumAllMask (kEnumDistinguishedNameMask | kEnumAliasMask |
                      kEnumPseudonymMask | kEnumDNodeMask |
                      EnumInvisibleMask)

typedef unsigned long DirEnumChoices;

/* values of DirSortOption */
enum {
      kSortByName= 0,
      kSortByType= 1
};

typedef unsigned short DirSortOption;

/* values of DirSortDirection */

enum {
      kSortForwards= 0,
      kSortBackwards= 1
};

typedef unsigned short DirSortDirection;

/* values of DirMatchWith */
enum {
      kMatchAll,
      kExactMatch,
      kBeginsWith,
      kEndingWith,
      kContaining
};

typedef unsigned char DirMatchWith;

#define kCurrentOCESortVersion11

enum {
      kSupportsDNodeNumberBit
      kSupportsRecordCreationIDBit
      kSupportsAttributeCreationIDBit
      kSupportsMatchAllBit
      kSupportsBeginsWithBit
```

```
        kSupportsExactMatchBit
        kSupportsEndsWithBit
        kSupportsContainsBit
        kSupportsOrderedEnumerationBit
        kCanSupportNameOrderBit
        kCanSupportTypeOrderBit
        kSupportsSortBackwardsBit
        kSupportIndexRatioBit
        kSupportsEnumerationContinueBit
        kSupportsLookupContinueBit
        kSupportsEnumerateAttributeTypeContinueBit
        kSupportsEnumeratePseudonymContinueBit
        kSupportsAliasesBit
        kSupportsPseudonymsBit
        kSupportsPartialPathnamesBit
        kSupportsAuthenticationBit
        kSupportsProxiesBit
        kSupportsFindRecordBit
};

/* values of 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
};

typedef unsigned long DirGestalt;

struct DNodeID {
   DNodeNum    dNodeNumber;   /* dNode number  */
   long        reserved1;     /* reserved */
   RStringPtr  name;          /* name of the dNode */
   long        reserved2;     /* reserved */
};

typedef struct DNodeID DNodeID;

struct DirEnumSpec {
   DirEnumChoices    enumFlag;
   unsigned short    indexRatio; /* if supported, record position between 1
                                    and 100. 0 if not supported */
   union {
      LocalRecordID  recordIdentifier;
      DNodeID        dNodeIdentifier;
   }u;
};

typedef struct DirEnumSpec DirEnumSpec;

struct DirMetaInfo {
   unsigned longinfo[4];
};

typedef struct DirMetaInfo DirMetaInfo;

struct SLRV {
   ScriptCode  script;     /* script code in which entries are sorted */
   short       language;   /* language code in which entries are sorted */
   short       regionCode; /* region code in which entries are sorted */
   short       version;    /* version of AOCE sorting software */
};

typedef struct SLRV SLRV;

typedef unsigned long AuthIdentity;
```

```
typedef pascal Boolean (*ForEachRecordID) (long clientData,
                                           const RecordID* recordID);

typedef pascal Boolean (*ForEachAttrType) (long clientData,
                                           const AttributeType *attrType);

{ FUNCTION ForEachLookupRecordID(clientData: long; recordID: RecordID): BOOL-
EAN;}

{ FUNCTION ForEachAttrTypeLookup(clientData: long; attrType:
AttributeTypePtr; myAttrAccMask: AccessMask): BOOLEAN;}

{ FUNCTION ForEachAttrValue(clientData: long; attribute: Attribute):
BOOLEAN;}

typedef pascal Boolean (*ForEachDNodeAccessControl) (long clientData,
                        const DSSpec *dsObj, AccessMask activeDnodeAccMask,
                        AccessMask defaultRecordAccMask,
                        AccessMask defaultAttributeAccMask);

#define  AuthDirParamHeader
   Ptr            qLink;         /* reserved */\
   long           reserved_H1;   /* reserved */\
   long           reserved_H2;   /* reserved */\
   ProcPtr        ioCompletion;  /* your completion routine */\
   OSErr          ioResult;      /* result code */\
   unsigned long  saveA5;        /* reserved */\
   short          reqCode;       /* CSAM request code*/\
   long           reserved[2];   /* reserved */\
   AddrBlock      serverHint;    /* PowerShare server's AppleTalk address */
   short          dsRefNum;      /* personal catalog reference number */\
   unsigned long  callID;        /* reserved */\
   AuthIdentity   identity;      /* requester's authentication identity */
   long           gReserved1;    /* reserved */\
   long           gReserved2;    /* reserved */\
   long           gReserved3;    /* reserved */\
   long           clientData;    /* you define this field */

struct DirEnumerateDirectoriesGetPB {
   AuthDirParamHeader
   OCEDirectoryKind  directoryKind;            /* enumerate catalogs
                                                  bearing this signature */
   DirectoryNamePtr  startingDirectoryName;    /* starting catalog */
   DirDiscriminator  startingDirDiscriminator; /* starting catalog
                                                  discriminator */
```

```
   long              eReserved;
   long              fReserved;
   long              gReserved;
   long              hReserved;
   Boolean           includeStartingPoint;      /* if true, return the
                                                   catalog specified by
                                                   starting point */
   Byte              padByte;
   short             i1Reserved;
   Ptr               getBuffer;
   unsigned long     getBufferSize;
};

typedef struct DirEnumerateDirectoriesGetPB DirEnumerateDirectoriesGetPB;

struct DirEnumerateDirectoriesParsePB {
   AuthDirParamHeader
   long              aReserved;
   long              bReserved;
   long              cReserved;
   long              dReserved;
   ForEachDirectory  eachDirectory;
   long              fReserved;
   long              gReserved;
   long              hReserved;
   long              iReserved;
   Ptr               getBuffer;
   unsigned long     getBufferSize;
};

typedef struct DirEnumerateDirectoriesParsePB DirEnumerateDirectoriesParsePB;

struct DirFindRecordGetPB {
   AuthDirParamHeader
   RecordIDPtr       startingPoint;
   long              reservedA[2];
   RStringPtr        nameMatchString;
   RStringPtr*       typesList;
   unsigned long     typeCount;
   long              reservedB;
   short             reservedC;
   DirMatchWith      matchNameHow;
   DirMatchWith      matchTypeHow;
   Ptr               getBuffer;
```

```
   unsigned long     getBufferSize;
   DirectoryNamePtr  directoryName;
   DirDiscriminator  discriminator;
};

typedef struct DirFindRecordGetPB DirFindRecordGetPB;

struct DirFindRecordParsePB {
   AuthDirParamHeader
   RecordIDPtr       startingPoint;
   long              reservedA[2];
   RStringPtr        nameMatchString;
   RStringPtr*       typesList;
   unsigned long     typeCount;
   long              reservedB;
   short             reservedC;
   DirMatchWith      matchNameHow;
   DirMatchWith      matchTypeHow;
   Ptr               getBuffer;
   unsigned long     getBufferSize;
   DirectoryNamePtr  directoryName;
   DirDiscriminator  discriminator;
   ForEachRecord     forEachRecordFunc;
};

typedef struct DirFindRecordParsePB DirFindRecordParsePB;

struct DirGetDirectoryInfoPB {
   AuthDirParamHeader
   DirectoryNamePtr  directoryName;    /* catalog name */

   DirDiscriminator  discriminator; /* descriminate between duplicate
                                       catalog names */
   DirGestalt        features;      /* capability bit flags */
};

typedef struct DirGetDirectoryInfoPB DirGetDirectoryInfoPB;

struct DirGetLocalNetworkSpecPB {
   AuthDirParamHeader
   DirectoryNamePtr  directoryName;    /* catalog name */
   DirDiscriminator  discriminator;    /* discriminator */
   NetworkSpecPtr    networkSpec;      /* NetworkSpec */
};
```

```
typedef struct DirGetLocalNetworkSpecPB DirGetLocalNetworkSpecPB;

struct DirGetDirectoryIconPB {
   AuthDirParamHeader
   PackedRLIPtr      pRLI;            /* packed RLI for the catalog */
   OSType            iconType;        /* type of icon requested */
   Ptr               iconBuffer;      /* buffer to hold icon data */
   unsigned long     bufferSize;      /* size of buffer to hold icon data */
};

typedef struct DirGetDirectoryIconPB DirGetDirectoryIconPB;

struct DirGetExtendedDirectoriesInfoPB {
   AuthDirParamHeader
   Ptr               buffer;          /* Pointer to a buffer where data
                                         will be returned */
   unsigned long     bufferSize;         length of actual data will be
                                         returned here */
   unsigned long     totalEntries;    /* total number of catalogs found */
   unsigned long     actualEntries;   /* total number of catalog entries */
                                         returned */
};

typedef struct DirGetExtendedDirectoriesInfoPB
                                         DirGetExtendedDirectoriesInfoPB;

typedef pascal Boolean (*ForEachDirectory) (
   long clientData, const DirectoryName *dirName,
   const DirDiscriminator *discriminator,DirGestalt features);

struct DirEnumerateGetPB {
   AuthDirParamHeader
   PackedRLIPtr      aRLI;            /* an RLI specifying the cluster
                                         to be enumerated */
   DirEnumSpec       *startingPoint;
   DirSortOption     sortBy;
   DirSortDirection  sortDirection;
   long              dReserved;
   RStringPtr        nameMatchString; /* name from which enumeration should
                                         start */
   RStringPtr        *typesList;      /* list of entity types to be
                                         enumerated */
   unsigned long     typeCount;       /*  number of types in the list */
   DirEnumChoices    enumFlags;       /*  indicates what to enumerate */
   Boolean           includeStartingPoint;
```

```
                                        /* if true, return the record
                                           specified in starting point */
   Byte              padByte;
   DirMatchWith      matchNameHow;      /* matching criteria for
                                           nameMatchString */
   DirMatchWith      matchTypeHow;      /* matching riteria for typeList */
   Ptr               getBuffer;
   unsigned long     getBufferSize;
   SLRV              responseSLRV;      /* response SLRV */
};c

typedef struct DirEnumerateGetPB DirEnumerateGetPB;

struct DirEnumerateParsePB {
   AuthDirParamHeader
   PackedRLIPtr         aRLI;          /* an RLI specifying the cluster to
                                           be enumerated */
   long                 bReserved;
   long                 cReserved;
   ForEachDirEnumSpec   eachEnumSpec;
   long                 eReserved;
   long                 fReserved;
   long                 gReserved;
   long                 hReserved;
   long                 iReserved;
   Ptr                  getBuffer;
   unsigned long        getBufferSize;
   short                l1Reserved;
   short                l2Reserved;
   short                l3Reserved;
   short                l4Reserved;

};

typedef struct DirEnumerateParsePB DirEnumerateParsePB;

struct DirGetDNodeMetaInfoPB {
   AuthDirParamHeader
   PackedRLIPtr   pRLI;
   DirMetaInfo    metaInfo;
};

typedef struct DirGetDNodeMetaInfoPB DirGetDNodeMetaInfoPB;
```

```
struct DirMapDNodeNumberToPathNamePB {
   AuthDirParamHeader
   DirectoryNamePtr  directoryName;       /* catalog name */
   DirDiscriminator  discriminator;       /* discriminator */
   DNodeNum          dNodeNumber;         /* dNode number to be mapped */
   PackedPathNamePtr path;                /* packed pathname returned */
   unsigned short    lengthOfPathName;
                                          /* length of packed pathname
                                             structure*/
};

typedef struct DirMapDNodeNumberToPathNamePB DirMapDNodeNumberToPathNamePB;

struct DirMapPathNameToDNodeNumberPB {
   AuthDirParamHeader
   DirectoryNamePtr     directoryName;    /* catalog name */
   DirDiscriminator     discriminator;    /* discriminator */
   DNodeNum             dNodeNumber;      /* dNode number to the path */
   PackedPathNamePtr    path;             /* pathname to be mapped */
};

typedef struct DirMapPathNameToDNodeNumberPB DirMapPathNameToDNodeNumberPB;

struct DirGetDNodeInfoPB {
   AuthDirParamHeader
   PackedRLIPtr   pRLI;             /* packed RLI whose info is requested */
   DirNodeKind    descriptor;       /* dNode descriptor */
   NetworkSpecPtr networkSpec;      /* cluster's networkSpec if kIsCluster */
};

typedef struct DirGetDNodeInfoPB DirGetDNodeInfoPB;

struct DirAddADAPDirectoryPB {
   AuthDirParamHeader
   DirectoryNamePtr  directoryName;    /* catalog name */
   DirDiscriminator  discriminator;    /* discriminate between duplicate
                                          catalog names */
   Boolean           addToOCESetup;    /* add this catalog to PowerTalk
                                          Setup */
   Byte              padByte;
   CreationID        directoryRecordCID;
                                       /* creation ID for the catalog
                                          record */
};
```

```
typedef struct DirAddADAPDirectoryPB DirAddADAPDirectoryPB;

struct DirFindADAPDirectoryByNetSearchPB {
   AuthDirParamHeader
   DirectoryNamePtr  directoryName;    /* catalog name */
   DirDiscriminator  discriminator;    /* discriminate between duplicate
                                          catalog names */
   Boolean           addToOCESetup;    /* add this catalog to PowerTalk
                                          setup list */
   Byte              padByte;
   CreationID        directoryRecordCID;
                                       /* creation ID for the catalog
                                          record */
};

typedef struct DirFindADAPDirectoryByNetSearchPB
                                           DirFindADAPDirectoryByNetSearchPB;

struct DirNetSearchADAPDirectoriesGetPB {
   AuthDirParamHeader
   Ptr            getBuffer;
   unsigned long  getBufferSize;
   long           cReserved;
};

typedef struct DirNetSearchADAPDirectoriesGetPB
                                          DirNetSearchADAPDirectoriesGetPB;

struct DirNetSearchADAPDirectoriesParsePB {
   AuthDirParamHeader
   Ptr                     getBuffer;
   unsigned long           getBufferSize;
   ForEachADAPDirectory    eachADAPDirectory;
};

typedef struct DirNetSearchADAPDirectoriesParsePB
                                       DirNetSearchADAPDirectoriesParsePB;

typedef pascal Boolean (*ForEachADAPDirectory) (
   long clientData, const DirectoryName *dirName,
   const DirDiscriminator *discriminator, DirGestalt features,
   AddrBlock serverHint);
```

```
struct DirRemoveDirectoryPB {
   AuthDirParamHeader
   CreationID  directoryRecordCID;  /* creation ID for the catalog record */
};

typedef struct DirRemoveDirectoryPB DirRemoveDirectoryPB;

struct DirGetOCESetupRefNumPB {
   AuthDirParamHeader
   CreationID  oceSetupRecordCID;/* creation ID for the catalog record */
};

typedef struct DirGetOCESetupRefNumPB DirGetOCESetupRefNumPB;

struct DirCreatePersonalDirectoryPB {
   AuthDirParamHeader
   FSSpecPtr   fsSpec;     /* FSSpec for the personal catalog */
   OSType      fdType;     /* file type for the personal catalog */
   OSType      fdCreator;  /* file creator for the personal catalog */
};

typedef struct DirCreatePersonalDirectoryPB DirCreatePersonalDirectoryPB;

struct DirOpenPersonalDirectoryPB {
   AuthDirParamHeader
   FSSpecPtr   fsSpec;           /* open an existing personal catalog */
   char        accessRequested;  /* open: permissions requested(byte) */
   char        accessGranted;    /* open: permissions (byte) (granted) */
   DirGestalt  features;         /* features for personal catalog */
};

typedef struct DirOpenPersonalDirectoryPB DirOpenPersonalDirectoryPB;

struct DirClosePersonalDirectoryPB {
   AuthDirParamHeader
};

typedef struct DirClosePersonalDirectoryPB DirClosePersonalDirectoryPB;

struct DirMakePersonalDirectoryRLIPB {
   AuthDirParamHeader
   FSSpecPtr      fromFSSpec;       /* FSSpec for creating relative alias */
   unsigned short pRLIBufferSize;   /* length of 'pRLI' buffer */
   unsigned short pRLISize;         /* length of actual 'pRLI' */
   PackedRLIPtr   pRLI;             /* pRLI for the specified address book */
};
```

```
typedef struct DirMakePersonalDirectoryRLIPB DirMakePersonalDirectoryRLIPB;

struct DirAddRecordPB {
   AuthDirParamHeader
   RecordIDPtr    aRecord;          /* Creation ID returned here */
   Boolean        allowDuplicate;
};

typedef struct DirAddRecordPB DirAddRecordPB;

struct DirDeleteRecordPB {
   AuthDirParamHeader
   RecordIDPtr    aRecord;
};

typedef struct DirDeleteRecordPB DirDeleteRecordPB;

struct DirGetRecordMetaInfoPB {
   AuthDirParamHeader
   RecordIDPtr    aRecord;
   DirMetaInfo    metaInfo;
};

typedef struct DirGetRecordMetaInfoPB DirGetRecordMetaInfoPB;

struct DirGetNameAndTypePB {
   AuthDirParamHeader
   RecordIDPtr    aRecord;
};

typedef struct DirGetNameAndTypePB DirGetNameAndTypePB;

struct DirSetNameAndTypePB {
   AuthDirParamHeader
   RecordIDPtr aRecord;
   Boolean     allowDuplicate;
   Byte        padByte;
   RStringPtr  newName;          /* new name for the record */
   RStringPtr  newType;          /* new type for the record */
};

typedef struct DirSetNameAndTypePB DirSetNameAndTypePB;

struct DirAddPseudonymPB {
   AuthDirParamHeader
   RecordIDPtr aRecord;          /* Record ID to be added to pseudonym */
```

```
   RStringPtr  pseudonymName;    /* new name to be added as pseudonym */
   RStringPtr  pseudonymType;    /* new name to be added as pseudonym */
   Boolean     allowDuplicate;
};

typedef struct DirAddPseudonymPB DirAddPseudonymPB;

struct DirDeletePseudonymPB {
   AuthDirParamHeader
   RecordIDPtr aRecord;       /* Record ID to which pseudonym is
                                 to be added */
   RStringPtr  pseudonymName; /* pseudonym name to be deleted */
   RStringPtr  pseudonymType; /* pseudonym type to be deleted */
};

typedef struct DirDeletePseudonymPB DirDeletePseudonymPB;

struct DirEnumeratePseudonymGetPB {
   AuthDirParamHeader
   RecordIDPtr    aRecord;
   RStringPtr     startingName;
   RStringPtr     startingType;
   long           dReserved;
   long           eReserved;
   long           fReserved;
   long           gReserved;
   long           hReserved;
   Boolean        includeStartingPoint;   /* if true, the pseudonym
                                             specified by starting point will
                                             be included */
   Byte           padByte;
   short          i1Reserved;
   Ptr            getBuffer;
   unsigned long  getBufferSize;
};

typedef struct DirEnumeratePseudonymGetPB DirEnumeratePseudonymGetPB;

struct DirEnumeratePseudonymParsePB {
   AuthDirParamHeader
   RecordIDPtr       aRecord;       /* same as DirEnumerateAliasesGetPB */
   long              bReserved;
   long              cReserved;
   ForEachRecordID   eachRecordID;
   long              eReserved;
```

```
   long              fReserved;
   long              gReserved;
   long              hReserved;
   long              iReserved;
   Ptr               getBuffer;
   unsigned long     getBufferSize;
};

typedef struct DirEnumeratePseudonymParsePB DirEnumeratePseudonymParsePB;

struct DirAddAliasPB {
   AuthDirParamHeader
   RecordIDPtr aRecord;
   Boolean     allowDuplicate;
};

typedef struct DirAddAliasPB DirAddAliasPB;

struct DirAddAttributeValuePB {
   AuthDirParamHeader
   RecordIDPtr    aRecord;
   AttributePtr   attr;    /* AttributeCreationID returned here */
};

typedef struct DirAddAttributeValuePB DirAddAttributeValuePB;

struct DirDeleteAttributeValuePB {
   AuthDirParamHeader
   RecordIDPtr    aRecord;
   AttributePtr   attr;
};

typedef struct DirDeleteAttributeValuePB DirDeleteAttributeValuePB;

struct DirChangeAttributeValuePB {
   AuthDirParamHeader
   RecordIDPtr    aRecord;
   AttributePtr   currentAttr;
   AttributePtr   newAttr;
};
#ifndef __cplusplus
typedef struct DirChangeAttributeValuePB DirChangeAttributeValuePB;
#endif
```

```
struct DirVerifyAttributeValuePB {
   AuthDirParamHeader
   RecordIDPtr    aRecord;
   AttributePtr   attr;
};

typedef struct DirVerifyAttributeValuePB DirVerifyAttributeValuePB;

struct DirFindValuePB {
   AuthDirParamHeader
   PackedRLIPtr      aRLI;               /* an RLI specifying the cluster to
                                            be enumerated */
   LocalRecordIDPtr  aRecord;            /* if not nil, look only in this
                                            record */
   AttributeTypePtr  attrType;           /* if not nil, look only in this
                                            attribute type */
   LocalRecordIDPtr  startingRecord;     /* record in which to start
                                            searching */
   AttributePtr      startingAttribute;  /* attribute in which to start
                                            searching */
   LocalRecordIDPtr  recordFound;        /* record in which data was
                                            found */
   Attribute         attributeFound;     /* attribute in which data was
                                            found */
   unsigned long     matchSize;          /* length of matching bytes */
   Ptr               matchingData;       /* data bytes to be matched in */
                                            search */
   DirSortDirection  sortDirection;      /* sort direction (forward or */
                                            backward) */
};

typedef struct DirFindValuePB DirFindValuePB;

struct DirLookupGetPB {
   AuthDirParamHeader
   RecordIDPtr       *aRecordList;  /* an array of record ID pointers */
   AttributeTypePtr  *attrTypeList; /* an array of attribute types */
   long              cReserved;
   long              dReserved;
   long              eReserved;
   long              fReserved;
   unsigned long     recordIDCount;
   unsigned long     attrTypeCount;
   Boolean           includeStartingPoint;
```

```
                                    /* if true, return the value specified
                                       by the starting indices */
   Byte              padByte;
   short             i1Reserved;
   Ptr               getBuffer;
   unsigned long     getBufferSize;
   unsigned long     startingRecordIndex;
                                    /* start from this record */
   unsigned long     startingAttrTypeIndex;
                                    /* start from this attribute type */
   Attribute         startingAttribute;
                                    /* start from this attribute value */
   long              pReserved;
};

typedef struct DirLookupGetPB DirLookupGetPB;

struct DirLookupParsePB {
   AuthDirParamHeader
   RecordIDPtr *           aRecordList;
                           /* must be same from the corresponding Get call */
   AttributeTypePtr *      attrTypeList;
                           /* must be same from the corresponding Get call */
   long                    cReserved;
   ForEachLookupRecordID   eachRecordID;
   ForEachAttrTypeLookup   eachAttrType;
   ForEachAttrValue        eachAttrValue;
   unsigned long           recordIDCount;
                           /* must be same from the corresponding Get call */
   unsigned long           attrTypeCount;
                           /* must be same from the corresponding Get call */
   long                    iReserved;
   Ptr                     getBuffer;
                           /* must be same from the corresponding Get call */
   unsigned long           getBufferSize;
                           /* must be same from the corresponding Get call */
   unsigned long           lastRecordIndex;
                           /* last record ID processed when parse
                              completed */
   unsigned long           lastAttributeIndex;
                           /* last attribute type processed when parse
                              completed */
   Attribute               lastAttribute;
                           /* last attribute value (with this CreationID)
```

```
                              processed when parse completed */
   unsigned long           attrSize;
                           /* length of the attribute that was not
                              returned */
};

typedef struct DirLookupParsePB DirLookupParsePB;

struct DirDeleteAttributeTypePB {
   AuthDirParamHeader
   RecordIDPtr       aRecord;
   AttributeTypePtr  attrType;
};

typedef struct DirDeleteAttributeTypePB DirDeleteAttributeTypePB;

struct DirEnumerateAttributeTypesGetPB {
   AuthDirParamHeader
   RecordIDPtr       aRecord;
   AttributeTypePtr  startingAttrType;
                                 /* starting point */
   long              cReserved;
   long              dReserved;
   long              eReserved;
   long              fReserved;
   long              gReserved;
   long              hReserved;
   Boolean           includeStartingPoint;
                                 /* if true, return the attrType
                                    specified by starting point */
   Byte              padByte;
   short             i1Reserved;
   Ptr               getBuffer;
   unsigned long     getBufferSize;
};

typedef struct DirEnumerateAttributeTypesGetPB
                                             DirEnumerateAttributeTypesGetPB;

struct DirEnumerateAttributeTypesParsePB {
   AuthDirParamHeader
   RecordIDPtr       aRecord;       /* Same as
                                       DirEnumerateAttributeTypesGetPB */
   long              bReserved;
   long              cReserved;
```

```
   long              dReserved;
   ForEachAttrType   eachAttrType;
   long              fReserved;
   long              gReserved;
   long              hReserved;
   long              iReserved;
   Ptr               getBuffer;
   unsigned long     getBufferSize;
};

typedef struct DirEnumerateAttributeTypesParsePB
                                        DirEnumerateAttributeTypesParsePB;

 struct DirGetDNodeAccessControlGetPB {
   AuthDirParamHeader
   PackedRLIPtr   pRLI;       /* RLI of the cluster whose access control
                                 list is sought */
   long           bReserved;  /* unused */
   long           cReserved;  /* unused */
   long           dReserved;  /* unused */
   long           eResreved;
   Boolean        forCurrentUserOnly;
   DSSpec         *startingPoint;
                              /* starting point */
   Boolean        includeStartingPoint;
                              /* if true, return the DsObject
                                 specified in starting point */
   Ptr            getBuffer;
   unsigned long  getBufferSize;
};

typedef struct DirGetDNodeAccessControlGetPB DirGetDNodeAccessControlGetPB;

struct DirGetDNodeAccessControlParsePB {
   AuthDirParamHeader
   PackedRLIPtr               pRLI;            /* RLI of the cluster  */
   long                       bReserved;       /* unused */
   long                       cReserved;       /* unused */
   long                       dReserved;       /* unused */
   ForEachDNodeAccessControl  eachObject;
   Boolean                    forCurrentUserOnly;
   DSSpec                     *startingPoint;   /* starting point */
   Boolean                    includeStartingPoint;
                                               /* if true, return the
```

```
                                                    record specified in
                                                    starting point */
   Ptr                      getBuffer;
   unsigned long            getBufferSize;

};

typedef struct DirGetDNodeAccessControlParsePB
                                        DirGetDNodeAccessControlParsePB;

struct DirGetRecordAccessControlGetPB {
   AuthDirParamHeader
   RecordIDPtr       aRecord;          /* ID of record whose access
                                          control list is sought */
   long              bReserved;        /* unused */
   long              cReserved;        /* unused */
   long              dReserved;        /* unused */
   long              eReserved;
   Boolean           forCurrentUserOnly;
   DSSpec            *startingPoint;   /* starting point */
   Boolean            includeStartingPoint;
                                       /* if true, return the DsObject
                                          specified in starting point */

   Ptr               getBuffer;
   unsigned long     getBufferSize;
};

typedef struct DirGetRecordAccessControlGetPB DirGetRecordAccessControlGetPB;

struct DirGetRecordAccessControlParsePB {
   AuthDirParamHeader
   RecordIDPtr               aRecord;    /* ID of record to which access
                                            control list is sought */
   long                      bReserved;  /* unused */
   long                      cReserved;  /* unused */
   long                      dReserved;  /* unused */
   ForEachRecordAccessControl eachObject;
   Boolean                   forCurrentUserOnly;
   DSSpec                    *startingPoint;
                                         /* starting point */
   Boolean                   includeStartingPoint;
                                         /* if true return the record
                                            specified in starting point */
```

```
   Ptr                         getBuffer;
   unsigned long               getBufferSize;
};

typedef struct DirGetRecordAccessControlParsePB
                                          DirGetRecordAccessControlParsePB;

struct DirGetAttributeAccessControlGetPB {
   AuthDirParamHeader
   RecordIDPtr       aRecord;       /* ID of record to which access control
                                       list is sought */
   AttributeTypePtr  aType;         /* attribute type to which access
                                       controls are sought */
   long              cReserved;     /* unused */
   long              dReserved;     /* unused */
   long              eResrved;
   Boolean           forCurrentUserOnly;

   DSSpec            *startingPoint;
                                    /* starting point */
   Boolean           includeStartingPoint;
                                    /* if true return the DsObject */
                                    /* specified in starting point */
   Ptr               getBuffer;
   unsigned longgetBufferSize;
};

typedef struct DirGetAttributeAccessControlGetPB
                                          DirGetAttributeAccessControlGetPB;

struct DirGetAttributeAccessControlParsePB {
   AuthDirParamHeader
   RecordIDPtr                   aRecord;    /* ID of record to which access
                                                control list is sought */
   AttributeTypePtr              aType;      /* attribute type to which
                                                access controls are sought */
   long                          cReserved;  /* unused */
   long                          dReserved;  /* unused */
   ForEachAttributeAccessControl eachObject;
   Boolean                       forCurrentUserOnly;
   DSSpec                        *startingPoint;
                                             /* starting point */
   Boolean                       includeStartingPoint;
                                          /* if true, return the record
```

```
                                          specified in starting point */
   Ptr                          getBuffer;
   unsigned long                getBufferSize;
};

typedef struct DirGetAttributeAccessControlParsePB
                                     DirGetAttributeAccessControlParsePB;

struct DirAbortPB {
   AuthDirParamHeader
   union DirParamBlock  *pb;/* parameter block for the call that must
                               be aborted */
};

typedef struct DirAbortPB            DirAbortPB;

typedef union AuthParamBlock         AuthParamBlock;

typedef AuthParamBlock               *AuthParamBlockPtr;

union DirParamBlock {
   struct {AuthDirParamHeader}       header;
   DirAddRecordPB                    addRecordPB;
   DirDeleteRecordPB                 deleteRecordPB;
   DirEnumerateGetPB                 enumerateGetPB;
   DirEnumerateParsePB               enumerateParsePB;
   DirFindRecordGetPB                findRecordGetPB;
   DirFindRecordParsePB              findRecordParsePB;
   DirLookupGetPB                    lookupGetPB;
   DirLookupParsePB                  lookupParsePB;
   DirAddAttributeValuePB            addAttributeValuePB;
   DirDeleteAttributeTypePB          deleteAttributeTypePB;
   DirDeleteAttributeValuePB         deleteAttributeValuePB;
   DirChangeAttributeValuePB         changeAttributeValuePB;
   DirVerifyAttributeValuePB         verifyAttributeValuePB;
   DirFindValuePB                    findValuePB;
   DirEnumeratePseudonymGetPB        enumeratePseudonymGetPB;
   DirEnumeratePseudonymParsePB      enumeratePseudonymParsePB;
   DirAddPseudonymPB                 addPseudonymPB;
   DirDeletePseudonymPB              deletePseudonymPB;
   DirAddAliasPB                     addAliasPB;
   DirEnumerateAttributeTypesGetPB   enumerateAttributeTypesGetPB;
   DirEnumerateAttributeTypesParsePB enumerateAttributeTypesParsePB;
   DirGetNameAndTypePB               getNameAndTypePB;
   DirSetNameAndTypePB               setNameAndTypePB;
```

```
   DirGetRecordMetaInfoPB              getRecordMetaInfoPB;
   DirGetDNodeMetaInfoPB               getDNodeMetaInfoPB;
   DirGetDirectoryInfoPB               getDirectoryInfoPB;

   DirGetDNodeAccessControlGetPB       getDNodeAccessControlGetPB;
   DirGetDNodeAccessControlParsePB     getDNodeAccessControlParsePB;

   DirGetRecordAccessControlGetPB      getRecordAccessControlGetPB;
   DirGetRecordAccessControlParsePB    getRecordAccessControlParsePB;

   DirGetAttributeAccessControlGetPB   getAttributeAccessControlGetPB;
   DirGetAttributeAccessControlParsePB getAttributeAccessControlParsePB;

   DirEnumerateDirectoriesGetPB        enumerateDirectoriesGetPB;

