This chapter describes how to write a ***catalog service access module (CSAM),*** a device driver that gives PowerTalk users access to external catalogs. Read this chapter if you want to integrate an external catalog into an AOCE system. You do not need to read this chapter if you simply want to use the Standard Catalog Package or the Catalog Manager to obtain catalog services.

To write a CSAM, you must already be familiar with the Catalog Manager application program interface (API). It is essential that you read the chapters "AOCE Utilities" and "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* before reading this chapter. This chapter assumes that you understand the Catalog Manager's functions and data types.

Because a CSAM is implemented as a Macintosh device driver, you also need to be familiar with the Device Manager. For information about the Device Manager and writing a device driver, see *Inside Macintosh: Devices*.

To allow the user to add and remove your CSAM and its catalogs from an AOCE system, you need to provide an AOCE setup template. The chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces* describes how to write an AOCE template. The chapter "Service Access Module Setup" in this book describes the setup template specifically, including how the setup template adds and removes a CSAM and its catalogs from the Setup catalog.

This chapter provides a brief introduction to CSAMs. Then it describes

- the components of a CSAM
- a CSAM's driver resource, including the Open and Close driver subroutines
- a CSAM's catalog service and parse functions, which respond to requests from clients of the Catalog Manager
- the method of indicating the features a catalog can support
- the impact of various catalog features on the user's experience with a catalog

Introduction to Catalog Service Access Modules

The Catalog Manager provides a consistent interface for applications that use AOCE catalog services, regardless of whether the catalog is external to or part of AOCE software. Apple PowerShare catalogs and personal catalogs are part of AOCE software. Any other type of catalog is referred to as an ***external catalog.*** An external catalog is made available within an AOCE system by means of a CSAM, which supports the Catalog Manager API.

A CSAM provides these basic functions:

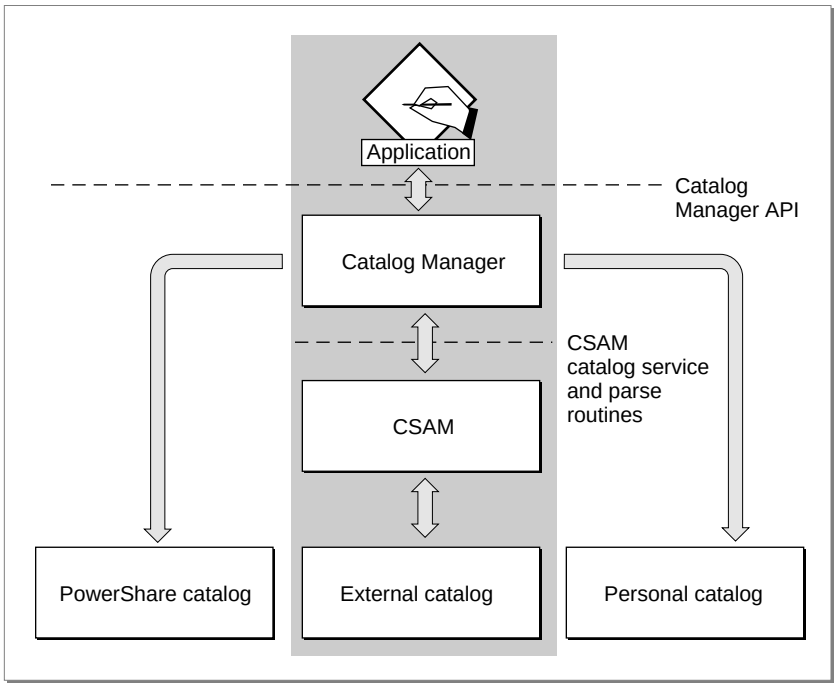
- accepting Catalog Manager requests
- translating the requests into a form that its external catalog understands
- processing the requests, including activities such as obtaining information from the external catalog and adding information to the external catalog

- translating information for the Catalog Manager client into AOCE data formats such as records and attributes
- returning the information to the Catalog Manager

AOCE data formats are described in detail in the chapter "AOCE Utilities." The Catalog Manager API is described in the chapter "Catalog Manager." Both chapters are in *Inside Macintosh: AOCE Application Interfaces*.

A CSAM is not invoked directly by an application but indirectly through the Catalog Manager. The CSAM hides any underlying differences in how data is accessed and stored in its external catalog. For example, suppose an application wants to add a record to a catalog. The application calls the `DirAddRecord` function. If the target catalog is an external catalog, the Catalog Manager passes the request to the CSAM that supports that catalog. The CSAM then adds the record to its catalog and provides the creation ID of the new record. Thus, a Catalog Manager client can interact with all catalogs in the same way and can use standard AOCE data types to manipulate data. Figure 3-1 shows the relationship of an application, the Catalog Manager, a CSAM, and an external catalog. Although the figure shows a single external catalog, a CSAM can actually support any number of catalogs. The application and the Catalog Manager communicate through the Catalog Manager API. The Catalog Manager and the CSAM communicate through the CSAM's catalog service and parse functions, which are introduced in the next section.