   DirEnumerateDirectoriesParsePB      enumerateDirectoriesParsePB;
   DirAddADAPDirectoryPB               addADAPDirectoryPB;
   DirRemoveDirectoryPB                removeDirectoryPB;
   DirNetSearchADAPDirectoriesGetPB    netSearchADAPDirectoriesGetPB;
   DirNetSearchADAPDirectoriesParsePB  netSearchADAPDirectoriesParsePB;
   DirFindADAPDirectoryByNetSearchPB   findADAPDirectoryByNetSearchPB;
   DirMapDNodeNumberToPathNamePB       mapDNodeNumberToPathNamePB;
   DirMapPathNameToDNodeNumberPB       mapPathNameToDNodeNumberPB;
   DirGetLocalNetworkSpecPB            getLocalNetworkSpecPB;
   DirGetDNodeInfoPB                   getDNodeInfoPB;

/* calls for personal catalogs */
   DirCreatePersonalDirectoryPB        createPersonalDirectoryPB;
   DirOpenPersonalDirectoryPB          openPersonalDirectoryPB;
   DirClosePersonalDirectoryPB         closePersonalDirectoryPB;
   DirMakePersonalDirectoryRLIPB       makePersonalDirectoryRLIPB;

/* calls for CSAM's */

   DirAddDSAMPB                        addDSAMPB;
   DirInstantiateDSAMPB                instantiateDSAMPB;
   DirRemoveDSAMPB                     removeDSAMPB;
   DirAddDSAMDirectoryPB               addDSAMDirectoryPB;
   DirGetExtendedDirectoriesInfoPB     getExtendedDirectoriesInfoPB;
   DirGetDirectoryIconPB               getDirectoryIconPB;

/* call to dsRefNum for system(PowerTalk Setup) personal catalog */
   DirGetOCESetupRefNumPB              dirGetOCESetupRefNumPB;
```

```
/* abort a aysnchronous call */
   DirAbortPB                          abortPB;
};

typedef union DirParamBlock            DirParamBlock;
typedef DirParamBlock                  *DirParamBlockPtr;
```

Catalog Manager Functions

Getting Information About Catalogs

```
pascal OSErr DirEnumerateDirectoriesGet
                            (DirParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr DirEnumerateDirectoriesParse
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirFindRecordGet
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirFindRecordParse
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetDirectoryInfo
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetLocalNetworkSpec
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetDirectoryIcon
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetExtendedDirectoriesInfo
                            (DirParamBlockPtr paramBlock,Boolean async);
```

Getting Information About DNodes

```
pascal OSErr DirEnumerateGet
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirEnumerateParse
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetDNodeMetaInfo
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirMapDNodeNumberToPathName
                            (DirParamBlockPtrparamBlock,Boolean async);
pascal OSErr DirMapPathNameToDNodeNumber
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetDNodeInfo
                            (DirParamBlockPtr paramBlock,Boolean async);
```

Maintaining the PowerTalk Setup Catalog

```
pascal OSErr DirAddADAPDirectory
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirFindADAPDirectoryByNetSearch
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr irNetSearchADAPDirectoriesGet
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirNetSearchADAPDirectoriesParse
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirRemoveDirectory
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetOCESetupRefnum
                            (DirParamBlockPtr paramBlock,Boolean async);
```

Creating, Opening, and Closing Personal Catalogs

```
pascal OSErr DirCreatePersonalDirectory
                            (DirParamBlockPtr paramBlock);
pascal OSErr DirOpenPersonalDirectory
                            (DirParamBlockPtr paramBlock);
pascal OSErr DirClosePersonalDirectory
                            (DirParamBlockPtr paramBlock);
pascal OSErr DirMakePersonalDirectoryRLI
                            (DirParamBlockPtr paramBlock);
```

Managing Records

```
pascal OSErr DirAddRecord   (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirDeleteRecord
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetRecordMetaInfo
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetNameAndType
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirSetNameAndType
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirAddPseudonym
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirDeletePseudonym
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirEnumeratePseudonymGet
                            (DirParamBlockPtr paramBlock,Boolean async);
```

```
pascal OSErr DirEnumeratePseudonymParse
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirAddAlias    (DirParamBlockPtr paramBlock,Boolean async);
```

Managing Attribute Types and Values

```
pascal OSErr DirAddAttributeValue
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirDeleteAttributeValue
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirChangeAttributeValue
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirVerifyAttributeValue
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirFindValue   (DirParamBlockPtr paramBlock, Boolean async);
pascal OSErr DirLookupGet   (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirLookupParse
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirDeleteAttributeType
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirEnumerateAttributeTypesGet
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirEnumerateAttributeTypesParse
                            (DirParamBlockPtr paramBlock,Boolean async);
```

Reading Access Controls for dNodes, Records, and Attribute Types

```
pascal DSSpec *OCEGetAccessControlDSSpec
                            (const CategoryMask categoryBitMask);
pascal OSErr DirGetDNodeAccessControlGet
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetDNodeAccessControlParse
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetRecordAccessControlGet
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetRecordAccessControlParse
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetAttributeAccessControlGet
                            (DirParamBlockPtr paramBlock,Boolean async);
pascal OSErr DirGetAttributeAccessControlParse
                            (DirParamBlockPtr paramBlock,Boolean async);
```

Canceling a Catalog Manager Function

```
pascal OSErr DirAbort       (DirParamBlockPtr paramBlock);
```

Application-Defined Functions

```
void MyCompletionRoutine    (DirParamBlockPtr paramBlk);
pascal Boolean MyForEachRecordID
                            (long clientData,const RecordID *recordID);
pascal Boolean MyForEachAttrType
                            (long clientData,
                             const AttributeType *attrType);
pascal Boolean MyForEachDirectory
                            (long clientData,
                             const DirectoryName *dirName,
                             const DirDiscriminator *discriminator,
                             DirGestalt features);
pascal Boolean MyForEachLookupRecordID
                            (long clientData,const RecordID *recordID);
pascal Boolean MyForEachAttrTypeLookup
                            (long clientData,
                             const AttributeType *attrType,
                             AccessMask myAttrAccMask);
pascal Boolean MyForEachAttrValue
                            (long clientData, const Attribute *attribute);
pascal Boolean MyForEachDirEnumSpec
                            (long clientData, const DirEnumSpec *enumSpec);
pascal Boolean MyForEachRecord
                            (long clientData,
                             const DirEnumSpec *enumSpec,
                             pRLI PackedRLIPtr);
pascal Boolean MyForEachADAPDirectory
                            (long clientData,
                             const DirectoryName *directoryName,
                             const DirDiscriminator *discriminator,
                             DirGestalt features, AddrBlock serverHint);
pascal Boolean MyForEachDNodeAccessControl
                            (long clientData, const DSSpec *dsObj,
                             AccessMask activeDnodeAccMask,
                             AccessMask defaultRecordAccMask,
                             AccessMask defaultAttributeAccMask);
```

```
pascal Boolean MyForEachRecordAccessControl
                            (long clientData,const DSSpec *dsObj,
                             AccessMask activeDnodeAccMask,
                             AccessMask activeRecordAccMask,
                             AccessMask defaultAttributeAccMask);
pascal Boolean MyForEachAttributeAccessControl
                            (long clientData,const DSSpec *dsObj,
                             AccessMask activeDnodeAccMask,
                             AccessMask activeRecordAccMask,
                             AccessMask activeAttributeAccMask);
```

Pascal Summary

Constants and Data Types

```
CONST
   {access categories bit numbers}
   kThisRecordOwnerBit           = 0;
   kFriendsBit                   = 1;
   kAuthenticatedInDNodeBit      = 2;
   kAuthenticatedInDirectoryBit  = 3;
   kGuestBit                     = 4;
   kMeBit                        = 5;

   {values of CategoryMask}
   kThisRecordOwnerMask          = $00000001;  {1<<kThisRecordOwnerBit}
   kFriendsMask                  = $00000002;  {1<<kFriendsBit}
   kAuthenticatedInDNodeMask     = $00000004;  {1<<kAuthenticatedInDNodeBit}
   kAuthenticatedInDirectoryMask = $00000008;
                                          {1<<kAuthenticatedInDirectoryBit}
   kGuestMask                    = $00000010;  {1<<kGuestBit}
   kMeMask                       = $00000020;  {1<<kMeBit}

   kEnumDistinguishedNameBit  = 0;
   kEnumAliasBit              = 1;
   kEnumPseudonymBit          = 2;
   kEnumDNodeBit              = 3;
   kEnumInvisibleBit          = 4;

   {values of DirEnumChoices}
   kEnumDistinguishedNameMask = $00000001;  {1<<kEnumDistinguishedNameBit}
   kEnumAliasMask             = $00000002;  {1<<kEnumAliasBit}
```

```
kEnumPseudonymMask          = $00000004;  {1<<kEnumPseudonymBit}
kEnumDNodeMask              = $00000008;  {1<<kEnumDNodeBit}
kEnumInvisibleMask          = $00000010;  {1<<kEnumInvisibleBit}

kEnumAllMask = (kEnumDistinguishedNameMask + kEnumAliasMask +
                kEnumPseudonymMask + kEnumDNodeMask + kEnumInvisibleMask);

{Values of DirSortOption}
kSortByName                                 = 0;
kSortByType                                 = 1;

{values of DirSortDirection}
kSortForwards                               = 0;
kSortBackwards                              = 1;

{values of DirMatchWith}
kMatchAll                                   = 0;
kExactMatch                                 = 1;
kBeginsWith                                 = 2;
kEndingWith                                 = 3;
kContaining                                 = 4;

kCurrentOCESortVersion                      = 1;

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;

{ values of DirGestalt }
kSupportsDNodeNumberMask                    = $00000001;
                            {1<<kSupportsDNodeNumberBit}
kSupportsRecordCreationIDMask               = $00000002;
                            {1<<kSupportsRecordCreationIDBit}
kSupportsAttributeCreationIDMask            = $00000004;
                            {1<<kSupportsAttributeCreationIDBit}
kSupportsMatchAllMask                       = $00000008;
                            {1<<kSupportsMatchAllBit}
kSupportsBeginsWithMask                     = $00000010;
                            {1<<kSupportsBeginsWithBit}
kSupportsExactMatchMask                     = $00000020;
                            {1<<kSupportsExactMatchBit}
kSupportsEndsWithMask                       = $00000040;
                            {1<<kSupportsEndsWithBit}
kSupportsContainsMask                       = $00000080;
                            {1<<kSupportsContainsBit}
kSupportsOrderedEnumerationMask             = $00000100;
                            {1<<kSupportsOrderedEnumerationBit}
kCanSupportNameOrderMask                    = $00000200;
                            {1<<kCanSupportNameOrderBit}
kCanSupportTypeOrderMask                    = $00000400;
                            {1<<kCanSupportTypeOrderBit}
kSupportSortBackwardsMask                   = $00000800;
                            {1<<kSupportSortBackwardsBit}
kSupportIndexRatioMask                      = $00001000;
                            {1<<kSupportIndexRatioBit}
kSupportsEnumerationContinueMask            = $00002000;
                            {1<<kSupportsEnumerationContinueBit}
kSupportsLookupContinueMask                 = $00004000;
                            {1<<kSupportsLookupContinueBit}
kSupportsEnumerateAttributeTypeContinueMask  = $00008000;
                            {1<<kSupportsEnumerateAttributeTypeContinueBit}
kSupportsEnumeratePseudonymContinueMask     = $00010000;
                            {1<<kSupportsEnumeratePseudonymContinueBit}
kSupportsAliasesMask                        = $00020000;
                            {1<<kSupportsAliasesBit}
kSupportsPseudonymsMask                     = $00040000;
                            {1<<kSupportsPseudonymsBit}
kSupportsPartialPathNamesMask               = $00080000;
```

```
                              {1<<kSupportsPartialPathNamesBit}
   kSupportsAuthenticationMask                  = $00100000;
                              {1<<kSupportsAuthenticationBit}
   kSupportsProxiesMask                         = $00200000;
                              {1<<kSupportsProxiesBit}
   kSupportsFindRecordMask                      = $00400000;
                              {1<<kSupportsFindRecordBit}

TYPE
   DirEnumChoices                   = LONGINT;
   DirMatchWith                     = BYTE;
   DirSortDirection                 = INTEGER;
   ForMyEachRecordID                = ProcPtr;
   ForMyEachLookupRecordID          = ProcPtr;
   ForMyEachAttrTypeLookup          = ProcPtr;
   ForMyEachAttrValue               = ProcPtr;
   ForMyEachAttrType                = ProcPtr;
   ForMyEachRecordID                = ProcPtr;
   ForMyEachDNodeAccessControl      = ProcPtr;
   ForMyEachRecordAccessControl     = ProcPtr;
   ForMyEachAttributeAccessControl  = ProcPtr;
   ForMyEachDirEnumSpec             = ProcPtr;
   ForMyEachDirectory               = ProcPtr;
   ForMyEachADAPDirectory           = ProcPtr;

DNodeID = RECORD
      dNodeNumber:   DNodeNum;      {dNode number}
      reserved1:     LONGINT;
      name:          RStringPtr;
      reserved2:     LONGINT;
   END;

   DirEnumSpec = RECORD
      enumFlag:      DirEnumChoices;
      indexRatio:    INTEGER;       {if supported, approx Record Position
                                      between 1 and 100; 0 If not supported}
      CASE INTEGER OF
         1: (recordIdentifier:   LocalRecordID);
         2: (dNodeIdentifier:    DNodeID);
   END;

   DirMetaInfo = RECORD
      info: ARRAY[1..4] OF LONGINT;
   END;
```

```
   SLRV = RECORD
      script:     ScriptCode;    {script code in which entries are sorted}
      language:   INTEGER;       {language code in which entries are sorted}
      regionCode: INTEGER;       {region code in which entries are sorted}
      version:    INTEGER;       {version of AOCE sorting software }
   END;

   AuthDirParamHeader = RECORD
      qLink:         Ptr;
      reserved1:     LONGINT;
      reserved2:     LONGINT;
      ioCompletion:  ProcPtr;
      ioResult:      OSErr;
      saveA5:        LONGINT;
      reqCode:       INTEGER;
      reserved:      ARRAY[1..2] OF LONGINT;
      serverHint:    AddrBlock;
      dsRefNum:      INTEGER;
      callID:        LONGINT;
      identity:      AuthIdentity;
      gReserved1:    LONGINT;
      gReserved2:    LONGINT;
      gReserved3:    LONGINT;
      clientData:    LONGINT;
   END;

{Catalog types and operations}
   AuthIdentity         = LONGINT;        {unique identifier for an identity}
   LocalIdentity        = AuthIdentity;   {umbrella localIdentity}

   DirEnumerateDirectoriesGetPB = PACKED RECORD
      qLink:                    Ptr;
      reserved1:                LONGINT;
      reserved2:                LONGINT;
      ioCompletion:             ProcPtr;
      ioResult:                 OSErr;
      saveA5:                   LONGINT;
      reqCode:                  INTEGER;
      reserved:                 ARRAY[1..2] OF LONGINT;
      serverHint:               AddrBlock;
      dsRefNum:                 INTEGER;
      callID:                   LONGINT;
      identity:                 AuthIdentity;
      gReserved1:               LONGINT;
```

```
   gReserved2:                LONGINT;
   gReserved3:                LONGINT;
   clientData:                LONGINT;
   directoryKind:             OCEDirectoryKind; {enumerate catalogs
                                                 bearing this signature}
   startingDirectoryName:     DirectoryNamePtr; {starting catalog name}
   startingDirDiscriminator:  DirDiscriminator; {starting catalog
                                                 discriminator}
   eReserved:                 LONGINT;
   fReserved:                 LONGINT;
   gReserved:                 LONGINT;
   hReserved:                 LONGINT;
   includeStartingPoint:      BOOLEAN;          {if true, return catalog
                                                specified by starting
                                                point}
   padByte:                   Byte;
   i1Reserved:                INTEGER;
   getBuffer:                 Ptr;
   getBufferSize:             LONGINT;
END;

DirEnumerateDirectoriesParsePB = RECORD
   qLink:                     Ptr;
   reserved1:                 LONGINT;
   reserved2:                 LONGINT;
   ioCompletion:              ProcPtr;
   ioResult:                  OSErr;
   saveA5:                    LONGINT;
   reqCode:                   INTEGER;
   reserved:                  ARRAY[1..2] OF LONGINT;
   serverHint:                AddrBlock;
   dsRefNum:                  INTEGER;
   callID:                    LONGINT;
   identity:                  AuthIdentity;
   gReserved1:                LONGINT;
   gReserved2:                LONGINT;
   gReserved3:                LONGINT;
   clientData:                LONGINT;
   aReserved:                 LONGINT;
   bReserved:                 LONGINT;
   cReserved:                 LONGINT;
   dReserved:                 LONGINT;
   eachDirectory:             ForEachDirectory;
   fReserved:                 LONGINT;
```

```
   gReserved:                 LONGINT;
   hReserved:                 LONGINT;
   iReserved:                 LONGINT;
   getBuffer:                 Ptr;
   getBufferSize:             LONGINT;
END;

DirFindRecordGetPB = RECORD

   qLink:                     Ptr;
   reserved1:                 LONGINT;
   reserved2:                 LONGINT;
   ioCompletion:              ProcPtr;
   ioResult:                  OSErr;
   saveA5:                    LONGINT;
   reqCode:                   INTEGER;
   reserved:                  ARRAY[1..2] OF LONGINT;
   serverHint:                AddrBlock;
   dsRefNum:                  INTEGER;
   callID:                    LONGINT;
   identity:                  AuthIdentity;
   gReserved1:                LONGINT;
   gReserved2:                LONGINT;
   gReserved3:                LONGINT;
   clientData:                LONGINT;
   startingPoint:             RecordIDPtr;
   reservedA:                 ARRAY[1..2] OF LONGINT;
   nameMatchString:           RStringPtr;
   typesList:                 ^RStringPtr;
   typeCount:                 LONGINT;
   reservedB:                 LONGINT;
   reservedC:                 INTEGER;
   matchNameHow:              DirMatchWith;
   matchTypeHow:              DirMatchWith;
   getBuffer:                 Ptr;
   getBufferSize:             LONGINT;
   directoryName:             DirectoryNamePtr;
   discriminator:             DirDiscriminator;
END;

DirFindRecordParsePB = RECORD
   qLink:                     Ptr;
   reserved1:                 LONGINT;
   reserved2:                 LONGINT;
```

```
   ioCompletion:              ProcPtr;
   ioResult:                  OSErr;
   saveA5:                    LONGINT;
   reqCode:                   INTEGER;
   reserved:                  ARRAY[1..2] OF LONGINT;
   serverHint:                AddrBlock;
   dsRefNum:                  INTEGER;
   callID:                    LONGINT;
   identity:                  AuthIdentity;
   gReserved1:                LONGINT;
   gReserved2:                LONGINT;
   gReserved3:                LONGINT;
   clientData:                LONGINT;
   startingPoint:             RecordIDPtr;
   reservedA:                 ARRAY[1..2] OF LONGINT;
   nameMatchString:           RStringPtr;
   typesList:                 ^RStringPtr;
   typeCount:                 LONGINT;
   reservedB:                 LONGINT;
   reservedC:                 INTEGER;
   matchNameHow:              DirMatchWith;
   matchTypeHow:              DirMatchWith;
   getBuffer:                 Ptr;
   getBufferSize:             LONGINT;
   directoryName:             DirectoryNamePtr;
   discriminator:             DirDiscriminator;
   forEachRecordFunc:         ForEachRecord;
END;

DirGetDirectoryInfoPB = RECORD
   qLink:                     Ptr;
   reserved1:                 LONGINT;
   reserved2:                 LONGINT;
   ioCompletion:              ProcPtr;
   ioResult:                  OSErr;
   saveA5:                    LONGINT;
   reqCode:                   INTEGER;
   reserved:                  ARRAY[1..2] OF LONGINT;
   serverHint:                AddrBlock;
   dsRefNum:                  INTEGER;
   callID:                    LONGINT;
   identity:                  AuthIdentity;
   gReserved1:                LONGINT;
   gReserved2:                LONGINT;
```

```
      gReserved3:               LONGINT;
      clientData:               LONGINT;
      directoryName:            DirectoryNamePtr; {catalog name}
      discriminator:            DirDiscriminator; discriminate between
                                                  duplicate catalog
                                                  names}
      features:                 DirGestalt;       {capability bit flags}
   END;

DirGetLocalNetworkSpecPB = RECORD
      qLink:                    Ptr;
      reserved1:                LONGINT;
      reserved2:                LONGINT;
      ioCompletion:             ProcPtr;
      ioResult:                 OSErr;
      saveA5:                   LONGINT;
      reqCode:                  INTEGER;
      reserved:                 ARRAY[1..2] OF LONGINT;
      serverHint:               AddrBlock;
      dsRefNum:                 INTEGER;
      callID:                   LONGINT;
      identity:                 AuthIdentity;
      gReserved1:               LONGINT;
      gReserved2:               LONGINT;
      gReserved3:               LONGINT;
      clientData:               LONGINT;
      directoryName:            DirectoryNamePtr;    {catalog name}
      discriminator:            DirDiscriminator;    {discriminator}
      networkSpec:              NetworkSpecPtr;      {NetworkSpec}

   END;

DirGetDirectoryIconPB = RECORD
      qLink:                    Ptr;
      reserved1:                LONGINT;
      reserved2:                LONGINT;
      ioCompletion:             ProcPtr;
      ioResult:                 OSErr;
      saveA5:                   LONGINT;
      reqCode:                  INTEGER;
      reserved:                 ARRAY[1..2] OF LONGINT;
      serverHint:               AddrBlock;
      dsRefNum:                 INTEGER;
      callID:                   LONGINT;
```

```
      identity:                 AuthIdentity;
      gReserved1:               LONGINT;
      gReserved2:               LONGINT;
      gReserved3:               LONGINT;
      clientData:               LONGINT;
      pRLI:                     PackedRLIPtr;  {packed RLI for the catalog}
      iconType:                 OSType;        {type of Icon requested}
      iconBuffer:               Ptr;           {buffer to hold Icon Data}
      bufferSize:               LONGINT;       {size of buffer to hold icon
                                                 data}
      END;

DirGetExtendedDirectoriesInfoPB = RECORD
      qLink:                    Ptr;
      reserved1:                LONGINT;
      reserved2:                LONGINT;
      ioCompletion:             ProcPtr;
      ioResult:                 OSErr;
      saveA5:                   LONGINT;
      reqCode:                  INTEGER;
      reserved:                 ARRAY[1..2] OF LONGINT;
      serverHint:               AddrBlock;
      dsRefNum:                 INTEGER;
      callID:                   LONGINT;
      identity:                 AuthIdentity;
      gReserved1:               LONGINT;
      gReserved2:               LONGINT;
      gReserved3:               LONGINT;
      clientData:               LONGINT;
      buffer:                   Ptr;           {Pointer to a buufer
                                                 where data is returned}
      bufferSize:               LONGINT;       {Length of buffer in which
                                               actual data is returned}
      totalEntries:             LONGINT;    {total number of catalogs found}
      actualEntries:            LONGINT;    {total number of catalog
                                             entries returned}
      END;

   ForEachDirectory = ProcPtr;
      {FUNCTION ForEachDirectory(clientData: long; dirName: DirectoryNamePtr;
      discriminator: DirDiscriminator; features: DirGestalt): BOOLEAN;}
```

```
DirEnumerateGetPB = PACKED RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
   gReserved1:          LONGINT;
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
   clientData:          LONGINT;
   aRLI:                PackedRLIPtr;     {an RLI specifying the cluster
                                          to be enumerated}
   startingPoint:       ^DirEnumSpec;
   sortBy:              DirSortOption;
   sortDirection:       DirSortDirection;
   dReserved:           LONGINT;
   nameMatchString:     RStringPtr;       {name from which enumeration
                                          should start}
   typesList:           ^RStringPtr;      {list of entity types to be
                                          enumerated}
   typeCount:           LONGINT;          {number of types in the list}
   enumFlags:           DirEnumChoices;   {indicates what to enumerate}
   includeStartingPoint:BOOLEAN;          {if true return the record
                                           specified in starting point}
   padByte:             Byte;
   matchNameHow:        DirMatchWith;     {matching Criteria}
                                          {for nameMatchString}
   matchTypeHow:        DirMatchWith;     {matching criteria for typeList}
   getBuffer:           Ptr;
   getBufferSize:       LONGINT;
   responseSLRV:        SLRV;             {response SLRV}
END;

DirEnumerateParsePB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
```

```
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   aRLI:          PackedRLIPtr;          {an RLI specifying the cluster
                                          to be enumerated}
   bReserved:     LONGINT;
   cReserved:     LONGINT;
   eachEnumSpec:  ForEachDirEnumSpec;
   eReserved:     LONGINT;
   fReserved:     LONGINT;
   gReserved:     LONGINT;
   hReserved:     LONGINT;
   iReserved:     LONGINT;
   getBuffer:     Ptr;
   getBufferSize: LONGINT;
   l1Reserved:    INTEGER;
   l2Reserved:    INTEGER;
   l3Reserved:    INTEGER;
   l4Reserved:    INTEGER;
END;

DirGetDNodeMetaInfoPB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
```

```
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   pRLI:          PackedRLIPtr;
   metaInfo:      DirMetaInfo;
END;

DirMapDNodeNumberToPathNamePB = RECORD
   qLink:            Ptr;
   reserved1:        LONGINT;
   reserved2:        LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   reserved:         ARRAY[1..2] OF LONGINT;
   serverHint:       AddrBlock;
   dsRefNum:         INTEGER;
   callID:           LONGINT;
   identity:         AuthIdentity;
   gReserved1:       LONGINT;
   gReserved2:       LONGINT;
   gReserved3:       LONGINT;
   clientData:       LONGINT;
   directoryName:    DirectoryNamePtr;       {catalog name}
   discriminator:    DirDiscriminator;       {discriminator}
   dNodeNumber:      DNodeNum;               {dNodenumber to be mapped}
   path:             PackedPathNamePtr;      {packed path name returned}
   lengthOfPathName: INTEGER;                {length of packed pathname
                                              structure}
END;

DirMapPathNameToDNodeNumberPB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
```

```
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   directoryName: DirectoryNamePtr;          {catalog name}
   discriminator: DirDiscriminator;          {discriminator}
   dNodeNumber:   DNodeNum;                  {dNode number to the path}
   path:          PackedPathNamePtr;         {pathname to be mapped}
END;

DirGetDNodeInfoPB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   pRLI:          PackedRLIPtr;     {packed RLI whose info is requested}
   descriptor:    DirNodeKind;      {dNode descriptor}
   networkSpec:   NetworkSpecPtr;   {cluster's networkSpec if kIsCluster}
END;

DirAddADAPDirectoryPB = PACKED RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
```

```
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
   gReserved1:          LONGINT;
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
   clientData:          LONGINT;
   directoryName:       DirectoryNamePtr;    {catalog name}
   discriminator:       DirDiscriminator;    {discriminate between}
                                             duplicate catalog names}
   addToOCESetup:       BOOLEAN;             {add this catalog to
                                              PowerTalk setup}
   padByte:             Byte;
   directoryRecordCID:  CreationID;          {creation ID for the
                                             catalog record}
END;

DirFindADAPDirectoryByNetSearchPB = PACKED RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
   gReserved1:          LONGINT;
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
   clientData:          LONGINT;
   directoryName:       DirectoryNamePtr; {catalog name}
   discriminator:       DirDiscriminator; {discriminate between
                                           duplicate names}
   addToOCESetup:       BOOLEAN;          {add this catalog to PowerTalk
                                          Setup list}
   padByte:             Byte;
   directoryRecordCID:  CreationID;       {creation ID for the catalog
                                           record}
END;
```

```
   DirNetSearchADAPDirectoriesGetPB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   getBuffer:     Ptr;
   getBufferSize: LONGINT;
   cReserved:     LONGINT;
END;

DirNetSearchADAPDirectoriesParsePB = RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
   gReserved1:          LONGINT;
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
   clientData:          LONGINT;
   getBuffer:           Ptr;
   getBufferSize:       LONGINT;
   eachADAPDirectory:   ForEachADAPDirectory;
END;
```

```
ForEachADAPDirectory = ProcPtr;
   {FUNCTION ForEachADAPDirectory(
      clientData: long;
      dirName: DirectoryNamePtr;
      discriminator: DirDiscriminator;
      features: DirGestalt;
      serverHint: AddrBlock): BOOLEAN;}

DirRemoveDirectoryPB = RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
   gReserved1:          LONGINT;
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
   clientData:          LONGINT;
   directoryRecordCID:  CreationID; {creation ID for the catalog record}
END;

DirRemoveDirectoryPB = RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
   gReserved1:          LONGINT;
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
```

```
   clientData:          LONGINT;
   directoryRecordCID:  CreationID; {creation ID for the catalog record}
END;

DirGetOCESetupRefNumPB = RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
   gReserved1:          LONGINT;
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
   clientData:          LONGINT;
   oceSetupRecordCID:   CreationID; {creation ID for the catalog record}
END;

DirCreatePersonalDirectoryPB = RECORD
   qLink:          Ptr;
   reserved1:      LONGINT;
   reserved2:      LONGINT;
   ioCompletion:   ProcPtr;
   ioResult:       OSErr;
   saveA5:         LONGINT;
   reqCode:        INTEGER;
   reserved:       ARRAY[1..2] OF LONGINT;
   serverHint:     AddrBlock;
   dsRefNum:       INTEGER;
   callID:         LONGINT;
   identity:       AuthIdentity;
   gReserved1:     LONGINT;
   gReserved2:     LONGINT;
   gReserved3:     LONGINT;
   clientData:     LONGINT;
   fsSpec:         FSSpecPtr;     {FSSpec for the personal catalog}
```

```
   fdType:        OSType;        {file type for the personal catalog}
   fdCreator:     OSType;        {file creator for the personal catalog}
END;

DirOpenPersonalDirectoryPB = PACKED RECORD
   qLink:            Ptr;
   reserved1:        LONGINT;
   reserved2:        LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   reserved:         ARRAY[1..2] OF LONGINT;
   serverHint:       AddrBlock;
   dsRefNum:         INTEGER;
   callID:           LONGINT;
   identity:         AuthIdentity;
   gReserved1:       LONGINT;
   gReserved2:       LONGINT;
   gReserved3:       LONGINT;
   clientData:       LONGINT;
   fsSpec:           FSSpecPtr;     {open an existing personal catalog}
   accessRequested:  Char;          {open: permissions requested (byte)}
   accessGranted:    Char;          {open: permissions (byte) (Granted)
   features:         DirGestalt;    {features for personal catalog}
END;

DirClosePersonalDirectoryPB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
```

```
   gReserved3:    LONGINT;
   clientData:    LONGINT;
END;

DirMakePersonalDirectoryRLIPB = RECORD
   qLink:            Ptr;
   reserved1:        LONGINT;
   reserved2:        LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   reserved:         ARRAY[1..2] OF LONGINT;
   serverHint:       AddrBlock;
   dsRefNum:         INTEGER;
   callID:           LONGINT;
   identity:         AuthIdentity;
   gReserved1:       LONGINT;
   gReserved2:       LONGINT;
   gReserved3:       LONGINT;
   clientData:       LONGINT;
   fromFSSpec:       FSSpecPtr;     {FSSpec for creating relative alias}
   pRLIBufferSize:   INTEGER;       {length of 'pRLI' buffer}
   pRLISize:         INTEGER;       {length of actual 'pRLI'}
   pRLI:             PackedRLIPtr;  {pRLI for the specified address book}
END;

DirAddRecordPB = RECORD
   qLink:            Ptr;
   reserved1:        LONGINT;
   reserved2:        LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   reserved:         ARRAY[1..2] OF LONGINT;
   serverHint:       AddrBlock;
   dsRefNum:         INTEGER;
   callID:           LONGINT;
   identity:         AuthIdentity;
   gReserved1:       LONGINT;
   gReserved2:       LONGINT;
   gReserved3:       LONGINT;
   clientData:       LONGINT;
```

```
   aRecord:          RecordIDPtr;            {creation ID returned here}
   allowDuplicate:   BOOLEAN;
END;

DirDeleteRecordPB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   aRecord:       RecordIDPtr;
END;

DirGetRecordMetaInfoPB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   aRecord:       RecordIDPtr;
   metaInfo:      DirMetaInfo;
END;
```

```
DirGetNameAndTypePB = RECORD
   qLink:          Ptr;
   reserved1:      LONGINT;
   reserved2:      LONGINT;
   ioCompletion:   ProcPtr;
   ioResult:       OSErr;
   saveA5:         LONGINT;
   reqCode:        INTEGER;
   reserved:       ARRAY[1..2] OF LONGINT;
   serverHint:     AddrBlock;
   dsRefNum:       INTEGER;
   callID:         LONGINT;
   identity:       AuthIdentity;
   gReserved1:     LONGINT;
   gReserved2:     LONGINT;
   gReserved3:     LONGINT;
   clientData:     Longint;
   aRecord:        RecordIDPtr;
END;

DirSetNameAndTypePB = PACKED RECORD
   qLink:             Ptr;
   reserved1:         LONGINT;
   reserved2:         LONGINT;
   ioCompletion:      ProcPtr;
   ioResult:          OSErr;
   saveA5:            LONGINT;
   reqCode:           INTEGER;
   reserved:          ARRAY[1..2] OF LONGINT;
   serverHint:        AddrBlock;
   dsRefNum:          INTEGER;
   callID:            LONGINT;
   identity:          AuthIdentity;
   gReserved1:        LONGINT;
   gReserved2:        LONGINT;
   gReserved3:        LONGINT;
   clientData:        LONGINT;
   aRecord:           RecordIDPtr;
   allowDuplicate:    BOOLEAN;
   padByte:           Byte;
   newName:           RStringPtr;             {new name for the record}
   newType:           RStringPtr;             {new type for the record}
END;
```

```
DirAddPseudonymPB = RECORD
   qLink:            Ptr;
   reserved1:        LONGINT;
   reserved2:        LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   reserved:         ARRAY[1..2] OF LONGINT;
   serverHint:       AddrBlock;
   dsRefNum:         INTEGER;
   callID:           LONGINT;
   identity:         AuthIdentity;
   gReserved1:       LONGINT;
   gReserved2:       LONGINT;
   gReserved3:       LONGINT;
   clientData:       LONGINT;
   aRecord:          RecordIDPtr;   {record ID to be added to pseudonym}
   pseudonymName:    RStringPtr;    {new name to be added as pseudonym}
   pseudonymType:    RStringPtr;    {new name to be added as pseudonym}
   allowDuplicate:   BOOLEAN;
END;

DirDeletePseudonymPB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   aRecord:       RecordIDPtr;         {record ID to which pseudonym is
                                        to be added}
```

```
   pseudonymName: RStringPtr;          {pseudonymName to be deleted}
   pseudonymType: RStringPtr;          {pseudonymType to be deleted}
END;

DirEnumeratePseudonymGetPB = PACKED RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
   reserved2:              LONGINT;
   ioCompletion:           ProcPtr;
   ioResult:               OSErr;
   saveA5:                 LONGINT;
   reqCode:                INTEGER;
   reserved:               ARRAY[1..2] OF LONGINT;
   serverHint:             AddrBlock;
   dsRefNum:               INTEGER;
   callID:                 LONGINT;
   identity:               AuthIdentity;
   gReserved1:             LONGINT;
   gReserved2:             LONGINT;
   gReserved3:             LONGINT;
   clientData:             LONGINT;
   aRecord:                RecordIDPtr;
   startingName:           RStringPtr;
   startingType:           RStringPtr;
   dReserved:              LONGINT;
   eReserved:              LONGINT;
   fReserved:              LONGINT;
   gReserved:              LONGINT;
   hReserved:              LONGINT;
   includeStartingPoint:   BOOLEAN; {if true, the pseudonym specified}
                                     by starting point will be included}
   padByte:                Byte;
   i1Reserved:             INTEGER;
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
END;

DirEnumeratePseudonymParsePB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
```

```
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   aRecord:       RecordIDPtr;      {same as DirEnumerateAliasesGetPB}
   bReserved:     LONGINT;
   cReserved:     LONGINT;
   eachRecordID:  ForEachRecordID;
   eReserved:     LONGINT;
   fReserved:     LONGINT;
   gReserved:     LONGINT;
   hReserved:     LONGINT;
   iReserved:     LONGINT;
   getBuffer:     Ptr;
   getBufferSize: LONGINT;
END;

DirAddAliasPB = RECORD
   qLink:            Ptr;
   reserved1:        LONGINT;
   reserved2:        LONGINT;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LONGINT;
   reqCode:          INTEGER;
   reserved:         ARRAY[1..2] OF LONGINT;
   serverHint:       AddrBlock;
   dsRefNum:         INTEGER;
   callID:           LONGINT;
   identity:         AuthIdentity;
   gReserved1:       LONGINT;
   gReserved2:       LONGINT;
   gReserved3:       LONGINT;
   clientData:       LONGINT;
   aRecord:          RecordIDPtr;
   allowDuplicate:   BOOLEAN;
END;
DirAddAttributeValuePB = RECORD
```

```
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    Longint;
   aRecord:       RecordIDPtr;
   attr:          AttributePtr;
END;

DirDeleteAttributeValuePB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   aRecord:       RecordIDPtr;
   attr:          AttributePtr;
END;

DirChangeAttributeValuePB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
```

```
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   aRecord:       RecordIDPtr;
   currentAttr:   AttributePtr;
   newAttr:       AttributePtr;
END;

DirVerifyAttributeValuePB = RECORD
   qLink:         Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   aRecord:       RecordIDPtr;
   attr:          AttributePtr;
END;

DirFindValuePB = RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
```

```
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
   gReserved1:          LONGINT;
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
   clientData:          LONGINT;
   aRLI:                PackedRLIPtr;     {an RLI specifying the cluster
                                           to be enumerated}
   aRecord:             LocalRecordIDPtr; {if not nil, look only in this
                                           record}
   attrType:            AttributeTypePtr; {if not nil, look only in this
                                           attribute type}
   startingRecord:      LocalRecordIDPtr; {record in which to start
                                           searching}
   startingAttribute:   AttributePtr;     {attribute in which to start
                                           searching}
   recordFound:         LocalRecordIDPtr; {record in which data was found}
   attributeFound:      Attribute;        {attribute in which data was
                                           found}
   matchSize:           LONGINT;          {length of matching bytes}
   matchingData:        Ptr;              {data bytes to be matched in
                                           search}
   sortDirection:       DirSortDirection; {sort direction (forwards or
                                           backwards)}
END;

DirLookupGetPB = RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
   reserved2:              LONGINT;
   ioCompletion:           ProcPtr;
   ioResult:               OSErr;
   saveA5:                 LONGINT;
   reqCode:                INTEGER;
   reserved:               ARRAY[1..2] OF LONGINT;
   serverHint:             AddrBlock;
   dsRefNum:               INTEGER;
```

```
   callID:                 LONGINT;
   identity:               AuthIdentity;
   gReserved1:             LONGINT;
   gReserved2:             LONGINT;
   gReserved3:             LONGINT;
   clientData:             LONGINT;
   aRecordList:            ^RecordIDPtr;     {an array of record ID
                                              pointers}
   attrTypeList:           ^AttributeTypePtr;{an array of attribute types}
   cReserved:              LONGINT;
   dReserved:              LONGINT;
   eReserved:              LONGINT;
   fReserved:              LONGINT;
   recordIDCount:          LONGINT;
   attrTypeCount:          LONGINT;
   includeStartingPoint:   BOOLEAN;          {if true, return the value
                                              specified by the starting
                                              indices}
   {padByte:               Byte;}
   i1Reserved:             INTEGER;
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
   startingRecordIndex:    LONGINT;          {start from this record}
   startingAttrTypeIndex:  LONGINT;          {start from this attribute
                                              type}
   startingAttribute:      Attribute;        {start from this attribute
                                              value}
   pReserved:              LONGINT;
END;

DirLookupParsePB = RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
   gReserved1:          LONGINT;
```

```
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
   clientData:          LONGINT;
   aRecordList:         ^RecordIDPtr;     {must be same from the
                                           corresponding Get call}
   attrTypeList:        ^AttributeTypePtr;{must be same from the
                                           corresponding Get call}
   cReserved:           LONGINT;
   eachRecordID:        ForEachLookupRecordID;
   eachAttrType:        ForEachAttrTypeLookup;
   eachAttrValue:       ForEachAttrValue;
   recordIDCount:       LONGINT;          {must be same from the
                                           corresponding Get call}
   attrTypeCount:       LONGINT;          {must be same from the
                                           corresponding Get call}
   iReserved:           LONGINT;
   getBuffer:           Ptr;              {must be same from the
                                           corresponding Get call}
   getBufferSize:       LONGINT;          {must be same from the
                                           corresponding Get call}
   lastRecordIndex:     LONGINT;          {last RecordID processed when
                                           parse completed}
   lastAttributeIndex:  LONGINT;          {last Attribute Type processed
                                           when parse completed}
   lastAttribute:       Attribute;        {last attribute value (with
                                           this creation ID) processed
                                           when parse completed}
   attrSize:            LONGINT;          {length of the attribute that
                                           was not returned}
END;

DirDeleteAttributeTypePB = RECORD
   qLink:               Ptr;
   reserved1:           LONGINT;
   reserved2:           LONGINT;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LONGINT;
   reqCode:             INTEGER;
   reserved:            ARRAY[1..2] OF LONGINT;
   serverHint:          AddrBlock;
   dsRefNum:            INTEGER;
   callID:              LONGINT;
   identity:            AuthIdentity;
```

```
   gReserved1:          LONGINT;
   gReserved2:          LONGINT;
   gReserved3:          LONGINT;
   clientData:          LONGINT;
   aRecord:             RecordIDPtr;
   attrType:            AttributeTypePtr;
END;

DirEnumerateAttributeTypesGetPB = PACKED RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
   reserved2:              LONGINT;
   ioCompletion:           ProcPtr;
   ioResult:               OSErr;
   saveA5:                 LONGINT;
   reqCode:                INTEGER;
   reserved:               ARRAY[1..2] OF LONGINT;
   serverHint:             AddrBlock;
   dsRefNum:               INTEGER;
   callID:                 LONGINT;
   identity:               AuthIdentity;
   gReserved1:             LONGINT;
   gReserved2:             LONGINT;
   gReserved3:             LONGINT;
   clientData:             LONGINT;
   aRecord:                RecordIDPtr;
   startingAttrType:       AttributeTypePtr;{starting point}
   cReserved:              LONGINT;
   dReserved:              LONGINT;
   eReserved:              LONGINT;
   fReserved:              LONGINT;
   gReserved:              LONGINT;
   hReserved:              LONGINT;
   includeStartingPoint:   BOOLEAN;    {if true, return the attribute
                                        Type specified by starting point}
   padByte:                Byte;
   i1Reserved:             INTEGER;
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
END;

DirEnumerateAttributeTypesParsePB = RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
```

```
   reserved2:              LONGINT;
   ioCompletion:           ProcPtr;
   ioResult:               OSErr;
   saveA5:                 LONGINT;
   reqCode:                INTEGER;
   reserved:               ARRAY[1..2] OF LONGINT;
   serverHint:             AddrBlock;
   dsRefNum:               INTEGER;
   callID:                 LONGINT;
   identity:               AuthIdentity;
   gReserved1:             LONGINT;
   gReserved2:             LONGINT;
   gReserved3:             LONGINT;
   clientData:             LONGINT;
   aRecord:                RecordIDPtr;{Same as
                                   DirEnumerateAttributeTypesGetPB}
   bReserved:              LONGINT;
   cReserved:              LONGINT;
   dReserved:              LONGINT;
   eachAttrType:           ForEachAttrType;
   fReserved:              LONGINT;
   gReserved:              LONGINT;
   hReserved:              LONGINT;
   iReserved:              LONGINT;
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
END;

DirGetDNodeAccessControlGetPB = RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
   reserved2:              LONGINT;
   ioCompletion:           ProcPtr;
   ioResult:               OSErr;
   saveA5:                 LONGINT;
   reqCode:                INTEGER;
   reserved:               ARRAY[1..2] OF LONGINT;
   serverHint:             AddrBlock;
   dsRefNum:               INTEGER;
   callID:                 LONGINT;
   identity:               AuthIdentity;
   gReserved1:             LONGINT;
   gReserved2:             LONGINT;
   gReserved3:             LONGINT;
```

```
   clientData:             LONGINT;
   pRLI:                   PackedRLIPtr;  {RLI of the cluster whose
                                           access control list is sought}
   bReserved:              LONGINT;
   cReserved:              LONGINT;
   dReserved:              LONGINT;
   eResreved:              LONGINT;
   forCurrentUserOnly:     BOOLEAN;
   startingPoint:          ^DSSpec;       {starting point}
   includeStartingPoint:   BOOLEAN;       {if true, return the DsObject
                                           specified in starting point}
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
END;

DirGetDNodeAccessControlParsePB = RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
   reserved2:              LONGINT;
   ioCompletion:           ProcPtr;
   ioResult:               OSErr;
   saveA5:                 LONGINT;
   reqCode:                INTEGER;
   reserved:               ARRAY[1..2] OF LONGINT;
   serverHint:             AddrBlock;
   dsRefNum:               INTEGER;
   callID:                 LONGINT;
   identity:               AuthIdentity;
   gReserved1:             LONGINT;
   gReserved2:             LONGINT;
   gReserved3:             LONGINT;
   clientData:             LONGINT;
   pRLI:                   PackedRLIPtr;              {RLI of the cluster}
   bReserved:              LONGINT;                   {unused}
   cReserved:              LONGINT;                   {unused}
   dReserved:              LONGINT;                   {unused}
   eachObject:             ForEachDNodeAccessControl;
   forCurrentUserOnly:     BOOLEAN;
   startingPoint:          ^DSSpec;                   {starting point}
   includeStartingPoint:   BOOLEAN;                   {if true, return
                                                       the record
                                                       specified in
                                                       in starting point}
```

```
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
END;

DirGetRecordAccessControlGetPB = RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
   reserved2:              LONGINT;
   ioCompletion:           ProcPtr;
   ioResult:               OSErr;
   saveA5:                 LONGINT;
   reqCode:                INTEGER;
   reserved:               ARRAY[1..2] OF LONGINT;
   serverHint:             AddrBlock;
   dsRefNum:               INTEGER;
   callID:                 LONGINT;
   identity:               AuthIdentity;
   gReserved1:             LONGINT;
   gReserved2:             LONGINT;
   gReserved3:             LONGINT;
   clientData:             LONGINT;
   aRecord:                RecordIDPtr;   {RecordID whose access
                                           control list is sought }
   bReserved:              LONGINT;       {unused}
   cReserved:              LONGINT;       {unused}
   dReserved:              LONGINT;       {unused}
   eResreved:              LONGINT;
   forCurrentUserOnly:     BOOLEAN;
   startingPoint:          ^DSSpec;       {starting Point}
   includeStartingPoint:   BOOLEAN;       {if true, return the DsObject
                                           specified in starting point}
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
END;

DirGetRecordAccessControlParsePB = RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
   reserved2:              LONGINT;
   ioCompletion:           ProcPtr;
   ioResult:               OSErr;
   saveA5:                 LONGINT;
   reqCode:                INTEGER;
   reserved:               ARRAY[1..2] OF LONGINT;
```

```
   serverHint:             AddrBlock;
   dsRefNum:               INTEGER;
   callID:                 LONGINT;
   identity:               AuthIdentity;
   gReserved1:             LONGINT;
   gReserved2:             LONGINT;
   gReserved3:             LONGINT;
   clientData:             LONGINT;
   aRecord:                RecordIDPtr;         {RecordID whose access
                                                 control list is sought}
   bReserved:              LONGINT;             {unused}
   cReserved:              LONGINT;             {unused}
   dReserved:              LONGINT;             {unused}
   eachObject:             ForEachRecordAccessControl;
   forCurrentUserOnly:     BOOLEAN;
   startingPoint:          ^DSSpec;             {starting point}
   includeStartingPoint:   BOOLEAN;             {if true, return the
                                                 record specified in}
                                                {starting point}
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
END;

DirGetAttributeAccessControlGetPB = RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
   reserved2:              LONGINT;
   ioCompletion:           ProcPtr;
   ioResult:               OSErr;
   saveA5:                 LONGINT;
   reqCode:                INTEGER;
   reserved:               ARRAY[1..2] OF LONGINT;
   serverHint:             AddrBlock;
   dsRefNum:               INTEGER;
   callID:                 LONGINT;
   identity:               AuthIdentity;
   gReserved1:             LONGINT;
   gReserved2:             LONGINT;
   gReserved3:             LONGINT;
   clientData:             LONGINT;
   aRecord:                RecordIDPtr;      {RecordID whose access
                                              control list is sought}
   aType:                  AttributeTypePtr; {attribute type to which
                                              access controls are sought}
```

```
   cReserved:              LONGINT;
   dReserved:              LONGINT;          {unused}
   eResreved:              LONGINT;
   forCurrentUserOnly:     BOOLEAN;
   includeStartingPoint:   BOOLEAN;          {if true, return the DsObject
                                              specified in starting point}
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
END;

DirGetAttributeAccessControlParsePB = RECORD
   qLink:                  Ptr;
   reserved1:              LONGINT;
   reserved2:              LONGINT;
   ioCompletion:           ProcPtr;
   ioResult:               OSErr;
   saveA5:                 LONGINT;
   reqCode:                INTEGER;
   reserved:               ARRAY[1..2] OF LONGINT;
   serverHint:             AddrBlock;
   dsRefNum:               INTEGER;
   callID:                 LONGINT;
   identity:               AuthIdentity;
   gReserved1:             LONGINT;
   gReserved2:             LONGINT;
   gReserved3:             LONGINT;
   clientData:             LONGINT;
   aRecord:                RecordIDPtr;      {record ID whose access
                                              control list is sought}
   aType:                  AttributeTypePtr; {attribute type whose
                                              access controls are sought}
   cReserved:              LONGINT;
   dReserved:              LONGINT;
   eachObject:             ForEachAttributeAccessControl;
   forCurrentUserOnly:     BOOLEAN;
   startingPoint:          ^DSSpec;          {starting Point }
   includeStartingPoint:   BOOLEAN;          {if true, return the record
                                              specified in starting point}
   getBuffer:              Ptr;
   getBufferSize:          LONGINT;
END;
```

```
DirAbortPB = RECORD
   qLink: Ptr;
   reserved1:     LONGINT;
   reserved2:     LONGINT;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LONGINT;
   reqCode:       INTEGER;
   reserved:      ARRAY[1..2] OF LONGINT;
   serverHint:    AddrBlock;
   dsRefNum:      INTEGER;
   callID:        LONGINT;
   identity:      AuthIdentity;
   gReserved1:    LONGINT;
   gReserved2:    LONGINT;
   gReserved3:    LONGINT;
   clientData:    LONGINT;
   pb:            Ptr;        {parameter block for the call that must be
                               aborted {^DirParamBlock}}
END;

DirParamBlock = RECORD
   CASE INTEGER OF
   1: (header:                      AuthDirParamHeader);
   2: (addRecordPB:                 DirAddRecordPB);
   3: (deleteRecordPB:              DirDeleteRecordPB);
   4: (enumerateGetPB:              DirEnumerateGetPB);
   5: (enumerateParsePB:            DirEnumerateParsePB);
   6: (findRecordGetPB:             DirFindRecordGetPB);
   7: (findRecordParsePB:           DirFindRecordParsePB);
   8:(lookupGetPB:                  DirLookupGetPB);
   9:(lookupParsePB:                DirLookupParsePB);
   10:(addAttributeValuePB:         DirAddAttributeValuePB);
   11:(deleteAttributeTypePB:       DirDeleteAttributeTypePB);
   12:(deleteAttributeValuePB:      DirDeleteAttributeValuePB);
   13:(changeAttributeValuePB:      DirChangeAttributeValuePB);
   14:(verifyAttributeValuePB:      DirVerifyAttributeValuePB);
   15:(findValuePB:                 DirFindValuePB);
   16:(enumeratePseudonymGetPB:     DirEnumeratePseudonymGetPB);
   17:(enumeratePseudonymParsePB:   DirEnumeratePseudonymParsePB);
   18:(addPseudonymPB:              DirAddPseudonymPB);
   19:(deletePseudonymPB:           DirDeletePseudonymPB);
   20:(addAliasPB:                  DirAddAliasPB);
   21:(enumerateAttributeTypesGetPB:DirEnumerateAttributeTypesGetPB);
```

```
22:(enumerateAttributeTypesParsePB:
                                DirEnumerateAttributeTypesParsePB);
23:(getNameAndTypePB:           DirGetNameAndTypePB);
24:(setNameAndTypePB:           DirSetNameAndTypePB);
25:(getRecordMetaInfoPB:        DirGetRecordMetaInfoPB);
26:(getDNodeMetaInfoPB:         DirGetDNodeMetaInfoPB);
27:(getDirectoryInfoPB:         DirGetDirectoryInfoPB);
28:(getDNodeAccessControlGetPB:  DirGetDNodeAccessControlGetPB);
29:(getDNodeAccessControlParsePB:DirGetDNodeAccessControlParsePB);
30:(getRecordAccessControlGetPB: DirGetRecordAccessControlGetPB);
31:(getRecordAccessControlParsePB:
                                DirGetRecordAccessControlParsePB);
32:(getAttributeAccessControlGetPB:
                                DirGetAttributeAccessControlGetPB);
33:(getAttributeAccessControlParsePB:
                                DirGetAttributeAccessControlParsePB);
34: (enumerateDirectoriesGetPB:  DirEnumerateDirectoriesGetPB);
35: (enumerateDirectoriesParsePB: DirEnumerateDirectoriesParsePB);
36: (addADAPDirectoryPB:         DirAddADAPDirectoryPB);
37: (removeDirectoryPB:          DirRemoveDirectoryPB);
38: (netSearchADAPDirectoriesGetPB:
                                DirNetSearchADAPDirectoriesGetPB);
39: (netSearchADAPDirectoriesParsePB:
                                DirNetSearchADAPDirectoriesParsePB);
40: (findADAPDirectoryByNetSearchPB:
                                DirFindADAPDirectoryByNetSearchPB);
41: (mapDNodeNumberToPathNamePB: DirMapDNodeNumberToPathNamePB);
42: (mapPathNameToDNodeNumberPB: DirMapPathNameToDNodeNumberPB);
43: (getLocalNetworkSpecPB:      DirGetLocalNetworkSpecPB);
44: (getDNodeInfoPB:             DirGetDNodeInfoPB);

{calls for personal catalogs}

45: (createPersonalDirectoryPB:  DirCreatePersonalDirectoryPB);
46: (openPersonalDirectoryPB:    DirOpenPersonalDirectoryPB);
47: (closePersonalDirectoryPB:   DirClosePersonalDirectoryPB);
48: (makePersonalDirectoryRLIPB: DirMakePersonalDirectoryRLIPB);

{calls For CSAMs}

49: (addDSAMPB:                  DirAddDSAMPB);
50: (instantiateDSAMPB:          DirInstantiateDSAMPB);
51: (removeDSAMPB:               DirRemoveDSAMPB);
52: (addDSAMDirectoryPB:         DirAddDSAMDirectoryPB);
```

```
      53: (getExtendedDirectoriesInfoPB:
                                    DirGetExtendedDirectoriesInfoPB);
      54: (getDirectoryIconPB:      DirGetDirectoryIconPB);

      {call to dsRefNum for system(Setup: PowerTalk) personal catalog}

      55: (dirGetOCESetupRefNumPB:  DirGetOCESetupRefNumPB);

      {abort a aysnchronous call}

      56: (abortPB:                 DirAbortPB);

   END;

   DirParamBlockPtr = ^DirParamBlock;
```

Catalog Manager Functions

Getting Information About Catalogs

```
FUNCTION DirEnumerateDirectoriesGet
                          (paramBlock: DirParamBlockPtr;
                           async: BOOLEAN): OSErr;
FUNCTION DirEnumerateDirectoriesParse
                          (paramBlock: DirParamBlockPtr;
                           async: BOOLEAN): OSErr;
FUNCTION DirFindRecordGet (paramBlock: DirParamBlockPtr;
                           async: BOOLEAN): OSErr;
FUNCTION DirFindRecordParse (paramBlock: DirParamBlockPtr;
                           async: BOOLEAN): OSErr;
FUNCTION DirGetDirectoryInfo
                          (paramBlock: DirParamBlockPtr;
                           async: BOOLEAN): OSErr;
FUNCTION DirGetLocalNetworkSpec
                          (paramBlock: DirParamBlockPtr;
                           async: BOOLEAN): OSErr;
FUNCTION DirGetDirectoryIcon
                          (paramBlock: DirParamBlockPtr;
                           async: BOOLEAN): OSErr;
FUNCTION DirGetExtendedDirectoriesInfo
                          (paramBlock: DirParamBlockPtr;
                           async: BOOLEAN): OSErr;
```

Getting Information About DNodes

```
FUNCTION DirEnumerateGet    (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirEnumerateParse  (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetDNodeMetaInfo
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirMapDNodeNumberToPathName
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirMapPathNameToDNodeNumber
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetDNodeInfo    (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
```

Maintaining the PowerTalk Setup Catalog

```
FUNCTION DirAddADAPDirectory
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirFindADAPDirectoryByNetSearch
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirNetSearchADAPDirectoriesGet
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirNetSearchADAPDirectoriesParse
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirRemoveDirectory
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetOCESetupRefNum
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
```

Creating, Opening, and Closing Personal Catalogs

```
FUNCTION DirCreatePersonalDirectory
                            (paramBlock: DirParamBlockPtr): OSErr;
FUNCTION DirOpenPersonalDirectory
                            (paramBlock: DirParamBlockPtr): OSErr;
```

```
FUNCTION DirClosePersonalDirectory
                            (paramBlock: DirParamBlockPtr): OSErr;
FUNCTION DirMakePersonalDirectoryRLI
                            (paramBlock: DirParamBlockPtr): OSErr;
```

Managing Records

```
FUNCTION DirAddRecord       (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirDeleteRecord    (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetRecordMetaInfo
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetNameAndType  (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirSetNameAndType  (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirAddPseudonym    (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirDeletePseudonym
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirEnumeratePseudonymGet
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirEnumeratePseudonymParse
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirAddAlias        (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
```

Managing Attribute Types and Values

```
FUNCTION DirAddAttributeValue
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirDeleteAttributeValue
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirChangeAttributeValue
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
```

```
FUNCTION DirVerifyAttributeValue
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirFindValue       (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirLookupGet       (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirLookupParse     (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirDeleteAttributeType
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirEnumerateAttributeTypesGet
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirEnumerateAttributeTypesParse
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
```

Reading Access Controls for dNodes, Records, and Attribute Types

```
FUNCTION OCEGetAccessControlDSSpec
                            (categoryBitMask: CategoryMask): DSSpecPtr;
FUNCTION DirGetDNodeAccessControlGet
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetDNodeAccessControlParse
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetRecordAccessControlGet
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetRecordAccessControlParse
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetAttributeAccessControlGet
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
FUNCTION DirGetAttributeAccessControlParse
                            (paramBlock: DirParamBlockPtr;
                             async: BOOLEAN): OSErr;
```

Canceling a Catalog Manager Function

```
FUNCTION DirAbort           (paramBlock: DirParamBlockPtr): OSErr;
```

Application-Defined Functions

```
FUNCTION MyCompletionRoutine
                            (paramBlk: DirParamBlockPtr);
FUNCTION MyForEachRecordID  (clientData: long;
                             recordID: RecordID): BOOLEAN;
FUNCTION MyForEachAttrType  (clientData: long;
                             attrType: AttributeType): BOOLEAN;
FUNCTION MyForEachDirectory
                            (clientData: long; dirName: DirectoryNamePtr;
                             discriminator: DirDiscriminator;
                             features: DirGestalt): BOOLEAN;
FUNCTION MyForEachLookupRecordID
                            (clientData: long;
                             recordID: RecordID): BOOLEAN;
FUNCTION MyForEachAttrTypeLookup
                            (clientData: long; attrType: AttributeTypePtr;
                             myAttrAccMask: AccessMask): BOOLEAN;
FUNCTION MyForEachAttrValue (clientData: long;
                             attribute: Attribute): BOOLEAN;
FUNCTION MyForEachDirEnumSpec
                            (clientData: LONGINT;
                             enumSpec: DirEnumSpec): BOOLEAN;
FUNCTION MyForEachRecord    ((clientData: long;
                             dsObj: DSSpec; activeDnodeAccMask: AccessMask;
                             activeRecordAccMask: AccessMask;
                             defaultAttributeAccMask: AccessMask): BOOLEAN;}
FUNCTION MyForEachADAPDirectory
                            (clientData: long; dirName: DirectoryNamePtr;
                             discriminator: DirDiscriminator;
                             features: DirGestalt; serverHint: AddrBlock):
                             BOOLEAN;
FUNCTION MyForEachDNodeAccessControl
                            (clientData: long; dsObj: DSSpec;
                             activeDnodeAccMask: AccessMask;
                             defaultRecordAccMask: AccessMask;
                             defaultAttributeAccMask: AccessMask):BOOLEAN;
```

```
FUNCTION MyForEachRecordAccessControl
                              (clientData: long; dsObj: DSSpec;
                               activeDnodeAccMask: AccessMask;
                               activeRecordAccMask: AccessMask;
                               defaultAttributeAccMask: AccessMask):BOOLEAN;
FUNCTION MyForEachAttributeAccessControl
                              (clientData: long; dsObj: DSSpec;
                               activeDnodeAccMask: AccessMask;
                               activeRecordAccMask: AccessMask;
                               activeAttributeAccMask: AccessMask): BOOLEAN;
```

Assembly-Language Summary

Trap Macros Requiring Routine Selectors

`_oceTBDispatch`

Selector	Routine
0x101	DirEnumerateParse
0x102	DirLookupParse
0x103	DirEnumerateAttributeTypesParse
0x104	DirEnumeratePseudonymParse
0x105	DirNetSearchADAPDirectoriesParse
0x106	DirEnumerateDirectoriesParse
0x107	DirFindADAPDirectoryByNetSearch
$0108	`DirNetSearchADAPDirectoriesGet`
$0109	`DirAddRecord`
$010A	`DirDeleteRecord`
$010B	`DirAddAttributeValue`
$010C	`DirDeleteAttributeValue`
$010D	`DirChangeAttributeValue`
$010E	`DirVerifyAttributeValue`
$010F	`DirAddPseudonym`
$0110	`DirDeletePseudonym`
$0111	`DirEnumerateGet`
$0112	`DirEnumerateAttributeTypesGet`
$0113	`DirEnumeratePseudonymGet`
$0114	`DirGetNameAndType`
$0115	`DirSetNameAndType`
$0116	`DirGetRecordMetaInfo`

Selector	Routine
$0117	DirLookupGet
$0118	DirGetDNodeMetaInfo
$0119	DirGetDirectoryInfo
$011A	DirEnumerateDirectoriesGet
$011B	DirAbort
$011C	DirAddAlias
$011D	DirAddDSAM
$011E	DirOpenPersonalDirectory
$011F	DirCreatePersonalDirectory
$0121	DirGetDirectoryIcon
$0122	DirMapPathNameToDNodeNumber
$0123	DirMapDNodeNumberToPathName
$0124	DirGetLocalNetworkSpec
$0125	DirGetDNodeInfo
$0126	DirFindValue
$0128	DirGetOCESetupRefNum
$012A	DirGetDNodeAccessControlGet
$012C	DirGetRecordAccessControlGet
$012E	DirGetAttributeAccessControlGet
$012F	DirGetDNodeAccessControlParse
$0130	DirDeleteAttributeType
$0131	DirClosePersonalDirectory
$0132	DirMakePersonalDirectoryRLI
$0134	DirGetRecordAccessControlParse
$0135	DirRemoveDirectory
$0136	DirGetExtendedDirectoriesInfo
$0137	DirAddADAPDirectory
$0138	DirGetAttributeAccessControlParse
$0140	DirFindRecordGet
$0141	DirFindRecordParse

Result Codes

The allocated range of result codes for the Catalog Manager is –1610 through –1646 and there are some result codes in the range –1503 through –1567. Functions may also return result codes from other AOCE managers and standard Macintosh result codes such as `noErr` 0 (No error) and `fnfErr` –43 (File not found).

`kOCEBufferTooSmall`	–1503	Buffer too small for data requested
`kOCEVersionErr`	–1504	Need to sort personal catalog
`kOCEAlreadyExists`	–1510	The catalog being added already exists
`kOCEReadAccessDenied`	–1540	Identity lacks read access privileges
`kOCEWriteAccessDenied`	–1541	Identity lacks write access privileges
`kOCEUnknownID`	–1567	Authentication identity is not valid
`kOCENotLocal`	-1610	The server does not serve the requested dNode
`kOCETooBusy`	-1611	Server cannot complete call at this time
`kOCEDatabaseFull`	-1612	The disk is full
`kOCETargetDirectoryInaccessible`	-1613	Target catalog is not currently available
`kOCEBogusArgs`	-1614	Args not formatted correctly on the wire
`kOCENoSuchDNode`	-1615	Can’t find specified dNode
`kOCEDNodeUnavailable`	-1616	Could not find any servers that serve the requested dNode
`kOCEBadRecordID`	-1617	Record name or record type doesn’t match creation ID
`kOCENoSuchRecord`	-1618	Can't find specified record
`kOCENoSuchAttributeValue`	-1619	Can’t find specified attribute value
`kOCENoSuchPseudonym`	-1620	The specified pseudonym does not exist
`kOCEAttributeValueTooBig`	-1621	Attribute value is larger than `kAttrValueMaxBytes` bytes
`kOCETypeExists`	-1622	The type already exists in the record
`kOCEMoreData`	-1623	More data available
`kOCERefNumBad`	-1624	RefNum is not valid
`kOCEStreamCreationErr`	-1625	Error in creating connection to server
`kOCEOperationNotSupported`	-1626	The specified catalog does not support this operation
`kOCEPABNotOpen`	-1627	The specified personal catalog is not open to make the operation
`kOCEDSAMInstallErr`	-1628	The specified CSAM could not be installed
`kOCEDirListFullErr`	-1629	The catalog list is full; try removing an entry
`kOCEDirectoryNotFoundErr`	-1630	Can’t find catalog
`kOCEAbortNotSupportedForThisCall`	-1631	Abort not supported

kOCEAborted	-1632	The call was aborted
kOCEOCESetupRequired	-1633	LocalIdentity Setup is required
kOCEDSAMRecordNotFound	-1634	CSAM Record not found
kOCEDSAMNotInstantiated	-1635	CSAM is not instantiated
kOCEDSAMRecordExists	-1636	CSAM record already exists
kOCELengthError	-1637	The buffer supplied was too small
kOCEBadStartingRecord	-1638	Starting record index out of range
kOCEBadStartingAttribute	-1639	Starting attribute index is not within range
kOCEMoreAttrValue	-1640	Buffer too small for a single attribute value
kOCENoDupAllowed	-1641	Duplicate name and type
kOCENoSuchAttributeType	-1642	Can't find specified attribute type
kOCEMiscError	-1643	Miscellaneous error
kOCENoSuchIcon	-1644	There is no matching icon from OCEGetDirectoryIcon
kOCERLIsDontMatch	-1645	RLIs of different records in the record list are not the same
kOCEDirectoryCorrupt	-1646	Serious disk fill corruption problem

CHAPTER 9

Authentication Manager

Contents

Authentication Manager

This chapter describes the AOCE Authentication Manager, which provides authentication services for users of PowerShare catalog servers. Providers of other AOCE-compatible catalog servers can also use the Authentication Manager and the AppleTalk Secure Data Stream Protocol (ASDSP) to provide authentication services for users of their catalog servers. The services provided by the Authentication Manager ensure both ends of a connection that the entity on the other end is who or what it claims to be. The Authentication Manager does not encrypt data or guarantee the integrity of transmitted data. For other security services, see the chapter "Digital Signature Manager" in this book.

The Authentication Manager application programming interface (API) provides the tools you need to implement an authenticated connection between two entities. Also, the API includes a function that provides a common server-based time service.

The Authentication Manager provides low-level functions that are called by the AOCE Collaboration package, the AOCE Collaboration toolbox, the PowerTalk Key Chain, and the PowerShare Admin program.

An application running in the background might call the Authentication Manager to get a local identity or a specific identity. You might want to add your application to the local-identity notification queue, so that the Authentication Manager calls your notification routine when the local identity is locked or unlocked or when the local-identity name is changed.

You must read this chapter if you want to create your own authentication service using AOCE functions. For example, if you want to authenticate connections between users who are not connected over an AppleTalk network, you can use the Authentication Manager functions described in this chapter.

This chapter starts with a brief introduction to authentication, including an introduction to the role of servers in authentication. The chapter then presents information to help you use the Authentication Manager functions to

- generate and use encryption keys
- create and use authentication identities
- acquire and use credentials for mutual verification of users' identities
- generate proxies and use them
- resolve creation IDs for records
- obtain the universal coordinated time
- implement your own challenge process for authenticating two entities

The language specific to this technology is defined as the concepts are introduced in the chapter.

For a general overview of AOCE services, see the chapter "Introduction to Apple Open Collaboration Environment" in this book.

Introduction to Authentication

To avoid fraud or impersonation, two users or services communicating over a network may need to identify each other conclusively. For example, a user may want to verify that a piece of electronic mail came from the sender named in the letter. In the world of networking, verification of the identity of an entity on a network or of one end of a communication link is called **authentication**.

The authentication process involves the exchange between two parties of a sequence of messages referred to as *challenges* and *replies*. The Authentication Manager uses the Data Encryption Standard (DES, a symmetric private-key encryption algorithm that uses the same key for encryption and decryption) and a secret key derived from the user's password to encrypt each challenge or reply message. The authentication server knows the keys of both ends of the connection. Keys are discussed in the next section.

These are the basic assumptions fundamental to authentication:

- Each user or service has a key, and that key is known only to the user and the authentication server.
- The authentication server is trusted to reveal the secret key to no one.

The originator of a message is called the *initiator;* the addressee is the *recipient*. The initiator and recipient do not share a key. If they did, they could use that key to encrypt every message they exchange.

Keys

Encryption **keys** are numbers used by an encryption algorithm to encrypt and decrypt data. The keys of the initiator and recipient are referred to as **client keys**. Because the authentication process requires that a trusted third party know everyone's keys, Authentication Manager functions allow you to store client keys in a server-based catalog.

The Authentication Manager uses client keys for encrypting requests to the server and for encrypting the response the server returns to an initiator. The server also uses client keys to verify that a user typed his or her password correctly.

During the authentication process, the authentication server creates a unique time-limited **session key**, encrypts it, and transmits it to the initiator, who sends it to the recipient. The initiator and recipient use the session key to exchange challenges and replies. The section "Steps in the Authentication Process" beginning on page 9-5 describes the use of client keys and session keys.

Credentials

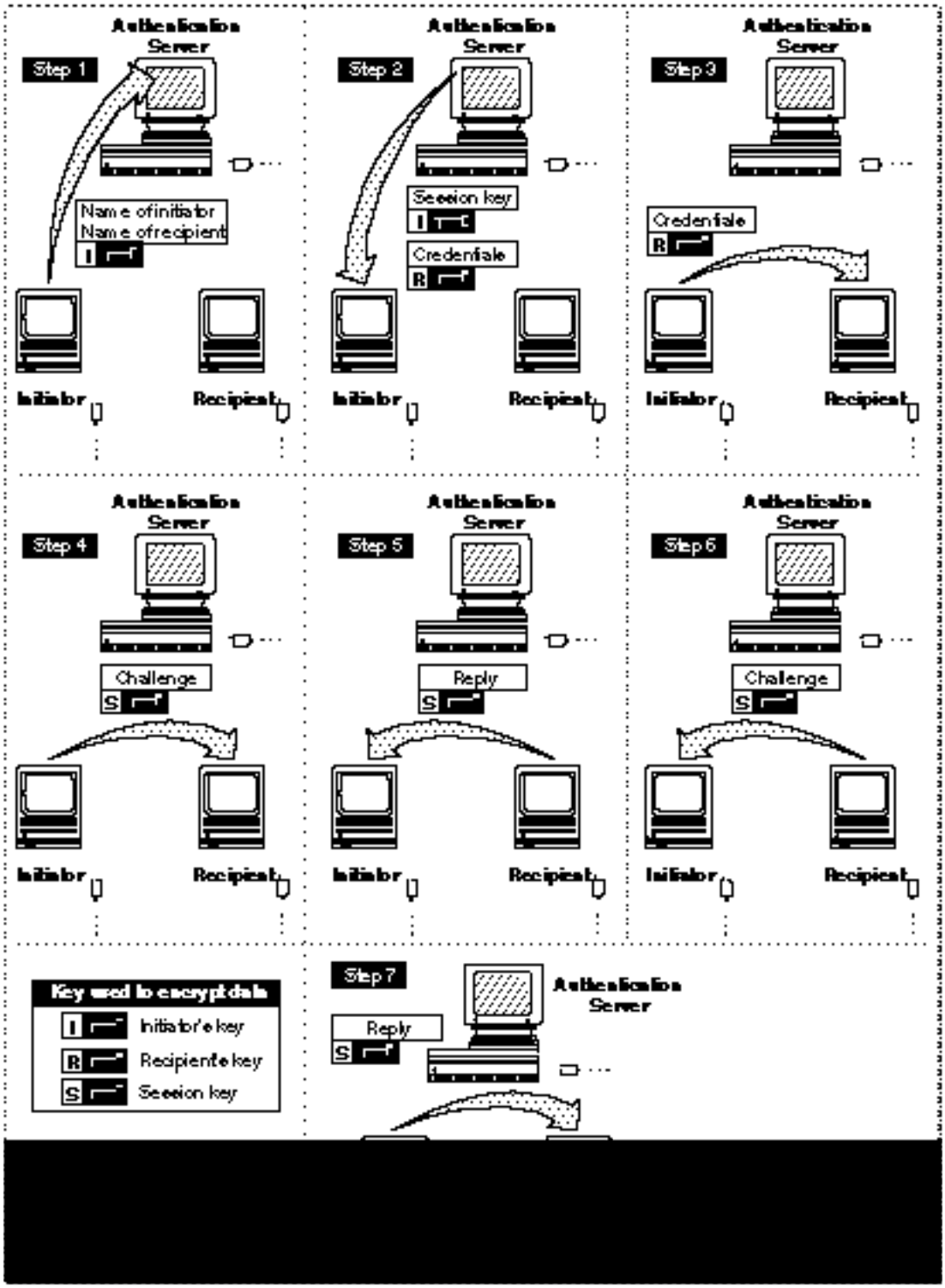
Credentials consist of an identifier for the initiator and a session key, encrypted in the key of the recipient. The initiator requests credentials from the authentication server and sends them to the recipient. With these, the recipient can determine which initiator wants to make an authenticated connection and can obtain the session key needed to complete the authentication process. Because the credentials are encrypted in the recipient's client key, only the intended recipient can use them, and the initiator cannot alter them. Therefore, the initiator can be sure that anyone responding with the correct session key is the intended recipient, and the recipient can be sure of the identity of the initiator.

Credentials are valid only for a particular initiator and recipient and only for a specific time period. After that time period, they cannot be used to establish a connection. However, once a communication stream is open and authenticated, the two ends of a connection can elect to maintain the connection even after the credentials have expired.

Steps in the Authentication Process

The authentication process consists of two phases: the *precontact phase* and the *challenge phase*. Figure 9-1 on page 9-6 shows the authentication process; in this figure, step 1 and step 2 represent the precontact phase, and the remaining steps represent the challenge phase of authentication. In Figure 9-1, For each step in the process, the figure shows what key was used to encrypt the data, who sends the data and to whom, and the nature of the data sent.

Figure 9-1 The authentication process



Here is what happens in the precontact phase of authentication:

1. The initiator encrypts both the name of the initiator and the name of the recipient in the initiator's client key and asks the server for credentials.
2. The server returns two quantities to the initiator: a session key and a credentials block. The session key is encrypted in the initiator's key. The credentials block is encrypted in the recipient's key so that not even the initiator can see what is in it.

Receipt of the credentials by the initiator completes the precontact phase of authentication. Next, the initiator can either use AppleTalk Secure Data Stream Protocol (ASDSP) to perform the challenge phase of authentication or else implement the challenge phase as described below. See the chapter "AppleTalk Data Stream Protocol" in *Inside Macintosh: Networking* for a discussion of ASDSP.

3. The initiator sends the credentials block to the recipient. This credentials block is encrypted in the recipient's key and contains the name of the initiator and a copy of the session key.

Now both the initiator and the recipient have a copy of the same session key. They now exchange challenges and replies to verify that each has the same session key.

4. The initiator selects a random number, encrypts it with the session key, and sends it to the recipient as a challenge.
5. The recipient decrypts the challenge, adds 1 to the number, encrypts the sum with the session key, and sends the new encrypted number to the initiator as a reply.

Because only the intended recipient can decrypt the credentials and therefore obtain the session key, the initiator has now established that the challenge was not intercepted by an impostor. The recipient must now issue a challenge to ensure that the initiator is truly the entity identified in the credentials.

6. The recipient selects a new random number, encrypts it with the session key, and sends it to the initiator as a challenge.
7. The initiator decrypts the number, adds 1, encrypts the sum with the session key, and sends it as a reply.

After two entities desiring a connection successfully complete this authentication process, they are ready to exchange authenticated messages. If you use ASDSP as the transport mechanism between an initiator and a recipient, the challenge phase of the authentication process is handled by the ASDSP function. If you are using another transport protocol, such as TCP/IP (Transmission Control Protocol/Internet Protocol), you can implement steps 4 through 7 of the authentication process using Authentication Manager functions described in "Non-ASDSP Authentication Utilities" beginning on page 9-54.

Identities

An **identity**, sometimes referred to as an *authentication identity*, is a number used as shorthand for the name and key of a user or service. Many AOCE functions require an identity to determine if the initiator is authorized to make a particular service request. There are two types of authentication identities: *local identities* and *specific identities*.

Whereas a local identity is associated with a particular computer, a specific identity is associated with a particular server or service. In most cases you use the local identity when you call an AOCE function, except when providing access to a service on behalf of someone other than the principal user of the computer. Local identities and specific identities are discussed in the following sections.

Local Identities

Because a user may have multiple "accounts" for a variety of applications or services, the PowerTalk system software provides a Setup catalog that contains (in encrypted form) the names and passwords for the services available to the user. A **local identity** is a number used as shorthand for the name and password associated with the user of a particular computer. This local identity is a "master" identity because it provides access to all catalogs and services in the PowerTalk Setup catalog without requiring each service's password individually. Any AOCE function that requires an identity as input can use the local identity.

The Standard Catalog Package function, `SDPPromptForID`, described in the chapter "Standard Catalog Package" in this book, prompts the user for his or her name and password and uses this information to generate the local identity.

A background application can obtain the local identity generated by the `SDPPromptForID` function by calling the Authentication Manager's `AuthGetLocalIdentity` function, described on page 9-28. If a local identity is not set up, you can install your application in a notification queue, so that the application is notified when the local identity is created or unlocked.

By supplying a valid local identity to any AOCE function that requires an identity parameter, you tell the AOCE toolbox what user is requesting the service. The toolbox prepares an authenticated stream to the server, and during this process the server learns the name of the user. Then the server checks the access controls for the user represented by the identity to ensure that the user has the privileges necessary to access the requested function. If the access controls are sufficient, the AOCE software provides the requested service. Otherwise, you receive a result code stating that the user's access rights are insufficient. Access controls are discussed in the chapter "Catalog Manager" in this book.

The functions you can use to manage local identities are described in "Local Identity Management" beginning on page 9-28.

Locking and Unlocking Local Identities

The PowerTalk system software gives users the option of protecting their accounts from unauthorized access. To do so, the user chooses Lock Key Chain from the Special menu of the Finder or sets the PowerTalk Setup control panel to lock the Key Chain after some specified period of inactivity. Upon returning, the user chooses Unlock Key Chain from the Finder's Special menu and is prompted for a password. You can also lock and unlock the local identity from within your application.

If the local identity is locked, it is the responsibility of your application to disable its own services appropriately. For example, if you are designing a mail application, you may

want it to continue receiving mail even when the local identity is locked but would probably not want to allow users to read mail that has been received.

Local Identity Status Notification

If your application needs to enable or disable features based on whether the local identity is unlocked, you may want to be notified of changes in the status of the local identity. If so, you can add your application to a notification queue. The applications in this queue are notified when the local identity is unlocked or locked. Through the notification queue, you can deny locking of the local identity when your application is in use. For instance, you might want to deny locking when your application is engaged in some process that would be seriously disrupted if the lock function succeeded.

Specific Identities

To provide a service to a user other than the principal user of a computer, you can use a specific identity rather than the local identity. The **specific identity** is a number used as shorthand for the name and key of the alternate user. You can use the specific identity in any AOCE function that requires an identity.

The Standard Catalog Package function `SDPPromptForID` prompts a user for a name and password and returns a specific identity. The `SDPPromptForID` function is described in the chapter “Standard Catalog Package” in this book.

Guest Access

When your application needs to accommodate users with no accounts on the computer or server, you can specify a “guest identity” by using the value 0 for the identity parameter in AOCE functions.

The PowerTalk Setup Catalog

The AOCE Catalog Manager defines a special personal catalog called the **PowerTalk Setup catalog**, which contains information about the catalogs and other services that are available to the principal user of the computer. The PowerTalk Setup catalog is stored on the user’s local disk. The records in the PowerTalk Setup catalog represent such entities as PowerShare catalogs, external catalogs, and catalog service access modules (CSAMs). Catalogs and CSAMs represented by records in the PowerTalk Setup catalog are said to be “listed in the PowerTalk Setup catalog.” The contents of the Setup catalog and the process of adding a CSAM or mail service access module (MSAM) to the Setup catalog are described in the chapter “Service Access Module Setup” in *Inside Macintosh: AOCE Service Access Modules***.**

You can use the functions described in “PowerTalk Setup Catalog Management” beginning on page 9-61 to set up, change, remove, or get information about items in the PowerTalk Setup catalog.

Proxies

A **proxy** allows an alternate entity to be authenticated as the user for a limited time. It is a privilege provided to an **intermediary**: a representative of the user or service. The intermediary uses the proxy to obtain the credentials needed to complete the authentication process. The proxy gives the intermediary access to a particular recipient to perform some task on behalf of an initiator.

For example, suppose a user of your application plans to be away from the computer but wants to back up some data when the server is not busy. In this case, your application can request a proxy for the user. You may assign the proxy to an intermediary, who can do the backup. With this proxy, the intermediary obtains credentials from the server and then uses them to create an authenticated connection in the usual way. Functions you can use to create and use proxies are described in “Credentials Management” beginning on page 9-43.

About the Authentication Manager

The Authentication Manager, the Digital Signature Manager, the Catalog Manager, and the Interprogram Messaging Manager together constitute the fundamental services of the AOCE system software. The Standard Catalog Package and the Standard Mail Package provide high-level interfaces to the Authentication Manager.

The Authentication Manager is a collection of functions that runs on the user’s computer and communicates with the authentication server to set up authenticated connections.

The Authentication Manager includes routines that provide the following services:

- key management: translating passwords to keys and adding, changing, and deleting keys in the server
- local identity management: determining the local identity for a computer; locking, unlocking, creating, changing, and removing local identities; and adding applications to and removing them from a notification queue for changes in the status of the local identity
- specific identity management: binding, unbinding, and getting information about specific identities
- credentials management: obtaining and using credentials and making and using proxies
- resolution of creation IDs: resolving creation IDs when multiple records have the same name and type
- time service: obtaining the universal coordinated time
- non-ASDSP authentication utilities: performing authentication as a step-by-step process
- PowerTalk Setup catalog management: setting up, changing, removing, and getting information about catalogs in the PowerTalk Setup catalog

Using the Authentication Manager

This section discusses the techniques you can use to perform tasks related to authentication. You can use the techniques in this section to

- perform the authentication process for initiators and recipients using ASDSP
- perform the precontact phase and challenge process of authentication for initiators and recipients using a different transport mechanism
- use a proxy in either of the above authentication processes
- monitor the status of access to the PowerTalk Setup catalog by installing your application in a notification queue

For more detailed descriptions of the routines described in this section, see "Authentication Manager Functions" beginning on page 9-20.

Determining Whether the Collaboration Toolbox Is Available

Before calling any of the Authentication Manager functions, you 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" of *Inside Macintosh: Operating System Utilities*.

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

Determining the Version of the Authentication Manager

To determine the version of the Authentication 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 you are using the Collaboration toolbox on a computer that has a PowerShare server, the function returns the version number of the server in the high-order word of the

response parameter. If the Collaboration toolbox or server is not present and available, the Gestalt function returns 0 for the relevant version number. You can use the constant gestaltOCETB for AOCE Collaboration toolbox version 1.0.

Authentication Using ASDSP

To establish mutual authentication between an initiator and a recipient, you use credentials that you get from the server. When you use ASDSP as the transport mechanism to complete the secure connection, you place these credentials in the appropriate field of the parameter block for the sdspOpen function. ASDSP is discussed in the chapter "AppleTalk Data Stream Protocol" in *Inside Macintosh: Networking*.

To get credentials, follow these steps:

1. Specify an expiration time for the AuthGetCredentials function (page 9-43). It is your responsibility to determine how long you want the connection to be available. Credentials are valid for at most 8 hours after they are returned to an initiator by the server. When you call the AuthGetCredentials function you may use the expiry field to specify a shorter time for credentials to be valid. Two ways to determine your expiration time are as follows:
   - Call the AuthGetUTCTime function (page 9-53) to get the current universal coordinated time (UTC) and an offset. Then, your expiration time is the UTC plus the amount of time, in seconds, that you want the credentials to be valid.
   - If you get credentials often, you may choose to remember the time provided by the AuthGetUTCTime function when you first call it and then add the results of the GetDateTime function to that time along with the amount of time, in seconds, that you want the credentials to be valid. Remembering the UTC makes it unnecessary to call the AuthGetUTCTime function each time you need credentials. The GetDateTime function is described in *Inside Macintosh: Operating System Utilities*.
2. Determine the initiator's identity and the recipient's record ID. You can use either the local identity or a specific identity for the initiator. A background application can get the local identity by calling the AuthGetLocalIdentity function (page 9-28). A foreground application can call the PromptForIdentity function, which is described in the chapter "Standard Catalog Package" in this book.

   To get a specific identity for an initiator, first call the AuthPasswordToKey function (page 9-21), providing the record ID and password for the initiator, to get the initiator's client key. Then call the AuthBindSpecificIdentity function (page 9-39) to get the specific identity.

   You must provide your own means for obtaining the recipient's record ID.

3. Call the `AuthGetCredentials` function to get credentials. The Authentication Manager expects you to provide the expected length of the credentials, as well as a pointer to a memory block for the credentials. A buffer three times the size of a packed record ID is usually sufficient for credentials. Use the `kPackedRecordIDMaxBytes` constant defined in the chapter "AOCE Utilities" in this book to determine the size of a packed record ID.
4. To use the ASDSP transport mechanism, call the Device Manager's `PBControl` function using the `SDSPParamBlock` parameter block defined in *Inside Macintosh*: *Networking*.

Authentication for Non-ASDSP Users

To establish mutual authentication between users without using ASDSP, first complete steps 1 through 3 of "Authentication Using ASDSP" on page 9-12. Then continue as indicated in the following sections.

The Initiator's Authentication Process

To complete the authentication process as an initiator, follow these steps. Note that you must devise your own protocol for exchanging the challenges and replies.

1. Call the `AuthMakeChallenge` function (page 9-55) to make a challenge. You provide a buffer and a buffer size. The buffer must be at least 8 bytes in length. The `AuthMakeChallenge` function returns the encrypted challenge in the buffer you supplied, and also returns the actual length of the challenge.
2. Send the credentials and challenge to the specified recipient, using the available transport mechanism.
3. Obtain the challenge reply from the recipient. The challenge reply includes both the reply to your challenge and a counterchallenge from the recipient (steps 5 and 6 in "Steps in the Authentication Process" beginning on page 9-5).
4. Call the `AuthVerifyReply` function (page 9-58) to verify the reply sent by the recipient and to generate a reply to the recipient's counterchallenge. You provide the session key that was supplied by the server as well as the challenge and challenge length returned by the `AuthMakeChallenge` function. You also provide the reply and reply buffer length sent by the recipient. If the `AuthVerifyReply` function finds that the recipient's reply was not valid, it returns an error and does not generate a reply to the counterchallenge.
5. If there was no error, then send the counterchallenge reply generated by the `AuthVerifyReply` function to the recipient.

The Recipient's Authentication Process

To complete authentication as a recipient, follow these steps:

1. Call the `AuthDecryptCredentials` function (page 9-59), passing it the credentials sent by the initiator. The function returns the session key, the issue and expiration times, and the record ID for the initiator. It is your responsibility to ensure that the times are acceptable for your application. Additionally, if there is an intermediary and you provide a pointer to a record ID for it, the `AuthDecryptCredentials` function provides the intermediary's record ID to you.
2. Call the `AuthMakeReply` function (page 9-56) to generate a reply to the challenge received from the initiator and to issue a challenge in return. The `challenge` pointer and `challengeLength` fields are received from the initiator and supplied to this function. The `reply` field contains the reply generated by the function and also the counterchallenge generated by the function.
3. Send this challenge reply and the counterchallenge to the initiator.
4. Obtain the counterchallenge reply from the initiator.
5. Call the `AuthVerifyReply` function to verify the reply sent by the initiator. You provide the session key that was supplied by the server with the credentials, the challenge and challenge length that you sent to the initiator, a pointer to the reply buffer, and the length of the reply.

Authentication Using a Proxy

To use a proxy to authenticate a connection, you request and receive a proxy and then give the proxy to an intermediary, who then uses the proxy to obtain credentials. After the intermediary obtains the credentials, it uses them to create an authenticated connection in the standard way, as described previously.

To obtain and use a proxy, follow these steps:

1. Call the `AuthMakeProxy` function (page 9-45). You must specify the identity of the initiator who wants to create a proxy, the record ID of the recipient with whom the intermediary wishes to communicate, and the record ID of the intermediary. Additionally, you provide times that you want the proxy to be become valid and to expire, a pointer to the buffer into which the `AuthMakeProxy` function will place the proxy, and the length of the buffer. A buffer twice the size of a packed record ID is usually sufficient for the proxy. The `kPackedRecordIDMaxBytes` constant, described in the chapter "AOCE Utilities" in this book, defines the maximum size of a packed record ID.
2. Send the proxy and the recipient record ID to the intermediary.
3. The intermediary calls the `AuthTradeProxyForCredentials` function (page 9-47), supplying the pointer to the proxy buffer and the buffer length. It also supplies its own identity and the recipient's record ID. The intermediary provides a pointer to the credentials and the expected length of the credentials. A buffer three times the size of a packed record ID is usually sufficient for credentials.

Using the Notification Queue

You can add your application's notification callback routine to a notification queue so that it is notified when the local identity is locked or unlocked. When you no longer need to know the status of the local identity, you can remove your callback routine from the notification queue. The `DoNoteQueue` routine in Listing 9-1 checks for a local identity and, if there is one, saves it in a global variable. It installs the SurfWriter application's notification callback routine in the notification queue, which informs it if the status of the local identity changes. Finally, the `DoNoteQueue` routine removes the callback routine from the queue.

If the local identity is locked and your application runs in the foreground, you should disable any functions or commands that require the user to be authenticated. You can then prompt the user to unlock or set up the local identity. If the application runs in the background, you would probably postpone some operations until the local identity is unlocked**.**

To install an application in or remove an application from the notification queue, you first set up the header block, as shown in the `DoInitializeASPB` function in Listing 9-1. Both the `DoInstallNotificationProc` function and `DoRemoveNotificationProc` function call the `DoInitializeASPB` function and then initialize the remaining fields for their respective functions.

The `MyNotificationProc` function in Listing 9-1 is a sample notification routine for the `AuthAddToLocalIdentityQueue` and `AuthRemoveFromLocalIdentityQueue` functions (page 9-30 and page 9-31). The `MyNotificationProc` callback routine is described on page 9-69.

In Listing 9-1, the notification routine updates a flag in the application's global data (the `identityIsLocked` field in the `MyClientData` structure) to notify the SurfWriter application when access to the PowerTalk Setup catalog is locked or unlocked. If the `identityIsLocked` field has the value `true`, the identity might be locked or might not be set up.

Listing 9-1 Using the notification queue

```
/* function to initialize header block */
pascal void DoInitializeASPB(AuthParamBlock *aspb)
   {
   *(long *)&aspb->header.serverHint = 0; /* set up serverHint */
   aspb->header.identity = 0;             /* identity setup */
   aspb->header.dsRefNum = kRefNumUnknown; /* refNum specifier */
   }
/* function to install an application's notification proc in the queue */
pascal OSErr DoInstallNotificationProc(NotificationProc notificationProc,
                                       AuthNotifications notifyFlags,
                                       StringPtr appName,
                                       long clientData)
```

```
   {
   OSErr err;
   AuthParamBlock aspb;
   DoInitializeASPB(&aspb); /* initialize header block */
   aspb.header.clientData = clientData;
   aspb.localIdentityQInstallPB.fNotificationProc=notificationProc;
   aspb.localIdentityQInstallPB.notifyFlags = notifyFlags;
   aspb.localIdentityQInstallPB.appName = appName;
   err = AuthAddToLocalIdentityQueue(&aspb, false);
   return err;
   }
/* function to remove an application's notification proc from the queue */
pascal OSErr DoRemoveNotificationProc(NotificationProc notificationProc)
   {
   OSErr err;
   AuthParamBlock aspb;
   InitializeASPB(&aspb); /* Initialize header block */
   aspb.localIdentityQInstallPB.fNotificationProc=nNotificationProc;
   err = AuthRemoveFromLocalIdentityQueue(&aspb, false);
   return err;
   }
struct MyClientData {
   LocalIdentity localID;
   Boolean identityIsLocked;
   };

pascal OSErr MyGetLocalIdentity(LocalIdentity *localID)
   {
   OSErr err;
   AuthParamBlock aspb;
   DoInitializeASPB(&aspb); /* Initialize header block */
   err = AuthGetLocalIdentity(&aspb, false);
   if (err == noErr)
      *localID = aspb.getLocalIdentityPB.theLocalIdentity;
   return err;
   }


/* notification procedure for your application */
pascal Boolean MyNotificationProc(long clientData,
                          AuthLocalIdentityOp callValue,
                          AuthLocalIdentityLockAction actionValue,
                          LocalIdentity identity)
```

```
   {
   struct MyClientData *myClientData = (struct MyClientData *)clientData;
   if ((callValue == kAuthLockLocalIdentityOp) &&
            (actionValue == kAuthLockWillBeDone)) {
      myClientData->identityIsLocked = true;
      myClientData->localID = 0;
      }
   else
      if (callValue == kAuthUnlockLocalIdentityOp) {
         myClientData->identityIsLocked = false;
         myClientData->localID = identity;
         }
   return false;  /* the sample app never denies a lock pending */
   }


DoNoteQueue () /* using the notification queue for your application */
   {
   OSErr err;
   struct MyClientData myClientData;

   err = MyGetLocalIdentity(&myClientData.localID);
   if (err == noErr)
      myClientData.identityIsLocked = false; /* the function returned a
                                                local identity, therefore
                                                it's not locked */
   else {
      myClientData.identityIsLocked = true;  /* it's either not set up or
                                                else locked */
      /* Set up the local ID if app is not in background, or else wait for
          local ID to be set up and unlocked.  If the latter, when the local
          ID is unlocked, you can get the local identity by looking at
          the localID field in MyClientData. */
      }

   err = DoInstallNotificationProc(
                     MyNotificationProc, kNotifyLockMask|kNotifyUnlockMask,
                     "\pSurfWriter", (long)&myClientData);

   /* ... perform your application's functions */
```

```
/* If identityIsLocked is true, postpone some operations until local ID
    becomes unlocked. */

RemoveNotificationProc(MyNotificationProc);
}
```

Authentication Manager Reference

This section describes the data structures and routines provided by the Authentication Manager.

Data Structures

This section describes the data structures that are specific to the Authentication Manager. See the chapter "AOCE Utilities" for descriptions of other data structures that you use to provide information to or obtain information from Authentication Manager routines.

Parameter Block Header

Each Authentication Manager routine takes, as input, a pointer to a parameter block of type `AuthParamBlockPtr`. This parameter block defines a union of substructures, each of which is a parameter block for one of the Authentication Manager functions. See the descriptions of individual routines, beginning on page 9-20, for a listing of fields in the corresponding parameter blocks. Each of these parameter blocks has the following header:

```
#define AuthDirParamHeader
   Ptr            qLink;         /* reserved */
   long           reserved1;     /* reserved */
   long           reserved2;     /* reserved */
   ProcPtr        ioCompletion;  /* your completion routine */
   OSErr          ioResult;      /* result code */
   unsigned long  saveA5;        /* reserved */
   short          reqCode;       /* reserved */
   long           reserved[2];   /* reserved */
   AddrBlock      serverHint;    /* PowerShare server AppleTalk
                                    addr */
   short          dsRefNum;      /* Set to kRefNumUnknown */
   unsigned long  callID;        /* reserved */
   AuthIdentity   identity;      /* initiator's identity */
   long           gReserved1;    /* reserved */
```

```
long            gReserved2;    /* reserved */
long            gReserved3;    /* reserved */
long            clientData;    /* you define this field */
```

Field descriptions

`qLink`	Reserved.
`reserved1`	Reserved.
`reserved2`	Reserved.
`ioCompletion`	A pointer to a completion routine that you can provide. If you call an Authentication Manager routine asynchronously, it calls your completion function upon returning. Set this field to `nil` if you do not wish to provide a completion routine. The function ignores this field if you call it synchronously.
`ioResult`	The result of the routine. When you execute the routine asynchronously, the Authentication Manager sets this field to 1 as soon as it queues the routine for execution. When the routine completes execution, the Authentication Manager sets this field to the result code.
`saveA5`	Reserved.
`reqCode`	Reserved.
`reserved[2]`	Reserved.
`serverHint`	The AppleTalk address of the PowerShare server to which you want to direct your request. Normally, you specify the value 0 for all fields of this structure, and the Authentication Manager directs the request to an appropriate PowerShare server. The `AddrBlock` data structure is described in *Inside Macintosh: Networking*.
`dsRefNum`	The personal catalog reference number. Because the Authentication Manager works only with server-based catalogs, you must set this parameter to the value `kRefNumUnknown` for all Authentication Manager functions.
`callID`	Reserved.
`identity`	The authentication identity of the entity calling a function. The authentication identity can be either a local identity, a specific identity, or 0 for guest access. The PowerShare server or CSAM uses the identity to determine if the requestor has the access privileges necessary to perform the requested operation. Functions that fail because of insufficient access privileges return either the `kOCEReadAccessDenied` or `kOCEWriteAccessDenied` result code. The `AuthGetLocalIdentity` function described on page 9-28 returns the local identity, and the `AuthBindSpecificIdentity` function described on page 9-39 returns a specific identity. See the chapter “Catalog Manager” in this book for more information about access controls.
`gReserved1`	Reserved.
`gReserved2`	Reserved.
`gReserved3`	Reserved.

`clientData` Available for your use. The Authentication Manager passes the value in this field to your completion or callback routine. If you use the same completion routine to process more than one asynchronous request, for example, your routine can use the `clientData` field to determine for which request it is processing results. You may also use this field to store a pointer to your application's private data.

The Key Structures

Keys are translated passwords used in cryptographic algorithms. See "Keys" on page 9-4. The client keys and session keys used by some Authentication Manager functions are defined by a structure of type `AuthKey`.

```
typedef unsigned long AuthKeyType;
typedef Byte RC4Key[kRC4KeySizeInBytes];
struct AuthKey {/* key type followed by its data */
   AuthKeyType keyType;
   union {
      DESKey des;
      RC4Key rc4;
   }u;
};
typedef AuthKey *AuthKeyPtr;
struct DESKey {/* A DES key is 8 bytes of data */
   unsigned long a;
   unsigned long b;
};
```

Authentication Manager Functions

This section describes functions provided by the Authentication Manager for your use. These functions make it possible for you to manage keys, local identities, specific identities, and credentials; resolve creation IDs; obtain universal coordinated time; implement non-ASDSP authentication, and manage the PowerTalk Setup catalog.

Note

As is generally true, to ensure that asynchronously called functions operate correctly, you must allocate nonrelocatable memory for all parameter blocks and any buffers required for the function. u

Assembly-Language Interface

To call an Authentication Manager function from assembly language, push the address of the `AuthParamBlock` parameter block and the `async` flag onto the stack using the Pascal calling convention, and place the appropriate routine selector value in register D0. Then invoke the `_oceTBDispatch` trap. Each function description includes the selector value for that function. The function returns its result code in the `ioResult` field of the parameter block.

Key Management

The Authentication Manager provides functions to

- translate a password into a key (`AuthPasswordToKey`)
- add a key to a server-based catalog (`AuthAddKey`)
- change a key in a server-based catalog (`AuthChangeKey`)
- delete a key from a server-based catalog (`AuthDeleteKey`)

The three functions that communicate with the server are subject to the access controls specified in the record of the entity for whom you're making the request. Access controls are discussed in the chapter "Catalog Manager" in this book**.**

Note

These functions operate only on client keys, not on session keys. Session keys are created by servers and are valid only for a limited time period. See "Keys" on page 9-4. u

AuthPasswordToKey

The `AuthPasswordToKey` function translates a password string into a client key.

```
pascal OSErr AuthPasswordToKey (AuthParamBlockPtr paramBlock,
                                Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	userRecord	RecordIDPtr	Target's record ID
↔	key	AuthKeyPtr	Target's key
→	password	RStringPtr	Target's password

See page 9-19 for descriptions of the ioCompletion and ioResult fields.

Field descriptions

userRecord	A pointer to the record ID of the user or service for which you want a client key.
key	A pointer to an AuthKey structure you allocate. The function places the key in this structure.
password	A pointer to the password string of the user or service whose record ID you specified in the userRecord parameter. Passwords must be at least 5 bytes and not more than 255 bytes.

DESCRIPTION

The AuthPasswordToKey function creates a new key from a new or changed password. The Authentication Manager returns the key to your local computer only; it does not store the key on the server.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$020A

RESULT CODES

noErr	0	No error
kOCEParamErr	–50	Password too long
kOCEUndesirableKey	–1556	Password too short or resulting key is undesirable

SEE ALSO

The AuthKey structure is described in "The Key Structures" on page 9-20.

The AuthPasswordToKey function is used in an example in "Authentication Using ASDSP" on page 9-12.

The AuthAddKey function is discussed next.

The AuthChangeKey function is described on page 9-24.

The AuthDeleteKey function is described on page 9-26.

AuthAddKey

The AuthAddKey function adds a key for an authentication client to the server-based catalog.

```
pascal OSErr AuthAddKey (AuthParamBlockPtr paramBlock,
                         Boolean async);
```

paramBlock
A pointer to a parameter block.

async A value that specifies whether the function is to be executed asynchronously. Set this parameter to true if you want the function to be executed asynchronously.

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	identity	AuthIdentity	Initiator's identity
→	userRecord	RecordIDPtr	Target's record ID
→	userKey	AuthKeyPtr	Target's key
→	password	RStringPtr	Target's password

See page 9-19 for descriptions of the ioCompletion, ioResult, and identity fields.

Field descriptions

userRecord A pointer to the record ID of the user or service whose key you are adding to a catalog.

userKey A pointer to the new key you are providing.

password A pointer to the password string of the user or service whose key you are providing. Specify nil for this field if you are not providing a password. If you provide a password, the Authentication Manager checks that the key was properly translated from the password before adding the key to the catalog.

DESCRIPTION

During the authentication process, the authentication server encrypts data using the keys of both the initiator and the recipient. For this reason, the server must store the key of every user of the system.

You must provide an identity to this function so that the server can check whether the caller has permission to add a key to the user's record.

Call the AuthPasswordToKey function before calling the AuthAddKey function to obtain a key corresponding to the user's password.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0207

RESULT CODES

noErr	0	No error
kOCEAWriteAccessDenied	–1541	Write access denied
kOCEKeyAlreadyRegistered	–1554	A key already exists
kOCEMalFormedKey	–1555	Key not derived properly from password
kOCEUnknownID	–1567	Identity passed is not valid
kOCENotLocal	–1610	Internal AOCE error
kOCETargetDirectoryInaccessible	–1613	Catalog server not responding
kOCENoSuchDNode	–1615	The dNode was not found
kOCEBadRecordID	–1617	Name and type incorrect for creation ID
kOCENoSuchRecord	–1618	No such record
kOCEStreamCreationErr	–1625	An error occurred in creating the stream

SEE ALSO

The use of keys in the authentication process is described in "Steps in the Authentication Process" beginning on page 9-5.

Access controls are discussed in the chapter "Catalog Manager" in this book.

Use the `AuthPasswordToKey` function (page 9-21) to create a key.

Use the `AuthChangeKey` function, described next, to replace a key already stored in the server-based catalog.

AuthChangeKey

The `AuthChangeKey` function changes a user's key stored in a server-based catalog.

```
pascal OSErr AuthChangeKey (AuthParamBlockPtr paramBlock,
                            Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`identity`	`AuthIdentity`	Initiator's identity
→	`userRecord`	`RecordIDPtr`	Target's record ID
→	`userKey`	`AuthKeyPtr`	Target's key
→	`password`	`RStringPtr`	Target's password

See page 9-19 for descriptions of the `ioCompletion`, `ioResult`, and `identity` fields.

Field descriptions

`userRecord` A pointer to the record ID of the user or service whose changed key you are storing in a catalog.

`userKey` A pointer to the new key you are providing.

`password` A pointer to the password string of the user or service whose key you are providing. Specify `nil` for this field if you are not providing a password. If you provide a password, the Authentication Manager checks that the key was properly translated from the password before adding the key to the catalog.

DESCRIPTION

Call the `AuthChangeKey` function when a password has been changed and you need to store a new key in a server-based catalog. Call the `AuthPasswordToKey` function before calling the `AuthChangeKey` function to obtain a key corresponding to the new password.

You must provide an identity to this function so that the server can verify that the caller has permission to change a key in the user's record.

SPECIAL CONSIDERATIONS

If you change a key for a user or service and later attempt to use a local or specific identity that was created using the old key, the function may fail. It is important to update identities when changes are made to the passwords and therefore to the keys. Before executing some functions, the Collaboration toolbox communicates with the server to check identities and keys relative to each other.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0208

RESULT CODES

noErr	0	No error
kOCEAWriteAccessDenied	–1541	Write access denied
kOCENoKeyFound	–1550	No key was found
kOCEMalFormedKey	–1555	Key not derived properly from password
kOCEUnknownID	–1567	Identity passed is not valid
kOCENotLocal	–1610	Internal AOCE error
kOCETargetDirectoryInaccessible	–1613	Catalog server not responding
kOCENoSuchDNode	–1615	The dNode was not found
kOCEBadRecordID	–1617	Name and Type incorrect for creation ID
kOCENoSuchRecord	–1618	No such record
kOCEStreamCreationErr	–1625	An error occurred in creating the stream

SEE ALSO

Use the `AuthBindSpecificIdentity` function (page 9-39) to update an identity when you change a key.

The `AuthPasswordToKey` function is described on page 9-21.

The `AuthAddKey` function is discussed on page 9-23.

The `AuthDeleteKey` function is described next.

AuthDeleteKey

Call the `AuthDeleteKey` function to delete a key for a specified authentication client from the server-based catalog.

```
pascal OSErr AuthDeleteKey (AuthParamBlockPtr paramBlock,
                            Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	identity	AuthIdentity	Initiator's identity
→	userRecord	RecordIDPtr	Target's record ID

See page 9-19 for descriptions of the `ioCompletion`, `ioResult`, and `identity` fields.

Field descriptions

`userRecord`	A pointer to the record ID of the user or service whose key is to be deleted.

DESCRIPTION

Call the `AuthDeleteKey` function to remove a `key` from the server-based catalog.

If you wish a new key to take the place of the deleted one in the server-based catalog, you can call the `AuthPasswordToKey` function and then the `AuthAddKey` function.

SPECIAL CONSIDERATIONS

When you remove a key for a user or service from the server-based catalog, the Authentication Manager can no longer create an authentication identity for that user or service, build credentials, or have others build credentials to authenticate connections to the user or service.

If you remove a key for a user and then later attempt to use a local or specific identity that was created using the key, the function may fail. It is important to update identities when changes are made to passwords and therefore to keys. Identities and keys are checked relative to each other before some functions are allowed.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0209

RESULT CODES

`noErr`	0	No error
`kOCEAWriteAccessDenied`	–1541	Write access denied
`kOCENoKeyFound`	–1550	No key was found
`kOCEUnknownID`	–1567	Identity passed is not valid
`kOCENotLocal`	–1610	Internal AOCE error
`kOCETargetDirectoryInaccessible`	–1613	Catalog server not responding
`kOCENoSuchDNode`	–1615	The dNode was not found
`kOCEBadRecordID`	–1617	Name and type incorrect for creation ID
`kOCENoSuchRecord`	–1618	No such record
`kOCEStreamCreationErr`	–1625	An error occurred in creating the stream

SEE ALSO

The `AuthPasswordToKey` function is described on page 9-21.

The `AuthAddKey` function is discussed on page 9-23

The AuthChangeKey function is described on page 9-24.

Use the AuthBindSpecificIdentity function (page 9-39) to update an identity when you change a key.

Local Identity Management

A local identity provides transparent access to the PowerTalk Setup catalog: it gives the user access to all catalogs and services in the PowerTalk Setup catalog without the user having to log on to each one individually. Any AOCE Catalog Manager or Authentication Manager function that requires an identity parameter can use a local identity. See "Local Identities" on page 9-8 for a discussion of local identities.

The Authentication Manager provides functions that you can use to

- get the local identity number (AuthGetLocalIdentity)
- add an application to the local identity notification queue (AuthAddToLocalIdentityQueue)
- remove an application from the local identity notification queue (AuthRemoveFromLocalIdentityQueue)

The Authentication Manager also provides functions that the PowerTalk Key Chain uses to

- set up the local identity (AuthSetupLocalIdentity)
- change the local identity (AuthChangeLocalIdentity)
- lock the local identify (AuthLockLocalIdentity)
- unlock the local identity (AuthUnlockLocalIdentity)
- remove the local identity (AuthRemoveLocalIdentity)

AuthGetLocalIdentity

Call the AuthGetLocalIdentity function to get the local identity.

```
pascal OSErr AuthGetLocalIdentity (AuthParamBlockPtr paramBlock,
                                   Boolean async);
```

paramBlock
: A pointer to a parameter block.

async
: A value that specifies whether the function is to be executed asynchronously. Set this parameter to true if you want the function to be executed asynchronously.

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
←	theLocalIdentity	LocalIdentity	The local identity

See page 9-19 for descriptions of the ioCompletion and ioResult fields.

Field descriptions

theLocalIdentity
The local identity.

DESCRIPTION

You can call the AuthGetLocalIdentity function to obtain the local identity. If the local identity has not been set up, the AuthGetLocalIdentity function returns a kOCEOCESetupRequired result code. If the local identity is locked, the AuthGetLocalIdentity function returns a kOCELocalAuthenticationFail result code.

If your application is not a background application, you can call the SDPPromptForID function to prompt the user to unlock the local identity.

If your application runs only in the background, you can register with the Authentication Manager using the AuthAddToLocalIdentityQueue function. Then your application is notified when the local identity is created or unlocked.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0204

RESULT CODES

noErr	0	No error
kOCELocalAuthenticationFail	–1561	Local identity locked
kOCESetupRequired	–1633	Setup of local identity required

SEE ALSO

The AuthGetLocalIdentity function is used in an example in the section "Authentication Using ASDSP" on page 9-12.

The SDPPromptForID function is described in the chapter "Standard Catalog Package" in this book.

The AuthAddToLocalIdentityQueue function is discussed next.

AuthAddToLocalIdentityQueue

Call the `AuthAddToLocalIdentityQueue` function to add an application to the Authentication Manager's local identity notification queue.

```
pascal OSErr AuthAddToLocalIdentityQueue
              (AuthParamBlockPtr paramBlock, Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`clientData`	`long`	For your use
→	`notifyProc`	`NotificationProc`	Notification function
→	`notifyFlags`	`AuthNotifications`	Notification flags
→	`appName`	`StringPtr`	Application name

See page 9-19 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`clientdata` A value for your use. The Authentication Manager passes the value in this field to your notification routine.

`notifyProc` A pointer to your notification routine. You must provide a notification routine to be called by the notification queue.

`notifyFlags` A flag byte that specifies when you want your notification routine to be called: when the local identity is about to be locked, when it is unlocked, when the user changes the name in the PowerTalk Key Chain, or for some combination of these events.

`appName` A pointer to the name of your application.

DESCRIPTION

You call the `AuthAddToLocalIdentityQueue` function to add your notification routine to the Authentication Manager's notification queue.

You set the `notifyFlags` field to specify when you want your notification routine called. Possible values for this field are as follows:

```
enum {kNotifyLockBit, kNotifyUnlockBit, kNotifyNameChangeBit};
enum
   {kNotifyLockMask     = 1L << kNotifyLockBit,
```

```
   kNotifyUnlockMask    = 1L << kNotifyUnlockBit
   kNotifyNameChangeMask= 1L << kNotifyNameChangeBit
};
```

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0205

RESULT CODES

noErr	0	No error

SEE ALSO

For an example of the use of the `AuthAddToLocalIdentityQueue` function, see Listing 9-1 on page 9-15.

The notification routine is described on page 9-69.

AuthRemoveFromLocalIdentityQueue

Call the `AuthRemoveFromLocalIdentityQueue` function to remove your notification routine from the Authentication Manager's notification queue.

```
pascal OSErr AuthRemoveFromLocalIdentityQueue
                 (AuthParamBlockPtr paramBlock, Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	notifyProc	NotificationProc	Notification function

See page 9-19 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`notifyProc` The notification routine you provided when you called the `AuthAddToLocalIdentityQueue` function.

DESCRIPTION

You call the `AuthRemoveFromLocalIdentityQueue` function to remove your notification routine from the Authentication Manager's notification queue. The Authentication Manager informs the routines in the notification queue of changes in the state of the local identity access.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0206

RESULT CODES

`noErr`	0	No error

SEE ALSO

For an example of the use of the `AuthRemoveFromLocalIdentityQueue` function, see Listing 9-1 on page 9-15.

You use the `AuthAddToLocalIdentityQueue` function (page 9-30) to add a routine to the notification queue.

The notification procedure is described on page 9-69.

AuthSetupLocalIdentity

The `AuthSetupLocalIdentity` function sets up the user name and password for the local identity.

```
pascal OSErr AuthSetupLocalIdentity (AuthParamBlockPtr paramBlock,
                                     Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`userName`	`RStringPtr`	The user name
→	`password`	`RStringPtr`	The user password

See page 9-19 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`userName`	The name of the principal user of the local computer.
`password`	The password to assign to the principal user of the local computer.

DESCRIPTION

You can use this function to set up the user name and password for the local identity. Normally, however, the user sets up a local identity by specifying a name and password in the PowerTalk Key Chain. You can call the `SDPPromptForID` function to prompt a user for a password to unlock the local identity when necessary.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0216

RESULT CODES

`noErr`	0	No error
`kOCELocalIdentitySetupExists`	–1562	Local identity setup exists, use `AuthChangeLocalIdentity` instead

SEE ALSO

You can use the `AuthGetLocalIdentity` function (page 9-28) to obtain a local identity once it has been set up.

Use the `SDPPromptForID` function, which is described in the chapter "Standard Catalog Package" in this book, to prompt the user for a name and password to unlock the local identity. This function also returns the local identity.

AuthChangeLocalIdentity

The `AuthChangeLocalIdentity` function changes the password for the local identity.

```
pascal OSErr AuthChangeLocalIdentity
                                    (AuthParamBlockPtr paramBlock,
                                     Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	userName	RStringPtr	The user name
→	password	RStringPtr	The user password
→	newPassword	RStringPtr	The new user password

See page 9-19 for descriptions of the ioCompletion and ioResult fields.

Field descriptions

userName	The name of the principal user of the local computer.
password	The current password for the principal user of the local computer.
newPassword	The new password you want to assign to the principal user of the local computer.

DESCRIPTION

You can use this function to change the password for the local identity from within your application. Normally, however, the user uses the PowerTalk Key Chain to change the password for the local identity.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0217

RESULT CODES

noErr	0	No error
kOCELocalAuthenticationFail	–1561	Local identity locked
kOCEOCESetupRequired	–1633	Setup of local identity required

SEE ALSO

You can use the AuthSetupLocalIdentity function (page 9-32) to set up a local identity.

AuthLockLocalIdentity

The `AuthLockLocalIdentity` function locks the local identity.

```
pascal OSErr AuthLockLocalIdentity (AuthParamBlockPtr paramBlock,
                                    Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`theLocalIdentity`	`LocalIdentity`	The local identity
←	`appName`	`StringPtr`	The name of the application that denied locking (if any)

See page 9-19 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`theLocalIdentity`
The local identity.

`appName` The name of the application that denied locking, if the function fails and returns the `kOCEOperationDenied` result code. Allocate a pointer to a `Str31` data type for this parameter.

DESCRIPTION

To lock the local identity, a user can choose the Lock Key Chain command from the Special menu of the Finder or set the PowerTalk Setup control panel to lock the Key Chain after some specified period of inactivity. You can use the `AuthLockLocalIdentity` function to lock the local identity from within your application.

When you call the `AuthLockLocalIdentity` function, the Authentication Manager calls every routine in its notification queue to give it an opportunity to deny the lock operation. If any application denies the operation, the `AuthLockLocalIdentity` function returns the `kOCEOperationDenied` result code and the `appName` field points to the name of the application that denied the locking operation.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0215

RESULT CODES

noErr	0	No error
kOCEOperationDenied	–1568	Local identity operation denied

SEE ALSO

The notification queue is described in "Local Identity Status Notification" on page 9-9.

You use the AuthAddToLocalIdentityQueue function (page 9-30) to add a routine to the notification queue.

You can use the AuthUnlockLocalIdentity function (described next) to unlock a local identity.

AuthUnlockLocalIdentity

Call the AuthUnlockLocalIdentity function to unlock the local identity.

```
pascal OSErr AuthUnlockLocalIdentity
                                    (AuthParamBlockPtr paramBlock,
                                     Boolean async);
```

paramBlock
A pointer to a parameter block.

async A value that specifies whether the function is to be executed asynchronously. Set this parameter to true if you want the function to be executed asynchronously.

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	theLocalIdentity	LocalIdentity	The local identity
→	userName	RStringPtr	The name of the user
→	password	RStringPtr	The user password

See page 9-19 for descriptions of the ioCompletion and ioResult fields.

Field descriptions

`theLocalIdentity`
The local identity.

`userName` The name of the principal user of the local computer.

`password` The password for the principal user of the local computer.

DESCRIPTION

To unlock a local identity, the user can choose the Unlock Key Chain command from the Finder's Special menu. You can also call the `SDPPromptForID` function to prompt the user for a password and unlock the local identity. Alternatively, you can use the `AuthUnlockLocalIdentity` function to unlock the local identity from within your application. If the local identity does not exist, this function creates one.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0214

RESULT CODES

`noErr`	0	No error
`kOCEOCESetupRequired`	–1633	Setup of local identity required

SEE ALSO

The `AuthLockLocalIdentity` function is described on page 9-35.

The `AuthSetupLocalIdentity` function is described on page 9-32.

The `SDPPromptForID` function is described in the chapter "Standard Catalog Package" in this book.

AuthRemoveLocalIdentity

Call the `AuthRemoveLocalIdentity` function to remove the local identity.

```
pascal OSErr AuthRemoveLocalIdentity
                                    (AuthParamBlockPtr paramBlock,
                                     Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	userName	RStringPtr	The name of the user.
→	password	RStringPtr	The user password.

See page 9-19 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

userName	The name of the principal user of the local computer
password	The password for the principal user of the local computer

DESCRIPTION

Normally, a user cannot remove a local identity from a PowerTalk system without replacing it with a new local identity or reinstalling the PowerTalk system software. The user normally uses the Key Chain to change a local identity. You can use the `AuthRemoveLocalIdentity` function to remove the local identity, effectively rendering the Key Chain inoperable. The user then is prompted to set up a local identity the next time he or she attempts to use the PowerTalk system software.

IMPORTANT

Because removing the local identity disrupts the use of the PowerTalk system software on the user's computer, warn users before allowing them to remove a local identity. s

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0218

RESULT CODES

noErr	0	No error
kOCELocalAuthenticationFail	–1561	Local identity locked
kOCEOCESetupRequired	–1633	Setup of local identity required

SEE ALSO

To lock a local identity so that the user must enter the password before using PowerTalk, use the `AuthLockLocalIdentity` function (page 9-35).

Specific Identity Management

A specific identity is a shorthand representation for the name and key of an alternate user. See “Specific Identities” on page 9-9 for a further discussion.

The Authentication Manager provides the following specific identity management services:

- binding a new specific identity number to a user’s record ID and key (`AuthBindSpecificIdentity`)
- unbinding a specific identity number from a user’s record ID and key (`AuthUnbindSpecificIdentity`)
- using a specific identity to get a user’s record ID (`AuthGetSpecificIdentityInfo`)

AuthBindSpecificIdentity

Call the `AuthBindSpecificIdentity` function to bind an identity number to a specified authentication client’s record ID and key.

```
pascal OSErr AuthBindSpecificIdentity
                                    (AuthParamBlockPtr paramBlock,
                                     Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
←	`userIdentity`	`AuthIdentity`	Binding identity
→	`userRecord`	`RecordIDPtr`	Entity’s record ID
→	`userKey`	`AuthKeyPtr`	Entity’s key

See page 9-19 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`userIdentity` The specific identity.

`userRecord` A pointer to the record ID of the authentication client.

`userKey` A pointer to the user or service key for the client.

DESCRIPTION

Call the `AuthBindSpecificIdentity` function to bind an identity to a record ID and key you provide. The Authentication Manager contacts the catalog containing the record identified by the `userRecord` field to verify the name and key. If the name is valid and the key is correct, the `AuthBindSpecificIdentity` function returns an identity.

You can use the identity returned by this function as an input to any AOCE function that requires an identity. The AOCE software uses the identity to check whether the authentication client has sufficient access privileges to do the operation requested.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0200

RESULT CODES

`noErr`	0	No error
`kOCENoKeyFound`	–1550	Client has no key
`kOCEWrongIdentityOrKey`	–1557	Incorrect key for client
`kOCEUnknownID`	–1567	Identity passed is not valid
`kOCENotLocal`	–1610	Internal AOCE error
`kOCETargetDirectoryInaccessible`	–1613	Catalog server not responding
`kOCENoSuchDNode`	–1615	The dNode was not found
`kOCEBadRecordID`	–1617	Name and type incorrect for creation ID
`kOCENoSuchRecord`	–1618	Record ID doesn't exist
`kOCEStreamCreationErr`	–1625	An error occurred in creating the stream

SEE ALSO

The `AuthBindSpecificIdentity` function is used in an example in the section "Authentication Using ASDSP" on page 9-12.

You can use the `AuthPasswordToKey` function (page 9-21) to get a key from a password.

The `AuthUnbindSpecificIdentity` function is described next.

AuthUnbindSpecificIdentity

The `AuthUnbindSpecificIdentity` function unbinds an identity from a user's name and key.

```
pascal OSErr AuthUnbindSpecificIdentity
                (AuthParamBlockPtr paramBlock, Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`userIdentity`	`AuthIdentity`	Binding identity

See page 9-19 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`userIdentity` The identity to be deleted.

DESCRIPTION

Call the `AuthUnbindSpecificIdentity` function to remove permanently an identity you no longer need; for example, when your application quits.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0201

RESULT CODES

`noErr`	0	No error
`kOCENotLocalIdentity`	–1565	You cannot unbind a local identity
`kOCEUnknownID`	–1567	Identity passed is not valid

SEE ALSO

The `AuthBindSpecificIdentity` function is described on page 9-39.

The `AuthGetSpecificIdentityInfo` function (described next) returns the record ID associated with a specific identity.

AuthGetSpecificIdentityInfo

Call the `AuthGetSpecificIdentityInfo` function to get the record ID (but not the user or service key) associated with the specified identity.

```
pascal OSErr AuthGetSpecificIdentityInfo
                                    (AuthParamBlockPtr paramBlock,
                                     Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`userIdentity`	`AuthIdentity`	Identity
↔	`userRecord`	`RecordIDPtr`	Entity's record ID

See page 9-19 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`userIdentity` The identity whose record ID is desired.

`userRecord` A pointer to the record ID structure for the record, in which the record ID is returned.

DESCRIPTION

Call the `AuthGetSpecificIdentityInfo` function to obtain the record ID associated with a particular identity.

The `userRecord` field must contain a pointer to a `recordID` structure of maximum size.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0203

RESULT CODES

`noErr`	0	No error
`kOCENotLocalIdentity`	–1565	You cannot unbind a local identity
`kOCEUnknownID`	–1567	Identity passed is not valid

SEE ALSO

The chapter "AOCE Utilities" in this book describes how to allocate space for a record ID.

The `AuthBindSpecificIdentity` function is described on page 9-39.

The `AuthUnbindSpecificIdentity` function is described on page 9-41.

Credentials Management

Credentials enable initiators and recipients to verify each other's identities. See "Credentials" on page 9-5 for more information. The Authentication Manager provides functions to

- get credentials from the server (`AuthGetCredentials`)
- obtain a proxy with which to get credentials (`AuthMakeProxy`)
- use a proxy to get credentials from the server (`AuthTradeProxyForCredentials`)

AuthGetCredentials

Call the `AuthGetCredentials` function to obtain credentials from the authentication server.

```
pascal OSErr AuthGetCredentials (AuthParamBlockPtr paramBlock,
                                 Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`userIdentity`	`AuthIdentity`	Initiator identity
→	`recipient`	`RecordIDPtr`	Record ID of recipient
↔	`sessionKey`	`AuthKeyPtr`	Session key
↔	`expiry`	`UTCTime`	Desired / actual times
↔	`credentialsLength`	`unsigned long`	Buffer size and credentials size
↔	`credentials`	`Ptr`	Credentials buffer

See page 9-19 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`userIdentity`	The identity of the initiator.
`recipient`	A pointer to the record ID of the recipient.
`sessionKey`	A pointer to a buffer that you supply to the function. The function puts the session key into this buffer.
`expiry`	When you call the function, you use the `expiry` field to specify the time at which you want the credentials to expire. When the function completes, this field specifies the actual expiration time: your desired expiration time or the current time plus 8 hours, whichever is sooner.
`credentialsLength`	When you call the function, you use this field to specify the size of the buffer pointed to by the `credentials` field. A buffer three times the size of a packed record ID is usually sufficient for credentials. Use the `kPackedRecordIDMaxBytes` constant defined in the chapter "AOCE Utilities" in this book to determine the size of a packed record ID. When the function completes, this field indicates the actual amount of data written into the buffer.
`credentials`	A pointer to the buffer you provide to hold the returned credentials.

DESCRIPTION

Call the `AuthGetCredentials` function to get credentials to establish an authenticated connection with the named recipient. Any entity can request credentials for any other entity.

Your application should call the `AuthGetUTCTime` function before calling the `AuthGetCredentials` function because the expiration time you specify is based on universal coordinated time (UTC). You add the desired number of seconds to the current time returned by the `AuthGetUTCTime` function.

If the `AuthGetCredentials` function is successful, the buffer pointed to by the `credentials` field contains encrypted credentials and the `sessionKey` field contains the key to use during the challenge portion of the authentication process. The credentials returned by the server to the initiator are encrypted in the key of the recipient.

If the buffer you provide is not large enough to hold the credentials, the function returns the `kOCEMoreData` result code. You can increase the buffer size and call the function again.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$020B

RESULT CODES

`noErr`	0	No error
`kOCECredentialsExpired`	–1546	Desired expiration time has passed
`kOCERecipientKeyNotFound`	–1552	The recipient key was not found
`kOCEInitiatorKeyProblem`	–1558	No key, or initiator's key changed
`kOCEUnknownID`	–1567	Identity passed is not valid
`kOCENotLocal`	–1610	Internal AOCE error
`kOCETargetDirectoryInaccessible`	–1613	Catalog server not responding
`kOCENoSuchDNode`	–1615	The dNode was not found
`kOCEBadRecordID`	–1617	Name and type incorrect for creation ID
`kOCEMoreData`	–1623	Buffer was too small to hold all available data
`kOCEStreamCreationErr`	–1625	An error occurred in creating the stream

SEE ALSO

The authentication process is described in "Steps in the Authentication Process" beginning on page 9-5.

The `AuthGetCredentials` function is used in an example in the section "Authentication Using ASDSP" on page 9-12.

The `AuthGetUTCTime` function is discussed on page 9-53.

The `AuthDecryptCredentials` function is discussed on page 9-59.

AuthMakeProxy

Call the `AuthMakeProxy` function to create a proxy.

```
pascal OSErr AuthMakeProxy (AuthParamBlockPtr paramBlock,
                            Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	userIdentity	AuthIdentity	Principal identity
→	recipient	RecordIDPtr	Recipient record ID
→	firstValid	UTCTime	Time proxy becomes valid
→	expiry	UTCTime	Time proxy expires
→	authDataLength	unsigned long	Must be 0
→	authData	Ptr	Must be nil
↔	proxyLength	unsigned long	Buffer size and proxy size
↔	proxy	Ptr	Proxy buffer
→	intermediary	RecordIDPtr	Intermediary record ID

See page 9-19 for descriptions of the ioCompletion and ioResult fields.

Field descriptions

userIdentity	The identity of the user or service for which you are requesting the proxy.
recipient	A pointer to the record ID of the recipient.
firstValid	The time that the proxy is to become valid.
expiry	The last time at which you want the proxy to be valid
authDataLength	Reserved. Set this parameter to 0.
authData	Reserved. Set this parameter to nil.
proxyLength	The length of the buffer to which the proxy field points. A buffer twice the size of a packed record ID is usually sufficient for a proxy. Use the kPackedRecordIDMaxBytes constant defined in the chapter “AOCE Utilities” in this book to determine the size of a packed record ID. The function returns the actual length of the proxy in this parameter.
proxy	A pointer to the proxy buffer, in which the function returns the proxy.
intermediary	A pointer to the record ID of the intermediary that will use the proxy to obtain credentials in your behalf.

DESCRIPTION

Call the AuthMakeProxy function to create a proxy. A proxy is granted to an intermediary for use with a particular recipient during a specified time period only. The AuthMakeProxy function creates a proxy and returns it to you. You can then pass it to an intermediary to use on your behalf. The proxy is valid only until the expiration time you specify in the expiry field. To obtain credentials, the intermediary must call the AuthTradeProxyForCredentials function.

If the function returns a `kOCEMoreData` result code, you can call the `AuthMakeProxy` function again after increasing the buffer size.

SPECIAL CONSIDERATIONS

The Authentication Manager provides no mechanism for sending a proxy from an initiator to an intermediary. You must devise your own mechanism and protocol for this purpose.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0212

RESULT CODES

`noErr`	0	No error
`kOCEMoreData`	–1623	Buffer was too small to hold all available data

SEE ALSO

The `AuthMakeProxy` function is used in an example in the section "Authentication Using a Proxy" on page 9-14.

See "Proxies" on page 9-10 for a discussion of proxies and "Steps in the Authentication Process" beginning on page 9-5 for a description of the authentication process.

The `AuthTradeProxyForCredentials` function is described next.

AuthTradeProxyForCredentials

Call the `AuthTradeProxyForCredentials` function to trade a proxy for credentials.

```
pascal OSErr AuthTradeProxyForCredentials
                  (AuthParamBlockPtr paramBlock, Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	userIdentity	AuthIdentity	Intermediary identity
→	recipient	RecordIDPtr	Recipient name
↔	sessionKey	AuthKeyPtr	Session key
↔	expiry	UTCTime	Credentials expiration times
↔	credentialsLength	unsigned long	Buffer size and credentials size
↔	credentials	Ptr	Credentials buffer
→	proxyLength	unsigned long	Actual proxy size
→	proxy	Ptr	Proxy buffer
→	principal	RecordIDPtr	Record ID of principal

See page 9-19 for descriptions of the ioCompletion and ioResult fields.

Field descriptions

`userIdentity` The identity of the intermediary.

`recipient` A pointer to the record ID of the recipient.

`sessionKey` A pointer to the session key buffer that you supply. The function returns the session key in this buffer.

`expiry` The desired expiration time for the credentials. The function returns the actual expiration time.

`credentialsLength` As an input, the size of the buffer you are providing to hold the returned credentials. Use the `kPackedRecordIDMaxBytes` constant defined in the chapter "AOCE Utilities" in this book to determine the size needed. On return, this field holds the actual size of the credentials.

`credentials` A pointer to the buffer in which the function places the encrypted credentials.

`proxyLength` The size of the proxy.

`proxy` A pointer to the buffer containing the proxy used to get the credentials.

`principal` A pointer to the record ID of the user or service who created the proxy.

DESCRIPTION

Calling the `AuthTradeProxyForCredentials` function is very similar to calling the `AuthGetCredentials` function, except that the creator of the proxy first calls the `AuthMakeProxy` function to obtain a proxy and gives the proxy to an intermediary; then the intermediary calls the `AuthTradeProxyForCredentials` function for credentials. In the `principal` field, you specify the entity who made the proxy.

The expiration time of the credentials depends on the maximum lifetime permitted by the Authentication Manager, the period during which the proxy is valid, and the expiration time you request for the credentials. For example, assume that the proxy has an expiration time of 3:00 P.M. on a given day of a given month of a given year. Assume

all other times in this example are for the same day, month, and year as the proxy expiration time. First, if it is 3:15 P.M. when the intermediary requests credentials, the Authentication Manager refuses the request because the proxy has expired. If, however, the intermediary requests credentials at 5:00 A.M., the credentials expire at 1:00 P.M. even though you requested a 3:00 P.M. expiration, because the server enforces a maximum lifetime for credentials of 8 hours. If you request credentials at any time between 7:01 A.M. and 2:59 P.M., the credentials expire at 3:00 P.M., because credentials must expire at or before the time specified by the proxy expiration time.

You can use the `AuthTradeProxyForCredentials` function to request credentials as many times as you wish during the lifetime of the proxy.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0213

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	No recipient, or invalid recipient dNode
`kOCEAccessRightsInsufficient`	–1542	Intermediary's record ID does not appear in the proxy
`kOCEProxyImmature`	–1547	Proxy not yet valid
`kOCEProxyExpired`	–1548	Proxy has expired
`kOCEDisallowedRecipient`	–1549	Recipient record ID does not appear in proxy
`kOCERecipientKeyNotFound`	–1552	No key found
`kOCEAgentKeyNotFound`	–1553	Intermediary's key not found
`kOCEInitiatorKeyProblem`	–1558	Can't decipher instructions or the principal's key was not found
`kOCEUnknownID`	–1567	Identity passed is not valid
`kOCENotLocal`	–1610	Internal AOCE error
`kOCETargetDirectoryInaccessible`	–1613	Catalog server not responding
`kOCENoSuchDNode`	–1615	The dNode was not found
`kOCEBadRecordID`	–1617	Name and type incorrect for creation ID of recipient or principal
`kOCEMoreData`	–1623	Buffer was too small to hold all available data
`kOCEStreamCreationErr`	–1625	An error occurred in creating the stream

SEE ALSO

The `AuthTradeProxyForCredentials` function is used in an example in the section "Authentication Using a Proxy" on page 9-14.

See “Proxies” on page 9-10 for a discussion of proxies and “Steps in the Authentication Process” beginning on page 9-5 for a description of the authentication process.

The `AuthGetCredentials` function is discussed on page 9-43.

The `AuthMakeProxy` function is discussed on page 9-45.

Creation ID Resolution

Creation IDs are unique identifiers for records. The are described in detail in the chapters “AOCE Utilities” and “Catalog Manager” in this book. The `AuthResolveCreationID` function returns the creation ID of a record with the name and type that you supply. If there are multiple records with the same name and type, then it returns the creation IDs of all of the records that match the name and type.

AuthResolveCreationID

Call the `AuthResolveCreationID` function to obtain all the dNode numbers and creation IDs for all the records that have a given name and type.

```
pascal OSErr AuthResolveCreationID (AuthParamBlockPtr paramBlock,
                                    Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`identity`	`AuthIdentity`	Identity; must be 0
→	`userRecord`	`RecordIDPtr`	A record ID
→	`bufferLength`	`unsigned long`	Buffer size
↔	`buffer`	`Ptr`	Data buffer
←	`totalMatches`	`unsigned long`	Number of matches found
←	`actualMatches`	`unsigned long`	Number of matches returned

See page 9-19 for descriptions of the `ioCompletion`, `ioResult`, and `identity` fields.

Field descriptions

`userRecord` A pointer to the record ID of the entity whose dNode number and creation ID are to be returned. You must specify the name and type for the entity. The `RLI` must include the dNode number or the

	pathname of the dNode in which you expect the user's record to be located. The `cid` field of the record ID must be set to `NULL` before the function is called.
`bufferLength`	The size of the buffer for holding dNode numbers and creation IDs.
`buffer`	A pointer to the buffer to hold dNode numbers and creation IDs.
`totalMatches`	Total number of matching names found in the server catalog.
`actualMatches`	Number of matches returned in the buffer. This number is determined by how many dNode numbers and creation IDs fit in the buffer.

DESCRIPTION

The creation ID is a unique identifier for a given record. If you don't know this identifier but know the record ID, you can determine the creation ID by calling the `AuthResolveCreationID` function. There may be several records with the same name and type. It is the responsibility of your user application to prompt users to choose the record desired from those provided by this function.

In most cases, you should search the Users and Groups folder, which has the dNode number 3, for the record. This folder normally contains the User record or an alias to the User record of every user with an account on the PowerShare server. If the Collaboration toolbox finds a record with the name and type you specify, it returns the dNode number and creation ID of that record. If it finds an alias to a record with the name and type you specify, it resolves the alias and returns the dNode number and creation ID of the original record.

You must set the creation ID of the record ID to `NULL` before calling the `AuthResolveCreationID` function. You do this by calling the `OCESetCreationIDToNull` function.

The server finds all records in the catalog whose name and type match those in the `userRecord` field. Depending on the number of matches, the following results are returned

- Exactly one match: the dNode number and creation ID are put in the buffer.
- More than one match if the buffer is large enough to hold all matches: The buffer contains the dNode numbers and creation IDs of all records with matching names and types. A `kOCEAmbiguousMatches` result code is returned.
- More than one match if the buffer is not large enough to hold all the matches: the `totalMatches` field contains the number of matches that were found in the server catalog. The `actualMatches` field contains how many of the dNode numbers and creation IDs fit in the buffer, and the buffer contains as many dNode numbers and creation IDs as fit, packed one after the other. A `kOCEMoreData` result code is returned.
- No matches: a `kOCENoSuchRecord` result code is returned.

When you have more than one match and the buffer is not large enough, you can call this function again using an appropriately sized buffer. The dNode numbers and creation IDs are loaded into the user buffer in an array the size of the `actualMatches` field.

SPECIAL CONSIDERATIONS

This function does not check access controls. You must pass a 0 in the `identity` field.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0202

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEUnknownID`	–1567	Identity passed is not valid
`kOCEAmbiguousMatches`	–1569	More than one match
`kOCENotLocal`	–1610	Internal AOCE error
`kOCETargetDirectoryInaccessible`	–1613	Catalog server not responding
`kOCENoSuchDNode`	–1615	The dNode was not found
`kOCENoSuchRecord`	–1618	No such record found with creation ID
`kOCEMoreData`	–1623	Buffer was too small to hold all available data
`kOCEStreamCreationErr`	–1625	An error occurred in creating the stream

SEE ALSO

The `OCESetCreationIDToNull` function is described in the chapter “AOCE Utilities” in this book.

Time Service

In a distributed system of many computers, you need a common time for communication. The Authentication Manager provides the universal coordinated time (UTC), also known as Greenwich Mean Time. You can use UTC to specify issue and expiration times for credentials and for other possible uses in your application. Call the `AuthGetUTCTime` function to get the current UTC.

AuthGetUTCTime

The `AuthGetUTCTime` function returns the current universal coordinated time (UTC) that is maintained by a catalog server.

```
pascal OSErr AuthGetUTCTime (AuthParamBlockPtr paramBlock,
                             Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`pRLI`	`PackedRLIPtr`	Packed RLI of the node
←	`theUTCTime`	`UTCTime`	UTC seconds east of Greenwich
←	`theUTCOffset`	`UTCOffset`	Offset from UTC

See page 9-19 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`pRLI` Indicates which catalog to consult to determine the UTC. Time servers within a catalog communicate among themselves to determine their UTC. Servers in a different catalog might have a different value of UTC. If you pass a valid record location information structure (RLI), you get that catalog's version of UTC. If you pass `nil` as the value of the `pRLI` field, the Authentication Manager calculates the values of the `theUTCTime` and `theUTCOffset` fields according to the clock in the user's Macintosh computer and the settings in the Map control panel. Packed record location information structures are described in the chapter "AOCE Utilities" in this book.

`theUTCTime` The function returns the current universal coordinated time (UTC) expressed as the number of seconds since 12:00 midnight, 1 January, 1904.

`theUTCOffset` The function returns the difference between the user's Macintosh computer's clock and UTC at Greenwich, England, expressed as the number of seconds. A negative number indicates that the user's computer is west of Greenwich according to the setting in the Map control panel.

DESCRIPTION

Call the `AuthGetUTCTime` function to obtain the current UTC. When you provide a valid RLI for a catalog, the function determines the UTC from the catalog server and local time from the settings in the Map control panel. The function returns the current UTC seconds since 1/1/1904 along with the offset from UTC in seconds of the local time, based on the distance of the local computer from Greenwich, England. Other Authentication Manager functions require input parameters based on UTC.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$021A

RESULT CODES

`noErr`	0	No error
`kOCEUnknownID`	–1567	Identity passed is not valid
`kOCENotLocal`	–1610	Internal AOCE error
`kOCETargetDirectoryInaccessible`	–1613	Catalog server not responding
`kOCENoSuchDNode`	–1615	The dNode was not found
`kOCEStreamCreationErr`	–1625	An error occurred in creating the stream

SEE ALSO

The `AuthGetUTCTime` function is used in an example in the section “Authentication Using ASDSP” on page 9-12.

Non-ASDSP Authentication Utilities

After obtaining credentials using the `AuthGetCredentials` function or the `AuthTradeProxyForCredentials` function, if you are not using the ASDSP transport mechanism, you can call functions to help you complete the challenge phase of authentication directly. This process for authenticating users is described in “Authentication for Non-ASDSP Users” beginning on page 9-13.

The Authentication Manager provides functions to

- make a challenge (`AuthMakeChallenge`)
- generate a reply to the challenge and a counterchallenge (`AuthMakeReply`)
- verify the reply and reply to the counterchallenge (`AuthVerifyReply`)
- extract information from the credentials (`AuthDecryptCredentials`)

AuthMakeChallenge

Call the AuthMakeChallenge function to generate a challenge, encrypted in the session key.

```
pascal OSErr AuthMakeChallenge (AuthParamBlockPtr paramBlock,
                                Boolean async);
```

paramBlock
A pointer to a parameter block.

async A value that specifies whether the function is to be executed asynchronously. Set this parameter to true if you want the function to be executed asynchronously.

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	key	AuthKeyPtr	Session key
↔	challenge	Ptr	Challenge buffer
→	challengeBufferLength	unsigned long	Challenge buffer size
←	challengeLength	unsigned long	Challenge length

See page 9-19 for descriptions of the ioCompletion and ioResult fields.

Field descriptions

key A pointer to the session key.

challenge A pointer to a buffer you provide in which to put the encrypted challenge.

challengeBufferLength
The size of the challenge buffer. The buffer must be at least 8 bytes in size.

challengeLength
The length of the encrypted challenge.

DESCRIPTION

An application that does not use ASDSP as the transport mechanism calls the AuthMakeChallenge function when it begins the process of setting up a new authenticated connection. Prior to calling this function, the application must obtain credentials from the authentication server using the AuthGetCredentials function or the AuthTradeProxyForCredentials function.

The AuthMakeChallenge function generates a token (a random number as described in the section “Steps in the Authentication Process” beginning on page 9-5), and encrypts it with the session key to create a challenge. You must then send the challenge to the recipient. Only initiators call this function.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$020F

RESULT CODES

noErr	0	No error
kOCELengthError	–1637	The supplied buffer was too small

SEE ALSO

The AuthMakeChallenge function is used in an example in the section "Authentication for Non-ASDSP Users" on page 9-13.

The AuthGetCredentials function is described on page 9-43.

The AuthTradeProxyForCredentials function is described on page 9-47.

The recipient uses the AuthMakeReply function, described next, to reply to the challenge.

AuthMakeReply

The AuthMakeReply function uses the token from an initial challenge to generate another token to be used as a challenge reply and also makes a counterchallenge.

```
pascal OSErr AuthMakeReply (AuthParamBlockPtr paramBlock,
                            Boolean async);
```

paramBlock
A pointer to a parameter block.

async
A value that specifies whether the function is to be executed asynchronously. Set this parameter to true if you want the function to be executed asynchronously.

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	key	AuthKeyPtr	Session key
→	challenge	Ptr	Challenge
↔	reply	Ptr	Reply buffer pointer
→	replyBufferLength	unsigned long	Reply buffer length
→	challengeLength	unsigned long	Challenge length
←	replyLength	unsigned long	Length of reply

See page 9-19 for descriptions of the ioCompletion and ioResult fields.

Field descriptions

`key`	The session key.
`challenge`	The challenge that was received from the initiator.
`reply`	A pointer to the buffer you supply into which the function puts the reply and the counterchallenge.
`replyBufferLength`	The length of the challenge reply buffer.
`challengeLength`	The length of the challenge.
`replyLength`	The length of the reply.

DESCRIPTION

The `AuthMakeReply` function decrypts a challenge created by the `AuthMakeChallenge` function, increments by 1 the number contained in the challenge, and then encrypts that new number in the session key. The result is the challenge reply. If you are a recipient, you call the `AuthMakeReply` function after you use the `AuthDecryptCredentials` function to decrypt the credentials—which are encrypted in your client key—to obtain the session key.

The `AuthMakeReply` function places in your buffer the reply to the challenge plus a counterchallenge. After you send the reply and counterchallenge to the initiator, the initiator calls the `AuthVerifyReply` function to verify the reply, thus continuing the challenge phase for authenticating a connection. The `AuthMakeReply` function is called only by recipients.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0210

RESULT CODES

`noErr`	0	No error
`kOCELengthError`	–1637	The supplied buffer was too small

SEE ALSO

The `AuthMakeReply` function is used in an example in the section "Authentication for Non-ASDSP Users" on page 9-13.

Use the `AuthDecryptCredentials` function (page 9-59) to extract the session key from the encrypted credentials.

The `AuthMakeChallenge` function is described on page 9-55. The `AuthVerifyReply` function is discussed next.

AuthVerifyReply

The `AuthVerifyReply` function verifies a challenge reply and makes a reply to the counterchallenge.

```
pascal OSErr AuthVerifyReply (AuthParamBlockPtr paramBlock,
                              Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`key`	`AuthKeyPtr`	Session key
→	`challenge`	`Ptr`	Challenge
↔	`reply`	`Ptr`	Reply buffer
→	`challengeLength`	`unsigned long`	Length of challenge
↔	`replyLength`	`unsigned long`	Length of reply

See page 9-19 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`key` A pointer to the session key.

`challenge` A pointer to the challenge you sent last.

`reply` A pointer to a buffer containing the reply returned by the other end of the connection.

`challengeLength`
The length of the challenge.

`replyLength` The length of the reply.

DESCRIPTION

Call the `AuthVerifyReply` function to verify a challenge reply and to make a reply to the counterchallenge during the challenge phase of setting up a secure connection. Both the initiator and the recipient call this function to verify the challenge replies they receive.

This function returns the result code `noErr` if the reply, after decryption, equals the challenge sent plus 1. A value of `kOCEAuthenticationTrouble` is returned by the `AuthVerifyReply` function if the reply cannot be verified. In that case, authentication has failed, and you should either terminate communication with the other party or continue communication with the understanding that the other party is not an authenticated entity.

After calling this function, the initiator should send the recipient the contents of the buffer pointed to by the `reply` field.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0211

RESULT CODES

`noErr`	0	No error
`kOCEAuthenticationTrouble`	–1571	Reply incorrect for the challenge sent

SEE ALSO

The `AuthVerifyReply` function is used in an example in the section "Authentication for Non-ASDSP Users" on page 9-13.

The `AuthMakeReply` function is described on page 9-56.

AuthDecryptCredentials

The `AuthDecryptCredentials` function decrypts credentials, extracting the session key, a pointer to the initiator's record ID, and the issue and expiration times for the credentials. Additionally, if an intermediary used a proxy to generate the credentials, the function returns a pointer to the record ID for the intermediary.

```
pascal OSErr AuthDecryptCredentials (AuthParamBlockPtr paramBlock,
                                     Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	userIdentity	AuthIdentity	Recipient's identity
↔	initiatorRecord	RecordIDPtr	Initiator's record ID
←	sessionKey	AuthKeyPtr	Session key
←	expiry	UTCTime	Credentials expiry time
←	issueTime	UTCTime	Credentials issue time
→	credentialsLength	unsigned long	Actual credentials size
→	credentials	Ptr	Credentials to be decrypted
←	hasIntermediary	Boolean	Intermediary found flag
↔	intermediary	RecordIDPtr	Intermediary who called

See page 9-19 for descriptions of the ioCompletion and ioResult fields.

Field descriptions

`userIdentity` The identity of the recipient wanting to decrypt credentials. The Authentication Manager gets your client key from your user record.

`initiatorRecord` The record ID of the entity initiating the challenge process. If you pass a local identity in the `userIdentity` field, you must pass in the `initiatorRecord` field a record ID containing a record location information structure (`RLI` struct) that specifies the catalog of the recipient. The function returns the record ID of the initiator in this field.

`sessionKey` The session key.

`expiry` The expiration time for the credentials.

`issueTime` The credentials issue time.

`credentialsLength` The size of the credentials.

`credentials` A pointer to the buffer holding the credentials to be decrypted.

`hasIntermediary` A Boolean value indicating whether the credentials were sent by an intermediary. If `true`, these credentials were obtained via a proxy by calling the `AuthTradeProxyForCredentials` function.

`intermediary` A pointer to the record ID of an intermediary, if any. You must allocate the record ID structure when you call the function. If you specify `nil` for this pointer, the function does not return the intermediary's record ID.

DESCRIPTION

When you are not using ASDSP as the transport mechanism, a recipient can use the `AuthDecryptCredentials` function to decrypt credentials received during a challenge. ASDSP decrypts credentials for its users, so you do not need to call the `AuthDecryptCredentials` function if you are using ASDSP.

Because the credentials are encrypted in the client key of the intended recipient, the function fails (with the result code `kOCEUnsupportedCredentialsVersion`) if you were not the intended recipient.

The `sessionKey` field is also given to the user or service requesting the decrypted credentials so that communicating users or services can share a key temporarily. You use this information to make encrypted challenge and challenge reply messages to complete the authentication process.

It is up to the user or service to refuse service if the credentials are premature or have expired.

If the function completes successfully, the `initiatorRecord`, `sessionKey`, `expiry`, `issueTime`, and `intermediary` fields contain plain text information extracted from the credentials.

SPECIAL CONSIDERATIONS

The recipient and initiator must be using the same PowerShare catalog for this function to succeed.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$020C

RESULT CODES

`noErr`	0	No error
`kOCEUnsupportedCredentialsVersion`	–1543	Problem reading the credentials

SEE ALSO

The `AuthDecryptCredentials` function is used in an example in the section "Authentication for Non-ASDSP Users" on page 9-13.

The `AuthGetCredentials` function is discussed on page 9-43.

The `AuthTradeProxyForCredentials` function is described on page 9-47.

PowerTalk Setup Catalog Management

The PowerTalk Setup catalog is a special personal catalog that contains information about the catalogs and electronic mail systems that are available to the principal user of the computer (see "The PowerTalk Setup Catalog" on page 9-9). Only CSAM and personal-MSAM template developers need to use the functions described in this section. If you are writing an application, you do not need to use these functions. See the chapter

"Service Access Module Setup" in *Inside Macintosh: Service Access Modules* for a complete description of setup templates and the PowerTalk Setup catalog.

The Authentication Manager provides functions associated with the PowerTalk Setup catalog to

- get the record ID and native name for a catalog in the PowerTalk Setup catalog (`OCESetupGetDirectoryInfo`)
- install catalogs and their passwords in the PowerTalk Setup catalog (`OCESetupAddDirectoryInfo`)
- change the password used to access a catalog in the PowerTalk Setup catalog (`OCESetupChangeDirectoryInfo`)
- remove a catalog from the PowerTalk Setup catalog (`OCESetupRemoveDirectoryInfo`)

OCESetupGetDirectoryInfo

Call the `OCESetupGetDirectoryInfo` function to get the record ID and native name of a specified catalog.

```
pascal OSErr OCESetupGetDirectoryInfo
                (AuthParamBlockPtr paramBlock, Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`directoryName`	`DirectoryNamePtr`	Catalog name
→	`discriminator`	`DirDiscriminator`	Catalog discriminator
↔	`recordID`	`RecordIDPtr`	Catalog record ID
↔	`nativeName`	`RStringPtr`	User's name
↔	`password`	`RStringPtr`	Password

See page 9-19 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`directoryName` A pointer to the catalog name.

`discriminator` A value that differentiates two catalogs with the same name. It is part of the `RLI` structure.

`recordID` A pointer to a record ID structure into which the function places the record ID of the catalog.

`nativeName`	A pointer to an `RString` structure into which the function places the native name. Allocate a buffer large enough to hold an `RString64` structure to hold this name.
`password`	For non-PowerShare catalogs, a pointer to an `RString` structure into which the function places the user or service password. Allocate a buffer large enough to hold an `RString64` structure to hold this password. This field is undefined for PowerShare catalogs.

DESCRIPTION

Call the `OCESetupGetDirectoryInfo` function to obtain the native name and record ID for a particular catalog installed in the PowerTalk Setup catalog. You specify the catalog name and discriminator. The *native name* is generally the user's name or account name in the external catalog, if it is different from the name of the user's User record. The CSAM or MSAM developer specifies this native name when installing the SAM in the Setup catalog.

The Collaboration toolbox returns the password only for non-PowerShare catalogs. An MSAM or CSAM can use this function to obtain from the Setup catalog the password required by the external system the SAM supports.

You must provide the buffers for the record ID, native name, and password that are returned.

SPECIAL CONSIDERATIONS

The local ID must be unlocked before you call this function.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$020E

RESULT CODES

`noErr`	0	No error
`kOCELocalAuthenticationFail`	–1561	Local identity locked
`kOCEDirectoryIdentitySetupDoesNotExist`	–1564	Specific catalog has not been set up

SEE ALSO

See "The PowerTalk Setup Catalog" on page 9-9 for a description of the PowerTalk Setup catalog. See the chapter "Service Access Module Setup" in *Inside Macintosh: Service Access Modules* for a complete description of setup templates.

Record IDs and `RLI` structures are described in the chapter "AOCE Utilities" in this book.

The chapter "AOCE Utilities" in this book shows sample code that allocates space for a record ID.

OCESetupAddDirectoryInfo

Call the `OCESetupAddDirectoryInfo` function to add a catalog and its associated password to the PowerTalk Setup catalog.

```
pascal OSErr OCESetupAddDirectoryInfo
              (AuthParamBlockPtr paramBlock, Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`directoryRecordCID`	`CreationID`	Creation ID of catalog record
→	`recordID`	`RecordIDPtr`	Record ID for catalog
→	`password`	`RStringPtr`	Password

See page 9-19 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`directoryRecordCID`
The creation ID of the Combined record or Catalog record in the Setup catalog. You can use the `kDETcmdGetDSSpec` template callback function to determine the creation ID of the Combined record or Catalog record.

`recordID` A pointer to the record ID specifying the user for the catalog.

`password` A pointer to the password associated with the record ID in the catalog.

DESCRIPTION

Only a setup template for a service access module (SAM) calls the `OCESetupAddDirectoryInfo` function. Before calling the `OCESetupAddDirectoryInfo` function, be sure the local identity is unlocked.

The `RLI` data structure within the user's record ID must contain the catalog name to be added to the Combined record or Catalog record.

The AOCE software encrypts the password before putting it in the PowerTalk Setup catalog.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$0219

RESULT CODES

noErr	0	No error
kOCELocalAuthenticationFail	–1561	Local identity locked
kOCEDirectoryIdentitySetupExists	–1563	Identity has already been set up
kOCEDirectoryNotFoundErr	–1630	Catalog was not found in the list

SEE ALSO

Creation IDs and the RLI structure are discussed in the chapter "AOCE Utilities" in this book.

The kDETcmdGetDSSpec template callback function is described in the chapter "AOCE Templates" in this book.

Setup templates and the procedure for adding a SAM to the Setup catalog are described in the chapter "Service Access Module Setup" in *Inside Macintosh: Service Access Modules.*

OCESetupChangeDirectoryInfo

Call the OCESetupChangeDirectoryInfo function to change the record ID and password for an existing catalog in the PowerTalk Setup catalog. The Authentication Manager verifies the current catalog password before changing it to the new password.

```
pascal OSErr OCESetupChangeDirectoryInfo
                (AuthParamBlockPtr paramBlock, Boolean async);
```

paramBlock
A pointer to a parameter block.

async
A value that specifies whether the function is to be executed asynchronously. Set this parameter to true if you want the function to be executed asynchronously.

Parameter block

→	ioCompletion	ProcPtr	Your completion routine
←	ioResult	OSErr	Result code
→	directoryRecordCID	CreationID	Catalog creation ID
→	recordID	RecordIDPtr	User's record ID
→	password	RStringPtr	Password
→	newPassword	RStringPtr	New password

See page 9-19 for descriptions of the ioCompletion and ioResult fields.

Field descriptions

directoryRecordCID
: The creation ID of the Combined record or Catalog record in the Setup catalog. You can use the kDETcmdGetDSSpec template callback function to determine the creation ID of the Combined record or Catalog record.

recordID
: A pointer to the new record ID for the user. If you don't want to change the record ID, specify the old record ID.

password
: A pointer to the current password associated with the record ID.

newPassword
: A pointer to the new password to be associated with the record ID. If you don't want to change the password, repeat the old password in this field.

DESCRIPTION

Only a setup template for a SAM calls this function. Before calling the OCESetupChangeDirectoryInfo function, be sure the local identity is unlocked.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
_oceTBDispatch	$021B

RESULT CODES

noErr	0	No error
kOCELocalAuthenticationFail	–1561	Local identity locked
kOCEDirectoryNotFoundErr	–1630	Catalog was not found in the list

SEE ALSO

Creation IDs and record IDs are discussed in the chapter "AOCE Utilities" in this book.

The kDETcmdGetDSSpec template callback function is described in the chapter "AOCE Templates" in this book.

OCESetupRemoveDirectoryInfo

Call the OCESetupRemoveDirectoryInfo function to remove a catalog from the PowerTalk Setup catalog.

```
pascal OSErr OCESetupRemoveDirectoryInfo
                  (AuthParamBlockPtr paramBlock, Boolean async);
```

`paramBlock`
A pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`directoryRecordCID`	`CreationID`	Catalog creation ID

See page 9-19 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`directoryRecordCID`
The creation ID for the Catalog record or Combined record associated with the catalog to be removed from the PowerTalk Setup catalog. You can use the `kDETcmdGetDSSpec` template callback function to determine the creation ID of the Combined record or Catalog record.

DESCRIPTION

Only a setup template for a SAM can call the `OCESetupRemoveDirectoryInfo` function. This function removes from the Catalog or Combined record in the PowerTalk Setup catalog the attributes that were added by the `OCESetupAddDirectoryInfo` function.

Before calling the `OCESetupRemoveDirectoryInfo` function, be sure the local identity is unlocked.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$020D

RESULT CODES

`noErr`	0	No error
`kOCENoSuchRecord`	–1618	No such record

SEE ALSO

The `kDETcmdGetDSSpec` template callback function is described in the chapter “AOCE Templates” in this book.

Use the `OCESetupAddDirectoryInfo` function (page 9-64) to add a catalog and its associated password to the PowerTalk Setup catalog.

Use the `OCESetupChangeDirectoryInfo` function (page 9-65) to change the record ID and password for an existing catalog in the PowerTalk Setup catalog.

Application-Defined Functions

This section describes the completion routine required for asynchronous use of authentication functions and the notification routine that you provide to Authentication Manager functions that use a notification queue.

MyCompletion

An Authentication Manager completion routine has the following syntax:

```
void MyCompletion (Ptr paramBlk);
```

`paramBlk` A pointer to the parameter block that you provided when you called the Authentication Manager function.

DESCRIPTION

When you execute an Authentication Manager function asynchronously (by setting its `async` parameter to `true`) you can specify a completion routine by passing the routine's address in the `ioCompletion` field of the parameter block. A function called asynchronously returns control to your application with the result code `noErr` as soon as the function is placed in the execution queue. This result code does not indicate that the function has successfully completed but indicates only that the function was successfully placed in the queue. To determine when the function is actually completed, you can inspect the `ioResult` field of the parameter block. This field is set to 1 when the function is called and set to the actual result code when the function is completed. If you specify a completion routine, it is executed after the result code is placed in the `ioResult` field.

SPECIAL CONSIDERATIONS

Because a completion routine may be executed at interrupt time, it should not allocate, move, or purge memory (either directly or indirectly) and should not depend on the validity of handles to unlocked blocks.

When the Authentication Manager calls your completion routine, it sets the A5 register to the value it contained when you called the function that set up the completion routine.

ASSEMBLY-LANGUAGE INFORMATION

When your completion routine is called, register A0 contains a pointer to the parameter block of the function called, and register D0 contains the result code. The value in register D0 is always identical to the value in the `ioResult` field of the parameter block.

A completion routine must preserve all registers other than A0, A1, and D0–D2.

MyNotificationProc

The `MyNotificationProc` function is a notification routine you must provide when you use the notification queue.

```
pascal Boolean MyNotificationProc (long clientData,
                        AuthLocalIdentityOp callValue,
                        AuthLocalIdentityLockAction actionValue,
                        LocalIdentity identity);
```

`clientData`
The value that you provided in the `clientData` field of the parameter block that you passed to the `AuthAddToLocalIdentityQueue` function. This field provides a way for you to pass a parameter to your notification routine.

`callValue` When the Authentication Manager calls your notification routine, it sets this parameter to `kAuthLockLocalIdentityOp` to indicate a lock operation, `kAuthUnlockLocalIdentityOp` to indicate an unlock operation, or to `kAuthLocalIdentityNameChangeOp` to indicate a name change. In the case of a lock operation, you must also check the value of the `actionValue` parameter.

`actionValue`
When the Authentication Manager calls your notification routine with the `kAuthLockLocalIdentityOp` value in the `callValue` parameter, it sets the `actionValue` parameter to either `kAuthLockPending`, indicating a lock is pending, or to `kAuthLockWillBeDone` when a lock is about to be done.

`identity` The local identity.

DESCRIPTION

The AOCE toolbox calls the notification procedure you provide each time the local identity access to a user's computer is locked or unlocked, or when the user changes in the name in the Key Chain, so that the applications in the notification queue can be informed of changes in the access to catalogs listed in the PowerTalk Setup catalog.

Applications registered in the notification queue are notified when a user locks his or her local identity because he or she is leaving a computer unattended, and again when the user returns and provides his or her password to the system.

When it plans to lock local identity access, the Authentication Manager notifies all applications installed in the notification queue. To do so, the Authentication Manager passes the value `kAuthLockPending` in the `actionValue` parameter. Your notification procedure can return `true` to deny permission to lock the local identity. If none of the applications in the queue refuse the lock operation, the Collaboration toolbox passes the value `kAuthLockWillBeDone` to notify the applications that the lock is imminent.

You should deny locking only if you are performing some operation that would be seriously disrupted if the lock function succeeded.

The Authentication Manager handles the buffers associated with pointers that it passes to a notification procedure. You must copy the data in these buffers if you want to refer to it after your notification procedure completes execution.

SPECIAL CONSIDERATIONS

This routine should not allocate, move, or purge memory (either directly or indirectly). Like completion routines, your notification procedure should not call the `WaitNextEvent`, `EventAvail`, `OSEventAvail`, or `SystemTask` routines or any routine that might call those functions.

SEE ALSO

For an example of the use of the `MyNotificationProc` function, see Listing 9-1 on page 9-15.

See “Locking and Unlocking Local Identities” on page 9-8 for more information about locking and unlocking users’ computers.

See “The PowerTalk Setup Catalog” on page 9-9 for more information about the PowerTalk Setup catalog.

The `AuthAddToLocalIdentityQueue` function is discussed on page 9-30.

The `AuthRemoveFromLocalIdentityQueue` function is discussed on page 9-31.

Summary of the Authentication Manager

C Summary

Constants and Data Types

```
enum {
      /* values for key sizes */
      kRC4KeySizeInBytes               = 8,     /* size of an RC4 key */
      kRefNumUnknown                   = 0      /* dsRefNum specifier */
};

enum {
      /* values of AuthLocalIdentityOp for notification routine */
      kAuthLockLocalIdentityOp         = 1,
      kAuthUnlockLocalIdentityOp       = 2,
      kAuthLocalIdentityNameChangeOp   = 3
};

enum {
      /* values of AuthLocalIdentityLockAction for notification routine */
      kAuthLockPending                 = 1,
      kAuthLockWillBeDone              = 2
};

/* values of notifyFlags field of AuthAddToLocalIdentityQueue function*/
enum {kNotifyLockBit, kNotifyUnlockBit, kNotifyNameChangeBit};
enum {
      kNotifyLockMask                  = 1L << kNotifyLockBit,
      kNotifyUnlockMask                = 1L << kNotifyUnlockBit
      kNotifyNameChangeMask            = 1L << kNotifyNameChangeBit
};
```

Identity Declarations

```
typedef unsigned long AuthIdentity;                /* identity */
typedef AuthIdentity LocalIdentity;                /* local identity */
typedef unsigned long AuthLocalIdentityOp;
typedef unsigned long AuthLocalIdentityLockAction;
typedef unsigned long AuthNotifications;
```

Key Structures

```
struct DESKey { /* A DES key is 8 bytes of data */
   unsigned long a;
   unsigned long b;
};
typedef struct DESKey DESKey;
typedef Byte RC4Key[kRC4KeySizeInBytes];
typedef unsigned long AuthKeyType;
struct AuthKey {                           /* key type followed by its data */
   AuthKeyType keyType;
   union {
            DESKey des;
            RC4Key rc4;
         }u;
};
typedef struct Authkey AuthKey;
typedef AuthKey *AuthKeyPtr;
```

Parameter Block Header

```
#define AuthDirParamHeader
   Ptr            qLink;         /* reserved */
   long           reserved1;     /* reserved */
   long           reserved2;     /* reserved */
   ProcPtr        ioCompletion;  /* your completion function */
   OSErr          ioResult;      /* result code */
   unsigned long  saveA5;        /* reserved */
   short          reqCode;       /* reserved */
   long           reserved[2];   /* reserved */
   AddrBlock      serverHint;    /* PowerShare server's AppleTalk address */
   short          dsRefNum;      /* reserved */
   unsigned long  callID;        /* reserved */
   AuthIdentity   identity;      /* initiator's authentication identity */
   long           gReserved1;    /* reserved */
```

```
   long           gReserved2;    /* reserved */
   long           gReserved3;    /* reserved */
   long           clientData;    /* you define this field */
```

Parameter Blocks

```
struct AuthPasswordToKeyPB {
   AuthDirParamHeader
   RecordIDPtr          userRecord; /* User record */
   AuthKeyPtr           key;
   RStringPtr           password;   /* pointer to the new password string */
};
typedef struct AuthPasswordToKeyPB AuthPasswordToKeyPB;

struct AuthAddKeyPB {
   AuthDirParamHeader
   RecordIDPtr          userRecord; /* User record */
   AuthKeyPtr           userKey;    /* AOCE key for the user */
   RStringPtr           password;   /* pointer to password string */
};
typedef struct AuthAddKeyPB AuthAddKeyPB;

struct AuthChangeKeyPB {
   AuthDirParamHeader
   RecordIDPtr          userRecord; /* User record */
   AuthKeyPtr           userKey;    /* new AOCE key for the user */
   RStringPtr           password;   /* pointer to the new password string */
};
typedef struct AuthChangeKeyPB AuthChangeKeyPB;

struct AuthDeleteKeyPB {
   AuthDirParamHeader
   RecordIDPtr          userRecord; /* User record */
};
typedef struct AuthDeleteKeyPB AuthDeleteKeyPB;

struct AuthGetLocalIdentityPB {
   AuthDirParamHeader
   LocalIdentity        theLocalIdentity; /* local identity */
};
typedef struct AuthGetLocalIdentityPB AuthGetLocalIdentityPB;

struct AuthAddToLocalIdentityQueuePB {
   AuthDirParamHeader
   NotificationProc     notifyProc;    /* notification procedure */
```

```
   AuthNotifications    notifyFlags;   /* notifyFlags */
   StringPtr            appName;       /* name of application to be
                                           returned in Delete/Stop */
};
typedef struct AuthAddToLocalIdentityQueuePB AuthAddToLocalIdentityQueuePB;

struct AuthRemoveFromLocalIdentityQueuePB {
   AuthDirParamHeader
   NotificationProc     notifyProc;    /* notification procedure */
};
typedef struct AuthRemoveFromLocalIdentityQueuePB
AuthRemoveFromLocalIdentityQueuePB;

struct AuthSetupLocalIdentityPB {
   AuthDirParamHeader
   long                 aReserved;
   RStringPtr           userName;      /* user name */
   RStringPtr           password;      /* user password */
};
typedef struct AuthSetupLocalIdentityPB AuthSetupLocalIdentityPB;

struct AuthChangeLocalIdentityPB {
   AuthDirParamHeader
   long                 aReserved;
   RStringPtr           userName;      /* user name */
   RStringPtr           password;      /* current password */
   RStringPtr           newPassword;   /* new password */
};
typedef struct AuthChangeLocalIdentityPB AuthChangeLocalIdentityPB;

struct AuthLockLocalIdentityPB {
   AuthDirParamHeader
   LocalIdentity        theLocalIdentity; /* local identity */
   StringPtr            name;             /* name of the app that
                                             denied delete */
};
typedef struct AuthLockLocalIdentityPB AuthLockLocalIdentityPB;

struct AuthUnlockLocalIdentityPB {
   AuthDirParamHeader
   LocalIdentity        theLocalIdentity; /* local identity */
   RStringPtr           userName;         /* user name */
```

```
   RStringPtr           password;         /* user password */
};
typedef struct AuthUnlockLocalIdentityPB AuthUnlockLocalIdentityPB;

struct AuthRemoveLocalIdentityPB {
   AuthDirParamHeader
   long                 aReserved;
   RStringPtr           userName;         /* user name */
   RStringPtr           password;         /* current password */
};
typedef struct AuthRemoveLocalIdentityPB AuthRemoveLocalIdentityPB;

struct AuthBindSpecificIdentityPB {
   AuthDirParamHeader
   AuthIdentity         userIdentity;     /* binding identity */
   RecordIDPtr          userRecord;       /* User record */
   AuthKeyPtr           userKey;          /* AOCE key for the user */
};
typedef struct AuthBindSpecificIdentityPB AuthBindSpecificIdentityPB;

struct AuthUnbindSpecificIdentityPB {
   AuthDirParamHeader
   AuthIdentity         userIdentity;     /* identity to be deleted */
};
typedef struct AuthUnbindSpecificIdentityPB AuthUnbindSpecificIdentityPB;

struct AuthGetSpecificIdentityInfoPB {
   AuthDirParamHeader
   AuthIdentity         userIdentity;     /* identity of initiator */
   RecordIDPtr          userRecord;       /* User record */
};
typedef struct AuthGetSpecificIdentityInfoPB AuthGetSpecificIdentityInfoPB;

struct AuthGetCredentialsPB {
   AuthDirParamHeader
   AuthIdentity         userIdentity;     /* identity of initiator */
   RecordIDPtr          recipient;        /* AOCE name of recipient */
   AuthKeyPtr           sessionKey;       /* session key */
   UTCTime              expiry;           /* desired/actual expiration */
   unsigned long        credentialsLength;/* max/actual credentials size */
   Ptr                  credentials;      /* buffer where credentials
                                             are returned */
};
typedef struct AuthGetCredentialsPB AuthGetCredentialsPB;
```

```
struct AuthMakeProxyPB {
   AuthDirParamHeader
   AuthIdentity         userIdentity;  /* identity of principal */
   RecordIDPtr          recipient;     /* AOCE name of recipient */
   UTCTime              firstValid;    /* time at which proxy
                                          becomes valid */
   UTCTime              expiry;        /* time at which proxy expires */
   unsigned long        authDataLength;/* size of authorization data */
   Ptr                  authData;      /* pointer to authorization data */
   unsigned long        proxyLength;   /* max/actual proxy size */
   Ptr                  proxy;         /* buffer where proxy is returned */
   RecordIDPtr          intermediary;  /* record ID of intermediary */
};
typedef struct AuthMakeProxyPB AuthMakeProxyPB;

struct AuthTradeProxyForCredentialsPB {
   AuthDirParamHeader
   AuthIdentity         userIdentity;     /* identity of intermediary */
   RecordIDPtr          recipient;        /* AOCE name of recipient */
   AuthKeyPtr           sessionKey;       /* session key */
   UTCTime              expiry;           /* desired/actual expiration */
   unsigned long        credentialsLength;/* max/actual credentials size */
   Ptr                  credentials;      /* buffer where credentials
                                             are returned */
   unsigned long        proxyLength;      /* actual proxy size */
   Ptr                  proxy;            /* buffer containing proxy */
   RecordIDPtr          principal;        /* record ID of principal */
};
typedef struct AuthTradeProxyForCredentialsPB AuthTradeProxyForCredentialsPB;

struct AuthResolveCreationIDPB {
   AuthDirParamHeader
   RecordIDPtr          userRecord;       /* User record */
   unsigned long        bufferLength;     /* buffer Size */
   Ptr                  buffer;           /* buffer to hold creation IDs */
   unsigned long        totalMatches;     /* total number of matching
                                             names found */
   unsigned long        actualMatches;    /* number of matches returned in
                                             the buffer */
};
typedef struct AuthResolveCreationIDPB AuthResolveCreationIDPB;
```

```
struct AuthGetUTCTimePB {
   AuthDirParamHeader
   PackedRLIPtr      pRLI;          /* packed RLI of the dNode */
   UTCTime           theUTCTime;    /* current UTC(GMT) time in seconds
                                       since 1/1/1904 */
   UTCOffset         theUTCOffset;  /* offset from UTC(GMT) seconds
                                       east of Greenwich */
};
typedef struct AuthGetUTCTimePB AuthGetUTCTimePB;

struct AuthMakeChallengePB {
   AuthDirParamHeader
   AuthKeyPtr        key;                    /* unencrypted session key */
   Ptr               challenge;              /* encrypted challenge */
   unsigned long     challengeBufferLength;  /* length of challenge buffer */
   unsigned long     challengeLength;        /* length of encrypted
                                                challenge */
};
typedef struct AuthMakeChallengePB AuthMakeChallengePB;

struct AuthMakeReplyPB {
   AuthDirParamHeader
   AuthKeyPtr        key;                    /* unencrypted session key */
   Ptr               challenge;              /* encrypted challenge */
   Ptr               reply;                  /* encrypted reply */
   unsigned long     replyBufferLength;      /* length of challenge buffer */
   unsigned long     challengeLength;        /* length of encrypted
                                                challenge */
   unsigned long     replyLength;            /* length of encrypted reply */
};
typedef struct AuthMakeReplyPB AuthMakeReplyPB;

struct AuthVerifyReplyPB {
   AuthDirParamHeader
   AuthKeyPtr        key;                    /* unencrypted session key */
   Ptr               challenge;              /* encrypted challenge */
   Ptr               reply;                  /* encrypted reply */
   unsigned long     challengeLength;        /* length of encrypted
                                                challenge */
   unsigned long     replyLength;            /* length of encrypted reply */
};
typedef struct AuthVerifyReplyPB AuthVerifyReplyPB;
```

```
struct AuthDecryptCredentialsPB {
   AuthDirParamHeader
   AuthIdentity      userIdentity;     /* user's identity */
   RecordIDPtr       initiatorRecord;  /* AOCE name of the initiator */
   AuthKeyPtr        sessionKey;       /* session key */
   UTCTime           expiry;           /* credentials expiration time */
   unsigned long     credentialsLength;/* actual credentials size */
   Ptr               credentials;      /* credentials to be decrypted */
   UTCTime           issueTime;        /* credentials expiration time */
   Boolean           hasIntermediary;  /* if true, an intermediary record
                                          was found in credentials */
   RecordIDPtr       intermediary;     /* record ID of the intermediary */
};
typedef struct AuthDecryptCredentialsPB AuthDecryptCredentialsPB;

struct OCESetupGetDirectoryInfoPB {
   AuthDirParamHeader
   DirectoryNamePtr  directoryName;    /* catalog name */
   DirDiscriminator  discriminator;    /* discriminator for the catalog */
   RecordIDPtr       recordID;         /* record ID for the catalog */
   RStringPtr        nativeName;       /* user name in the catalog world */
   RStringPtr        password;         /* password in the catalog world */
};
typedef struct OCESetupGetDirectoryInfoPB OCESetupGetDirectoryInfoPB;

struct OCESetupAddDirectoryInfoPB {
   AuthDirParamHeader
   CreationID        directoryRecordCID;  /* creation ID for the catalog */
   RecordIDPtr       recordID;            /* record ID for the identity */
   RStringPtr        password;            /* password in the catalog world */
};
typedef struct OCESetupAddDirectoryInfoPB OCESetupAddDirectoryInfoPB;

struct OCESetupChangeDirectoryInfoPB {
   AuthDirParamHeader
   CreationID     directoryRecordCID;     /* creation ID for the catalog */
   RecordIDPtr    recordID;               /* record ID for the identity */
   RStringPtr     password;               /* password in the catalog world */
   RStringPtr     newPassword;            /* new password in the catalog */
};
typedef struct OCESetupChangeDirectoryInfoPB OCESetupChangeDirectoryInfoPB;
```

```
struct OCESetupRemoveDirectoryInfoPB {
   AuthDirParamHeader
   CreationID     directoryRecordCID;     /* creation ID for the catalog */
};
typedef struct OCESetupRemoveDirectoryInfoPB OCESetupRemoveDirectoryInfoPB;
```

Parameter Block Union Structure

```
union AuthParamBlock {
   struct {AuthDirParamHeader}header;
   AuthBindSpecificIdentityPB          bindIdentityPB;
   AuthUnbindSpecificIdentityPB        unbindIdentityPB;
   AuthResolveCreationIDPB             resolveCreationIDPB;
   AuthGetSpecificIdentityInfoPB       getIdentityInfoPB;
   AuthAddKeyPB                        addKeyPB;
   AuthChangeKeyPB                     changeKeyPB;
   AuthDeleteKeyPB                     deleteKeyPB;
   AuthPasswordToKeyPB                 passwordToKeyPB;
   AuthGetCredentialsPB                getCredentialsPB;
   AuthDecryptCredentialsPB            decryptCredentialsPB;
   AuthMakeChallengePB                 makeChallengePB;
   AuthMakeReplyPB                     makeReplyPB;
   AuthVerifyReplyPB                   verifyReplyPB;
   AuthGetUTCTimePB                    getUTCTimePB;
   AuthMakeProxyPB                     makeProxyPB;
   AuthTradeProxyForCredentialsPB      tradeProxyForCredentialsPB;
   AuthGetLocalIdentityPB              getLocalIdentityPB;
   AuthUnlockLocalIdentityPB           unLockLocalIdentityPB;
   AuthLockLocalIdentityPB             lockLocalIdentityPB;
   AuthAddToLocalIdentityQueuePB       localIdentityQInstallPB;
   AuthRemoveFromLocalIdentityQueuePB  localIdentityQRemovePB;
   AuthSetupLocalIdentityPB            setupLocalIdentityPB;
   AuthChangeLocalIdentityPB           changeLocalIdentityPB;
   AuthRemoveLocalIdentityPB           removeLocalIdentityPB;
   OCESetupAddDirectoryInfoPB          setupDirectoryIdentityPB;
   OCESetupChangeDirectoryInfoPB       changeDirectoryIdentityPB;
   OCESetupRemoveDirectoryInfoPB       removeDirectoryIdentityPB;
   OCESetupGetDirectoryInfoPB          getDirectoryIdentityInfoPB;
};

typedef union AuthParamBlock AuthParamBlock;
typedef AuthParamBlock *AuthParamBlockPtr;
```

Authentication Manager Functions

Key Management

```
pascal OSErr AuthPasswordToKey
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthAddKey     (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthChangeKey  (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthDeleteKey  (AuthParamBlockPtr paramBlock,
                             Boolean async);
```

Local Identity Management

```
pascal OSErr AuthGetLocalIdentity
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthAddToLocalIdentityQueue
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthRemoveFromLocalIdentityQueue
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthSetupLocalIdentity
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthChangeLocalIdentity
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthLockLocalIdentity
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthUnlockLocalIdentity
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthRemoveLocalIdentity
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
```

Specific Identity Management

```
pascal OSErr AuthBindSpecificIdentity
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthUnbindSpecificIdentity
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthGetSpecificIdentityInfo
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
```

Credentials Management

```
pascal OSErr AuthGetCredentials
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthMakeProxy  (AuthParamBlockPtr paramBlock, Boolean async);
pascal OSErr AuthTradeProxyForCredentials
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
```

Creation ID Resolution Management

```
pascal OSErr AuthResolveCreationID
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
```

Time Service

```
pascal OSErr AuthGetUTCTime (AuthParamBlockPtr paramBlock, Boolean async);
```

Non-ASDSP Authentication Utilities

```
pascal OSErr AuthMakeChallenge
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthMakeReply  (AuthParamBlockPtr paramBlock, Boolean async);
pascal OSErr AuthVerifyReply
                            (AuthParamBlockPtr paramBlock, Boolean async);
pascal OSErr AuthDecryptCredentials
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
```

AOCE Setup Catalog Management

```
pascal OSErr OCESetupGetDirectoryInfo
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr OCESetupAddDirectoryInfo
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr OCESetupChangeDirectoryInfo
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr OCESetupRemoveDirectoryInfo
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
```

Application-Defined Functions

```
void MyCompletion           (Ptr paramBlk);
pascal Boolean MyNotificationProc
                            (long clientData,
                             AuthLocalIdentityOp callValue,
                             AuthLocalIdentityLockAction actionValue,
                             LocalIdentity identity);
```

Pascal Summary

Constants

```
CONST {values for key sizes}
      kRC4KeySizeInBytes               = 8;        {size of an RC4 key}
      kRefNumUnknown                   = 0;        {dsRefNum specifier}

      {values of AuthLocalIdentityOp}
      kAuthLockLocalIdentityOp         = 1;
      kAuthUnlockLocalIdentityOp       = 2;
      kAuthLocalIdentityNameChangeOp   = 3;

      {values of AuthLocalIdentityLockAction}
      kAuthLockPending                 = 1;
      kAuthLockWillBeDone              = 2;
```

```
{values of AuthNotifications}
kNotifyLockBit                = 0;
kNotifyUnlockBit              = 1;
kNotifyNameChangeBit          = 2;
kNotifyLockMask               = $00000001;  {1<<kNotifyLockBit}
kNotifyUnlockMask             = $00000002;  {1<<kNotifyUnlockBit}
kNotifyNameChangeMask         = $00000004;  {1<<kNotifyNameChangeBit}
```

Data Types

```
AuthIdentity   = LongInt;              {unique identifier for an identity}
LocalIdentity  = AuthIdentity;         {umbrella local identity}

AuthLocalIdentityOp           = LongInt;
AuthLocalIdentityLockAction   = LongInt;
AuthNotifications             = LongInt;
```

Key Structures

```
TYPE
   DESKey =
   RECORD               { a DES key is 8 bytes of data }
      a: LongInt;
      b: LongInt;
   END;

   RC4Key      =  PACKED ARRAY[1..kRC4KeySizeInBytes] OF Byte;
   AuthKeyType =  LongInt;

   AuthKey =
   RECORD               { key type followed by its data }
      keyType: AuthKeyType;
      CASE INTEGER OF
         1: (des: DESKey);
         2: (rc4: RC4Key);
   END;

   AuthKeyPtr     = ^AuthKey;
```

Parameter Block Header

```
   AuthDirParamHeader = RECORD
      qLink:         Ptr;
      reserved1:     LongInt;
```

```
   reserved2:     LongInt;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LongInt;
   reqCode:       Integer;
   reserved:      ARRAY[1..2] OF LongInt;
   serverHint:    AddrBlock;
   dsRefNum:      Integer;
   callID:        LongInt;
   identity:      AuthIdentity;
   gReserved1:    LongInt;
   gReserved2:    LongInt;
   gReserved3:    LongInt;
   clientData:    LongInt;
END;
```

Parameter Blocks

```
AuthPasswordToKeyPB = RECORD
   qLink:         Ptr;
   reserved1:     LongInt;
   reserved2:     LongInt;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LongInt;
   reqCode:       Integer;
   reserved:      ARRAY[1..2] OF LongInt;
   serverHint:    AddrBlock;
   dsRefNum:      Integer;
   callID:        LongInt;
   identity:      AuthIdentity;
   gReserved1:    LongInt;
   gReserved2:    LongInt;
   gReserved3:    LongInt;
   clientData:    LongInt;
   userRecord:    RecordIDPtr;      {User record}
   key:           AuthKeyPtr;
   password:      RStringPtr;       {pointer to the new password string}
END;

AuthAddKeyPB = RECORD
   qLink:         Ptr;
   reserved1:     LongInt;
   reserved2:     LongInt;
```

```
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LongInt;
   reqCode:       Integer;
   reserved:      ARRAY[1..2] OF LongInt;
   serverHint:    AddrBlock;
   dsRefNum:      Integer;
   callID:        LongInt;
   identity:      AuthIdentity;
   gReserved1:    LongInt;
   gReserved2:    LongInt;
   gReserved3:    LongInt;
   clientData:    LongInt;
   userRecord:    RecordIDPtr;      {User record}
   userKey:       AuthKeyPtr;       {AOCE key for the user}
   password:      RStringPtr;       {pointer to password string}
END;

AuthChangeKeyPB = RECORD
   qLink:         Ptr;
   reserved1:     LongInt;
   reserved2:     LongInt;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LongInt;
   reqCode:       Integer;
   reserved:      ARRAY[1..2] OF LongInt;
   serverHint:    AddrBlock;
   dsRefNum:      Integer;
   callID:        LongInt;
   identity:      AuthIdentity;
   gReserved1:    LongInt;
   gReserved2:    LongInt;
   gReserved3:    LongInt;
   clientData:    LongInt;
   userRecord:    RecordIDPtr;      {User record}
   userKey:       AuthKeyPtr;       {new AOCE key for the user}
   password:      RStringPtr;       {pointer to the new password string}
END;

AuthDeleteKeyPB = RECORD
   qLink:         Ptr;
   reserved1:     LongInt;
   reserved2:     LongInt;
```

```
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LongInt;
   reqCode:       Integer;
   reserved:      ARRAY[1..2] OF LongInt;
   serverHint:    AddrBlock;
   dsRefNum:      Integer;
   callID:        LongInt;
   identity:      AuthIdentity;
   gReserved1:    LongInt;
   gReserved2:    LongInt;
   gReserved3:    LongInt;
   clientData:    LongInt;
   userRecord:    RecordIDPtr;            {User record}
END;

AuthGetLocalIdentityPB = RECORD
   qLink:            Ptr;
   reserved1:        LongInt;
   reserved2:        LongInt;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LongInt;
   reqCode:          Integer;
   reserved:         ARRAY[1..2] OF LongInt;
   serverHint:       AddrBlock;
   dsRefNum:         Integer;
   callID:           LongInt;
   identity:         AuthIdentity;
   gReserved1:       LongInt;
   gReserved2:       LongInt;
   gReserved3:       LongInt;
   clientData:       LongInt;
   theLocalIdentity: LocalIdentity;       {local identity}
END;

AuthAddToLocalIdentityQueuePB = RECORD
   qLink:         Ptr;
   reserved1:     LongInt;
   reserved2:     LongInt;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LongInt;
   reqCode:       Integer;
```

```
   reserved:      ARRAY[1..2] OF LongInt;
   serverHint:    AddrBlock;
   dsRefNum:      Integer;
   callID:        LongInt;
   identity:      AuthIdentity;
   gReserved1:    LongInt;
   gReserved2:    LongInt;
   gReserved3:    LongInt;
   clientData:    LongInt;
   notifyProc:    NotificationProc;    {notification procedure}
   notifyFlags:   AuthNotifications;   {notification flags}
   appName:       StringPtr;           {name of application to be
                                        returned in Delete/Stop}
END;

AuthRemoveFromLocalIdentityQueuePB = RECORD
   qLink:         Ptr;
   reserved1:     LongInt;
   reserved2:     LongInt;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LongInt;
   reqCode:       Integer;
   reserved:      ARRAY[1..2] OF LongInt;
   serverHint:    AddrBlock;
   dsRefNum:      Integer;
   callID:        LongInt;
   identity:      AuthIdentity;
   gReserved1:    LongInt;
   gReserved2:    LongInt;
   gReserved3:    LongInt;
   clientData:    LongInt;
   notifyProc:    NotificationProc;            {notification procedure}
END;

AuthSetupLocalIdentityPB = RECORD
   qLink:         Ptr;
   reserved1:     LongInt;
   reserved2:     LongInt;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LongInt;
   reqCode:       Integer;
   reserved:      ARRAY[1..2] OF LongInt;
```

```
   serverHint:    AddrBlock;
   dsRefNum:      Integer;
   callID:        LongInt;
   identity:      AuthIdentity;
   gReserved1:    LongInt;
   gReserved2:    LongInt;
   gReserved3:    LongInt;
   clientData:    LongInt;
   aReserved:     LongInt;
   userName:      RStringPtr;               {user name}
   password:      RStringPtr;               {user password}
END;

AuthChangeLocalIdentityPB = RECORD
   qLink:         Ptr;
   reserved1:     LongInt;
   reserved2:     LongInt;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LongInt;
   reqCode:       Integer;
   reserved:      ARRAY[1..2] OF LongInt;
   serverHint:    AddrBlock;
   dsRefNum:      Integer;
   callID:        LongInt;
   identity:      AuthIdentity;
   gReserved1:    LongInt;
   gReserved2:    LongInt;
   gReserved3:    LongInt;
   clientData:    LongInt;
   aReserved:     LongInt;
   userName:      RStringPtr;               {user name}
   password:      RStringPtr;               {current password}
   newPassword:   RStringPtr;               {new password}
END;

AuthLockLocalIdentityPB = RECORD
   qLink:            Ptr;
   reserved1:        LongInt;
   reserved2:        LongInt;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LongInt;
   reqCode:          Integer;
```

```
   reserved:         ARRAY[1..2] OF LongInt;
   serverHint:       AddrBlock;
   dsRefNum:         Integer;
   callID:           LongInt;
   identity:         AuthIdentity;
   gReserved1:       LongInt;
   gReserved2:       LongInt;
   gReserved3:       LongInt;
   clientData:       LongInt;
   theLocalIdentity: LocalIdentity; {local identity}
   name:             StringPtr;     {name of the app that denied delete}
END;

AuthUnlockLocalIdentityPB = RECORD
   qLink:            Ptr;
   reserved1:        LongInt;
   reserved2:        LongInt;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LongInt;
   reqCode:          Integer;
   reserved:         ARRAY[1..2] OF LongInt;
   serverHint:       AddrBlock;
   dsRefNum:         Integer;
   callID:           LongInt;
   identity:         AuthIdentity;
   gReserved1:       LongInt;
   gReserved2:       LongInt;
   gReserved3:       LongInt;
   clientData:       LongInt;
   theLocalIdentity: LocalIdentity;            {local identity}
   userName:         RStringPtr;               {user name}
   password:         RStringPtr;               {user password}
END;

AuthRemoveLocalIdentityPB = RECORD
   qLink:         Ptr;
   reserved1:     LongInt;
   reserved2:     LongInt;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LongInt;
   reqCode:       Integer;
   reserved:      ARRAY[1..2] OF LongInt;
```

```
   serverHint:    AddrBlock;
   dsRefNum:      Integer;
   callID:        LongInt;
   identity:      AuthIdentity;
   gReserved1:    LongInt;
   gReserved2:    Longint;
   gReserved3:    LongInt;
   clientData:    LongInt;
   aReserved:     LongInt;
   userName:      RStringPtr;                  {user name}
   password:      RStringPtr;                  {current password}
END;

AuthBindSpecificIdentityPB = RECORD
   qLink:         Ptr;
   reserved1:     LongInt;
   reserved2:     LongInt;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LongInt;
   reqCode:       Integer;
   reserved:      ARRAY[1..2] OF LongInt;
   serverHint:    AddrBlock;
   dsRefNum:      Integer;
   callID:        LongInt;
   identity:      AuthIdentity;
   gReserved1:    LongInt;
   gReserved2:    LongInt;
   gReserved3:    LongInt;
   clientData:    LongInt;
   userIdentity:  AuthIdentity;          {binding identity}
   userRecord:    RecordIDPtr;           {User record}
   userKey:       AuthKeyPtr;            {AOCE key for the user}
END;

AuthUnbindSpecificIdentityPB = RECORD
   qLink:         Ptr;
   reserved1:     LongInt;
   reserved2:     LongInt;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LongInt;
   reqCode:       Integer;
   reserved:      ARRAY[1..2] OF LongInt;
```

```
   serverHint:    AddrBlock;
   dsRefNum:      Integer;
   callID:        LongInt;
   identity:      AuthIdentity;
   gReserved1:    LongInt;
   gReserved2:    LongInt;
   gReserved3:    LongInt;
   clientData:    LongInt;
   userIdentity:  AuthIdentity;                {identity to be deleted}
END;

AuthGetSpecificIdentityInfoPB = RECORD
   qLink:         Ptr;
   reserved1:     LongInt;
   reserved2:     LongInt;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LongInt;
   reqCode:       Integer;
   reserved:      ARRAY[1..2] OF LongInt;
   serverHint:    AddrBlock;
   dsRefNum:      Integer;
   callID:        LongInt;
   identity:      AuthIdentity;
   gReserved1:    LongInt;
   gReserved2:    LongInt;
   gReserved3:    LongInt;
   clientData:    LongInt;
   userIdentity:  AuthIdentity;          {identity of initiator}
   userRecord:    RecordIDPtr;           {User record}
END;

AuthGetCredentialsPB = RECORD
   qLink:              Ptr;
   reserved1:          LongInt;
   reserved2:          LongInt;
   ioCompletion:       ProcPtr;
   ioResult:           OSErr;
   saveA5:             LongInt;
   reqCode:            Integer;
   reserved:           ARRAY[1..2] OF LongInt;
   serverHint:         AddrBlock;
   dsRefNum:           Integer;
   callID:             LongInt;
```

```
   identity:            AuthIdentity;
   gReserved1:          LongInt;
   gReserved2:          LongInt;
   gReserved3:          LongInt;
   clientData:          LongInt;
   userIdentity:        AuthIdentity;        {identity of initiator}
   recipient:           RecordIDPtr;         {AOCE name of recipient}
   sessionKey:          AuthKeyPtr;          {session key}
   expiry:              UTCTime;             {desired/actual expiration}
   credentialsLength:   LongInt;             {max/actual credentials size}
   credentials:         Ptr;                 {credentials buffer}
END;

AuthMakeProxyPB = RECORD
   qLink:            Ptr;
   reserved1:        LongInt;
   reserved2:        LongInt;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LongInt;
   reqCode:          Integer;
   reserved:         ARRAY[1..2] OF LongInt;
   serverHint:       AddrBlock;
   dsRefNum:         Integer;
   callID:           LongInt;
   identity:         AuthIdentity;
   gReserved1:       LongInt;
   gReserved2:       LongInt;
   gReserved3:       LongInt;
   clientData:       LongInt;
   userIdentity:     AuthIdentity;  {identity of principal}
   recipient:        RecordIDPtr;   {AOCE name of recipient}
   firstValid:       UTCTime;       {time at which proxy becomes valid}
   expiry:           UTCTime;       {time at which proxy expires}
   authDataLength:   LongInt;       {size of authorization data}
   authData:         Ptr;           {pointer to authorization data}
   proxyLength:      LongInt;       {max/actual proxy size}
   proxy:            Ptr;           {proxy buffer}
   intermediary:     RecordIDPtr;   {record ID of intermediary}
END;

AuthTradeProxyForCredentialsPB = RECORD
   qLink:               Ptr;
   reserved1:           LongInt;
```

```
   reserved2:           LongInt;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LongInt;
   reqCode:             Integer;
   reserved:            ARRAY[1..2] OF LongInt;
   serverHint:          AddrBlock;
   dsRefNum:            Integer;
   callID:              LongInt;
   identity:            AuthIdentity;
   gReserved1:          LongInt;
   gReserved2:          LongInt;
   gReserved3:          LongInt;
   clientData:          LongInt;
   userIdentity:        AuthIdentity;       {identity of intermediary}
   recipient:           RecordIDPtr;        {AOCE name of recipient}
   sessionKey:          AuthKeyPtr;         {session key}
   expiry:              UTCTime;            {desired/actual expiration}
   credentialsLength:   LongInt;            {max/actual credentials size}
   credentials:         Ptr;                {credentials buffer}
   proxyLength:         LongInt;            {actual proxy size}
   proxy:               Ptr;                {buffer containing proxy}
   principal:           RecordIDPtr;        {record ID of principal}
END;

AuthResolveCreationIDPB = RECORD
   qLink:          Ptr;
   reserved1:      LongInt;
   reserved2:      LongInt;
   ioCompletion:   ProcPtr;
   ioResult:       OSErr;
   saveA5:         LongInt;
   reqCode:        Integer;
   reserved:       ARRAY[1..2] OF LongInt;
   serverHint:     AddrBlock;
   dsRefNum:       Integer;
   callID:         LongInt;
   identity:       AuthIdentity;
   gReserved1:     LongInt;
   gReserved2:     LongInt;
   gReserved3:     LongInt;
   clientData:     LongInt;
   userRecord:     RecordIDPtr;{User record}
   bufferLength:   LongInt;    {buffer size}
```

```
      buffer:        Ptr;        {buffer to hold creation IDs}
      totalMatches:  LongInt;    {total number of matching names found}
      actualMatches: LongInt;    {number of matches returned in the buffer}
   END;

AuthGetUTCTimePB = RECORD
      qLink:         Ptr;
      reserved1:     LongInt;
      reserved2:     LongInt;
      ioCompletion:  ProcPtr;
      ioResult:      OSErr;
      saveA5:        LongInt;
      reqCode:       Integer;
      reserved:      ARRAY[1..2] OF LongInt;
      serverHint:    AddrBlock;
      dsRefNum:      Integer;
      callID:        LongInt;
      identity:      AuthIdentity;
      gReserved1:    LongInt;
      gReserved2:    LongInt;
      gReserved3:    LongInt;
      clientData:    LongInt;
      pRLI:          PackedRLIPtr;  {packed RLI of the dNode}
      theUTCTime:    UTCTime;       {current UTC(GMT) time in seconds
                                     since 1/1/1904}
      theUTCOffset:  UTCOffset;     {offset from UTC(GMT) seconds east
                                     of Greenwich}
   END;

   AuthMakeChallengePB = RECORD
      qLink:                 Ptr;
      reserved1:             LongInt;
      reserved2:             LongInt;
      ioCompletion:          ProcPtr;
      ioResult:              OSErr;
      saveA5:                LongInt;
      reqCode:               Integer;
      reserved:              ARRAY[1..2] OF LongInt;
      serverHint:            AddrBlock;
      dsRefNum:              Integer;
      callID:                LongInt;
      identity:              AuthIdentity;
      gReserved1:            LongInt;
      gReserved2:            LongInt;
```

```
   gReserved3:             LongInt;
   clientData:             LongInt;
   key:                    AuthKeyPtr;    {unencrypted session key}
   challenge:              Ptr;           {encrypted challenge}
   challengeBufferLength:  LongInt;       {length of challenge buffer}
   challengeLength:        LongInt;       {length of encrypted challenge}
END;

AuthMakeReplyPB = RECORD
   qLink:               Ptr;
   reserved1:           LongInt;
   reserved2:           LongInt;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LongInt;
   reqCode:             Integer;
   reserved:            ARRAY[1..2] OF LongInt;
   serverHint:          AddrBlock;
   dsRefNum:            Integer;
   callID:              LongInt;
   identity:            AuthIdentity;
   gReserved1:          LongInt;
   gReserved2:          LongInt;
   gReserved3:          LongInt;
   clientData:          LongInt;
   key:                 AuthKeyPtr;       {unencrypted session key}
   challenge:           Ptr;              {encrypted challenge}
   reply:               Ptr;              {encrypted reply}
   replyBufferLength:   LongInt;          {length of challenge buffer}
   challengeLength:     LongInt;          {length of encrypted challenge}
   replyLength:         LongInt;          {length of encrypted reply}
END;

AuthVerifyReplyPB = RECORD
   qLink:            Ptr;
   reserved1:        LongInt;
   reserved2:        LongInt;
   ioCompletion:     ProcPtr;
   ioResult:         OSErr;
   saveA5:           LongInt;
   reqCode:          Integer;
   reserved:         ARRAY[1..2] OF LongInt;
   serverHint:       AddrBlock;
   dsRefNum:         Integer;
```

```
   callID:           LongInt;
   identity:         AuthIdentity;
   gReserved1:       LongInt;
   gReserved2:       LongInt;
   gReserved3:       LongInt;
   clientData:       LongInt;
   key:              AuthKeyPtr;       {unencrypted session key}
   challenge:        Ptr;              {encrypted challenge}
   reply:            Ptr;              {encrypted reply}
   challengeLength:  LongInt;          {length of encrypted challenge}
   replyLength:      LongInt;          {length of encrypted reply}
END;

AuthDecryptCredentialsPB = RECORD
   qLink:               Ptr;
   reserved1:           LongInt;
   reserved2:           LongInt;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LongInt;
   reqCode:             Integer;
   reserved:            ARRAY[1..2] OF LongInt;
   serverHint:          AddrBlock;
   dsRefNum:            Integer;
   callID:              LongInt;
   identity:            AuthIdentity;
   gReserved1:          LongInt;
   gReserved2:          LongInt;
   gReserved3:          LongInt;
   clientData:          LongInt;
   userIdentity:        AuthIdentity;  {user's identity}
   initiatorRecord:     RecordIDPtr;   {AOCE name of the initiator}
   sessionKey:          AuthKeyPtr;    {session key}
   expiry:              UTCTime;       {credentials expiration time}
   credentialsLength:   LongInt;       {actual credentials size}
   credentials:         Ptr;           {credentials to be decrypted}
   issueTime:           UTCTime;       {credentials expiration time}
   hasIntermediary:     Boolean;       {if true, an intermediary record
                                        was found in credentials}
   intermediary:        RecordIDPtr;   {record ID of the intermediary}
END;
```

```
OCESetupGetDirectoryInfoPB = RECORD
   qLink:          Ptr;
   reserved1:      LongInt;
   reserved2:      LongInt;
   ioCompletion:   ProcPtr;
   ioResult:       OSErr;
   saveA5:         LongInt;
   reqCode:        Integer;
   reserved:       ARRAY[1..2] OF LongInt;
   serverHint:     AddrBlock;
   dsRefNum:       Integer;
   callID:         LongInt;
   identity:       AuthIdentity;
   gReserved1:     LongInt;
   gReserved2:     LongInt;
   gReserved3:     LongInt;
   clientData:     LongInt;
   directoryName:  DirectoryNamePtr;    {catalog name}
   discriminator:  DirDiscriminator;    {discriminator for the catalog}
   recordID:       RecordIDPtr;         {record ID for the catalog}
   nativeName:     RStringPtr;          {user name in the catalog world}
   password:       RStringPtr;          {password in the catalog world}
END;

OCESetupAddDirectoryInfoPB = RECORD
   qLink:               Ptr;
   reserved1:           LongInt;
   reserved2:           LongInt;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LongInt;
   reqCode:             Integer;
   reserved:            ARRAY[1..2] OF LongInt;
   serverHint:          AddrBlock;
   dsRefNum:            Integer;
   callID:              LongInt;
   identity:            AuthIdentity;
   gReserved1:          LongInt;
   gReserved2:          LongInt;
   gReserved3:          LongInt;
   clientData:          LongInt;
   directoryRecordCID:  CreationID;    {creation ID for the catalog}
```

```
   recordID:            RecordIDPtr;   {record ID for the identity}
   password:            RStringPtr;    {password in the catalog world}
END;

OCESetupChangeDirectoryInfoPB = RECORD
   qLink:               Ptr;
   reserved1:           LongInt;
   reserved2:           LongInt;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LongInt;
   reqCode:             Integer;
   reserved:            ARRAY[1..2] OF LongInt;
   serverHint:          AddrBlock;
   dsRefNum:            Integer;
   callID:              LongInt;
   identity:            AuthIdentity;
   gReserved1:          LongInt;
   gReserved2:          LongInt;
   gReserved3:          LongInt;
   clientData:          LongInt;
   directoryRecordCID:  CreationID;    {creation ID for the catalog}
   recordID:            RecordIDPtr;   {record ID for the identity}
   password:            RStringPtr;    {password in the catalog world}
   newPassword:         RStringPtr;    {new password in the catalog}
END;

OCESetupRemoveDirectoryInfoPB = RECORD
   qLink:               Ptr;
   reserved1:           LongInt;
   reserved2:           LongInt;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LongInt;
   reqCode:             Integer;
   reserved:            ARRAY[1..2] OF LongInt;
   serverHint:          AddrBlock;
   dsRefNum:            Integer;
   callID:              LongInt;
   identity:            AuthIdentity;
   gReserved1:          LongInt;
   gReserved2:          LongInt;
   gReserved3:          LongInt;
```

```
   clientData:          LongInt;
   directoryRecordCID:  CreationID;       {creation ID for the catalog}
END;
```

Parameter Block Case Statement

```
AuthParamBlock = RECORD
   CASE INTEGER OF
   1: (header:                      AuthDirParamHeader);
   2: (bindIdentityPB:              AuthBindSpecificIdentityPB);
   3: (unbindIdentityPB:            AuthUnbindSpecificIdentityPB);
   4: (resolveCreationIDPB:         AuthResolveCreationIDPB);
   5: (getIdentityInfoPB:           AuthGetSpecificIdentityInfoPB);
   6: (addKeyPB:                    AuthAddKeyPB);
   7: (changeKeyPB:                 AuthChangeKeyPB);
   8: (deleteKeyPB:                 AuthDeleteKeyPB);
   9: (passwordToKeyPB:             AuthPasswordToKeyPB);
   10:(getCredentialsPB:            AuthGetCredentialsPB);
   11:(decryptCredentialsPB:        AuthDecryptCredentialsPB);
   12:(makeChallengePB:             AuthMakeChallengePB);
   13:(makeReplyPB:                 AuthMakeReplyPB);
   14:(verifyReplyPB:               AuthVerifyReplyPB);
   15:(getUTCTimePB:                AuthGetUTCTimePB);
   16:(makeProxyPB:                 AuthMakeProxyPB);
   17:(tradeProxyForCredentialsPB:  AuthTradeProxyForCredentialsPB);
   18:(getLocalIdentityPB:          AuthGetLocalIdentityPB);
   19:(unLockLocalIdentityPB:       AuthUnlockLocalIdentityPB);
   20:(lockLocalIdentityPB:         AuthLockLocalIdentityPB);
   21:(localIdentityQInstallPB:     AuthAddToLocalIdentityQueuePB);
   22:(localIdentityQRemovePB:      AuthRemoveFromLocalIdentityQueuePB);
   23:(setupLocalIdentityPB:        AuthSetupLocalIdentityPB);
   24:(changeLocalIdentityPB:       AuthChangeLocalIdentityPB);
   25:(removeLocalIdentityPB:       AuthRemoveLocalIdentityPB);
   26:(setupDirectoryIdentityPB:    OCESetupAddDirectoryInfoPB);
   27:(changeDirectoryIdentityPB:   OCESetupChangeDirectoryInfoPB);
   28:(removeDirectoryIdentityPB:   OCESetupRemoveDirectoryInfoPB);
   29:(getDirectoryIdentityInfoPB:  OCESetupGetDirectoryInfoPB);
END;

AuthParamBlockPtr = ^AuthParamBlock;
```

Authentication Manager Functions

Key Management

```
FUNCTION AuthPasswordToKey  (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthAddKey         (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthChangeKey      (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthDeleteKey      (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
```

Local Identity Management

```
FUNCTION AuthGetLocalIdentity
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthAddToLocalIdentityQueue
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthRemoveFromLocalIdentityQueue
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthSetupLocalIdentity
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthChangeLocalIdentity
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthLockLocalIdentity
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthUnlockLocalIdentity
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthRemoveLocalIdentity
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
```

Specific Identity Management

```
FUNCTION AuthBindSpecificIdentity
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthUnbindSpecificIdentity
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthGetSpecificIdentityInfo
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
```

Credentials Management

```
FUNCTION AuthGetCredentials  (paramBlock: AuthParamBlockPtr;
                              async: Boolean): OSErr;
FUNCTION AuthMakeProxy       (paramBlock: AuthParamBlockPtr;
                              async: Boolean): OSErr;
FUNCTION AuthTradeProxyForCredentials
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
```

Creation ID Resolution Management

```
FUNCTION AuthResolveCreationID
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
```

Time Service

```
UNCTION AuthGetUTCTime      (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
```

Non-ASDSP Authentication Utilities

```
FUNCTION AuthMakeChallenge  (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthMakeReply      (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthVerifyReply    (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthDecryptCredentials
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
```

AOCE Setup Catalog Management

```
FUNCTION OCESetupGetDirectoryInfo
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION OCESetupAddDirectoryInfo
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION OCESetupChangeDirectoryInfo
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION OCESetupRemoveDirectoryInfo
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
```

Application-Defined Routines

```
PROCEDURE MyCompletion      (paramBlock: AuthParamBlockPtr);
FUNCTION NotificationProc   (clientData: LongInt;
                             callValue: AuthLocalIdentityOp;
                             actionValue: AuthLocalIdentityLockAction;
                             identity: LocalIdentity): Boolean;
```

Assembly-Language Summary

Trap Macros

Trap Macro Requiring Routine Selectors

_oceTBDispatch

Selector	Routine
$0200	AuthBindSpecificIdentity
$0201	AuthUnbindSpecificIdentity
$0202	AuthResolveCreationID
$0203	AuthGetSpecificIdentityInfo
$0204	AuthGetLocalIdentity
$0205	AuthAddToLocalIdentityQueue
$0206	AuthRemoveFromLocalIdentityQueue
$0207	AuthAddKey
$0208	AuthChangeKey

Selector	Routine
$0209	AuthDeleteKey
$020A	AuthPasswordToKey
$020B	AuthGetCredentials
$020C	AuthDecryptCredentials
$020D	OCESetupRemoveDirectoryInfo
$020E	OCESetupGetDirectoryInfo
$020F	AuthMakeChallenge
$0210	AuthMakeReply
$0211	AuthVerifyReply
$0212	AuthMakeProxy
$0213	AuthTradeProxyForCredentials
$0214	AuthUnlockLocalIdentity
$0215	AuthLockLocalIdentity
$0216	AuthSetupLocalIdentity
$0217	AuthChangeLocalIdentity
$0218	AuthRemoveLocalIdentity
$0219	OCESetupAddDirectoryInfo
$021A	AuthGetUTCTime
$021B	OCESetupChangeDirectoryInfo

Result Codes

Result codes in the range of –1540 to –1609 are reserved for the Authentication Manager. Routines may also return result codes from other AOCE managers and standard Macintosh result codes such as `noErr` 0 (No error) and `fnfErr` –43 (File not found).

`noErr`	0	No error
`kOCEParamErr`	–50	Parameter error
`kOCEReadAccessDenied`	–1540	Read access denied
`kOCEWriteAccessDenied`	–1541	Write access denied
`kOCEAccessRightsInsufficient`	–1542	Stream needs to be authenticated, or not authorized, or someone other than agent trying to TPFC, or problem in server-to-server authentication
`kOCEUnsupportedCredentialsVersion`	–1543	Can’t read this version of the credentials
`kOCECredentialsProblem`	–1544	Couldn’t decrypt credentials
`kOCECredentialsImmature`	–1545	Credentials not yet valid
`kOCECredentialsExpired`	–1546	Current time is later than credentials expiration time

kOCEProxyImmature	–1547	Proxy not yet valid
kOCEProxyExpired	–1548	Current time is later than proxy expiration time
kOCEDisallowedRecipient	–1549	Recipient record ID does not appear in proxy
kOCENoKeyFound	–1550	No key was found
kOCEPrincipalKeyNotFound	–1551	Couldn't decode proxy because principal has no key
kOCERecipientKeyNotFound	–1552	The recipient key was not found
kOCEAgentKeyNotFound	–1553	Intermediary's key not found
kOCEKeyAlreadyRegistered	–1554	A key already exists
kOCEMalFormedKey	–1555	Key not derived properly from password
kOCEUndesirableKey	–1556	Password too short or resulting key is undesirable
kOCEWrongIdentityOrKey	–1557	Incorrect key for client
kOCEInitiatorKeyProblem	–1558	No key, or initiator's key changed
kOCEBadEncryptionMethod	–1559	The specified encryption method is not supported
kOCELocalIdentityDoesNotExist	–1560	Local identity has not been set up
kOCELocalAuthenticationFail	–1561	Local identity locked
kOCELocalIdentitySetupExists	–1562	Local identity setup exists, use AuthChangeLocalIdentity instead
kOCEDirectoryIdentitySetupExists	–1563	Catalog has already been set up
kOCEDirectoryIdentitySetupDoesNotExist	–1564	Catalog has not been set up
kOCENotLocalIdentity	–1565	You cannot unbind a local identity
kOCENoMoreIDs	–1566	Identity table is full
kOCEUnknownID	–1567	Identity passed is not valid
kOCEOperationDenied	–1568	Local identity operation denied
kOCEAmbiguousMatches	–1569	Ambiguous matches found in resolving CIDs
kOCENoASDSPWorkSpace	–1570	No ASDSP workspace passed
kOCEAuthenticationTrouble	–1571	Reply incorrect for the challenge sent
kOCENotLocal	–1610	Internal AOCE error
kOCETargetDirectoryInaccessible	–1613	Catalog server not responding
kOCENoSuchDNode	–1615	The dNode was not found
kOCEBadRecordID	–1617	Name and type incorrect for creation ID
kOCENoSuchRecord	–1618	No such record
kOCEMoreData	–1623	Buffer was too small to hold all available data
kOCEStreamCreationErr	–1625	An error occurred in creating the stream
kOCEDirectoryNotFoundErr	–1630	Catalog was not found in the list
kOCEOCESetupRequired	–1633	Setup of local identity required
kOCELengthError	–1637	The supplied buffer was too small

A P P E N D I X

PowerTalk Built-in Templates

This appendix describes some of the details of the AOCE templates that are built into PowerTalk. You can use this information to gain access to the information in these templates or to provide additional templates that work with and extend the built-in templates. AOCE templates are described in the chapter “AOCE Templates” in this book.

User Records

User records have a record type of aoce User (`kUserRecTypeBody`).

There are several standard attribute types in a User record. You can add new attribute types to User records, but each new attribute type must have a unique name. To ensure the uniqueness of the attribute type, start it with a four-character application signature registered with Apple Developer Services.

Attribute type aoce mailslots (`kMailSlotsAttrTypeBody`) contains the list of addresses for the user. The preferred address is duplicated in attribute type aoce pref mailslot (`kPrefMailAttrTypeBody`).

Attribute type aoce PersonInfo contains the text information found on the first User record information page (the Business Card page). The following lookup table describes the format of the information:

```
resource 'dett' (kUserInfoPageAspect + kDETAspectLookup, purgeable)
   {{
   {"aoce PersonInfo"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      { 'rstr', kName, 0,
        'rstr', kTitle, 0,
        'rstr', kCompanyName , 0,
        'rstr', kCompanyAddr , 0,
        'rstr', kMisc , 0 };
   }};
```

Attribute type aoce Picture contains the picture found on the first User record information page.

```
resource 'dett' (kUserInfoPageAspect + kDETAspectLookup, purgeable)
   {{
   {"aoce Picture"}, typeBinary,
```

```
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      { 'rest', kPicture, 0 };
   }};
```

Attribute type aoce Personal contains the information found on the second User record information page (the Personal Info page).

```
resource 'dett' (kUserInfoPageAspect + kDETAspectLookup, purgeable){{
   {"aoce Personal"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      { 'rstr', kPersonal1, 0,
        'rstr', kPersonal2 , 0,
        'rstr', kPersonal3 , 0 };
   }};
```

Attribute type aoce PhoneNumber contains one value per phone number in the third User record information page (the Phone Numbers page). The tag of each value tells the type of phone number: `'work'` for work, `'home'` for home, `'mobi'` for mobile, and `'othr'` for other. You can add new tags, but you must be careful to avoid duplicate use of the same tag. The attribute values have the following format (using `'work'` as an example—the other types vary only in the tag, not in the internal format):

```
resource 'dett' (kWorkPhoneAspect + kDETAspectLookup, purgeable) {
   {
   {"aoce PhoneNumber"}, 'work',
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
      'rstr', kDETAspectName, 0;
      'rstr', kPhoneNumber, 0;
      'rstr', kPhoneInfo, 0
      };
   }
};
```

Attributes of type aoce mailslots are used both in User records and in stand-alone attributes and are described in “Addresses” on page A-4.

The User record information pages are 350 pixels wide by 180 pixels high. The page selection pop-up menu is fixed at location (6, 180, 24, 339) (top, left, bottom, right). If you add your own information page to a User record, it is recommended that you include a page-identifying small icon at location (6, 156, 22, 172).

The picture on the first information page is 100 pixels wide and 100 pixels high.

The sort-order number of the information pages for the User record is: 1000 for the first (Business Card) page, 2000 for the second (Personal Info) page, 3000 for the third (Phone Numbers) page, and 20000 for the fourth (Electronic Addresses) page. The Electronic Addresses page has a much higher sort-order number than the others because it is intended always to come last, even if developers add additional information pages between it and the other ones.

The names of the built-in templates for User records are listed in Table A-1.

Table A-1 Names of AOCE templates for User records

Template name	Template type	Function
aoce User main aspect	Aspect	User record main aspect
aoce User Info-page Aspect	Aspect	Properties for first two pages
aoce User Info-page	Information page	First (Business Card) page
aoce User Persoanl Info-page	Information page	Second (Personal Info) page
aoce User Phone Aspect	Aspect	Properties for third page
aoce User Phone Info Page	Information page	Third (Phone Numbers) page
aoce Work Phone Aspect	Aspect	Aspect for work phone
aoce Work Phone Info-page	Information page	Information page for work phone
aoce Home Phone Aspect	Aspect	Aspect for home phone
aoce Home Phone Info-page	Information page	Information page for home phone
aoce Mobile Phone Aspect	Aspect	Aspect for mobile phone
aoce Mobile Phone Info-page	Info-page	Information page for mobile phone
aoce Other Phone Aspect	Aspect	Aspect for other phone
aoce Other Phone Info-page	Info-Information	Information page for other phone
Mail Info Page Aspect	Aspect	Properties for last page
Mail Info Page	Information page	Last (Electronic Addresses) page
Drop Send Aspect User	Aspect	Drop-send (forwarded to by groups and addresses as well)

NOTE The spelling “Persoanl” in the name of the second information page of a User record is correct as shown.

Group Records

Group records have a record type of aoce Group (`kGroupRecTypeBody`).

The only standard attribute type in a Group record is aoce Member (`kMemberAttrTypeBody`).

The Group record information pages are 277 pixels wide by 303 pixels high. The page-selection pop-up menu is fixed at location (7, 35, 26, 201) (top, left, bottom, right). If you add your own information page to a Group record, it is recommended that you include a page-identifying small icon at location (7, 11, 23, 27).

Addresses

Within User records and stand-alone attributes, electronic addresses are stored in attribute values of type aoce mailslots (`kMailSlotsAttrTypeBody`). The tag is the internal subtype of the address (`'entn'` for PowerShare, `'alan'` for AppleTalk, and so forth).

The `'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 by 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:”, one for Fields and one for String. The “View as:” string 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, so your address template does not need to handle drops.

Other Built-in Templates

There are several other templates built into PowerTalk. These include the template for the Find-in-Catalog feature, the Key Chain templates, and address templates for AppleTalk and PowerShare addresses. These other templates are subject to change without notice; for this reason, you should not modify, replace, or otherwise depend on them.

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. ASDSP is a secure version of AppleTalk Data Stream Protocol (ADSP).

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 received 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.

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 card An HFS file located on a user's local disk that contains a single record.

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.

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

Symbols

A

B

C

D

E

F

G

H

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Proof pages were created on an Apple LaserWriter Pro printer. Final page negatives were output directly from text files on an Optrotech SPrint 220 imagesetter. Line art was created using Adobe Illustrator™ and Adobe Photoshop™. PostScript™, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier.

LEAD WRITER
Paul Black

WRITERS
Paul Black, Dee Eduardo, Ed Fernandez, Michael Kline, Alan Spragens, Angela Ferguson, Dave Bice

DEVELOPMENTAL EDITOR
Antonio Padial

INDEX SPECIALIST
Laurel Rezeau

ILLUSTRATOR
Deb Dennis

COVER DESIGNER
Barbara Smyth

PRODUCTION EDITORS
Pat Christenson, Alan Morgenegg

PROJECT MANAGER
Patricia Eastman

Special thanks to Pablo Calamera, Harry Chesley, Mike Cleron, Jamie Doll, John Evans, Steve Falkenburg, Steve Fisher, Charlie Kim, Karen Lam, Sarah Lindsley, Martin Minow, S. G. Sangameswara, Eric Trehus, Atticus Tysen, R. C. Venkatraman

Acknowledgments to Rick Andrews, Joseph Aseo, Andy Atkins, Michael Bayer, Timo Bruck, Hermán Camarena, Victor Chang, Henry Chen, Godfrey DiGiorgi, Laurence Gathy, Bruce Gaya, Darren Giles, Gerri Gray, John Hammett, Carol Lee, Miki Lee, Barbara Martinez, Shantanu Narayen, David O'Rourke, Monica Pal, Gursharan Sidhu, Alex Solinski, and the entire AOCE team.