Figure 3-1 Relationship of an application, the Catalog Manager, and a CSAM



Every CSAM should support Catalog Manager requests to

- examine the contents of a dNode by real-time browsing, a search mechanism, or both
- enumerate the attribute types within a record
- look up attribute values
- detect changes within a dNode or a record
- get access controls for a dNode, record, or attribute type

A CSAM resides on a user's Macintosh computer and provides personal access to an external catalog. The catalog itself can exist anywhere—on the user's Macintosh, on a network server, or at a remote site accessed by a modem connection.

You can package a CSAM as a stand-alone driver file or as part of an AOCE messaging service access module. A ***messaging service access module (MSAM)*** translates and transfers messages between an AOCE messaging system and another messaging system. If you choose the stand-alone option, you provide a file of type `'dsam'` that contains the resources described in the section "Writing a Driver Resource for a CSAM" beginning on page 3-7. The file must also contain the resources that constitute your setup template. If you package your CSAM with a messaging service access module, include your CSAM, its setup template, and the MSAM in a file of type `'csam'` (for "combined SAM"). MSAMs are documented in the chapter "Messaging Service Access Modules" in this book. The setup template resources are described in the chapter "Service Access Module Setup" in this book.

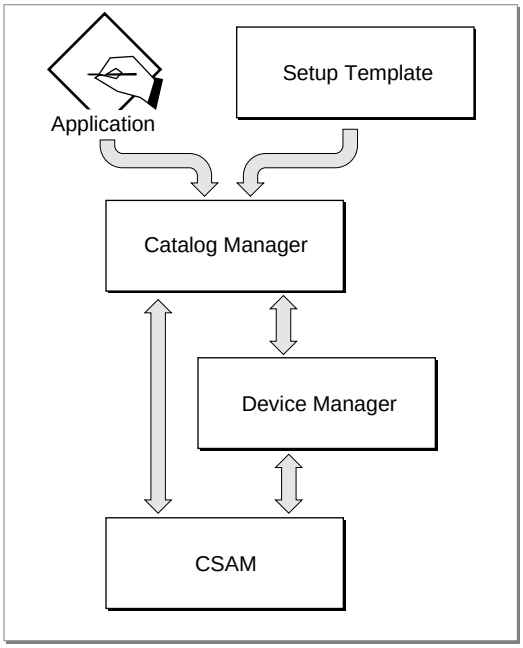
Note
For historical reasons, the string `dsam` (or `DSAM`) rather than `csam` (or `CSAM`) is often part of a function name, field name, or data type name referring to a CSAM. ◆

Components of a CSAM

A CSAM consists of two main components: a driver resource that includes at least your driver's Open and Close subroutines, and the collection of functions that implement Catalog Manager functions. In addition, you must provide an AOCE setup template that allows the user to add, remove, and configure the CSAM and its catalogs. It can be helpful to think of the template as the third component of a CSAM product.

The setup template consists of a set of associated resources that reside in the resource fork of the CSAM file. A template code resource calls the Catalog Manager functions that add, remove, and configure the CSAM and its catalogs. The setup template is described in the chapter "Service Access Module Setup" in this book. Figure 3-2 shows the calling relationships between an application, a setup template, the Catalog Manager, the Device Manager, and a CSAM.

Figure 3-2 Calling relationships



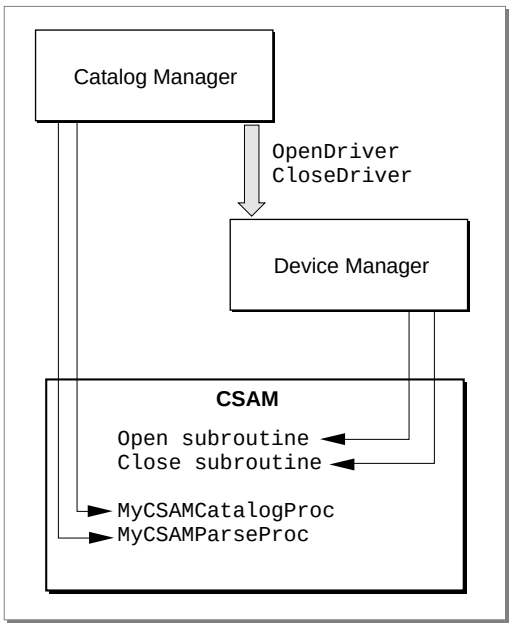
Requests for catalog services tend to be real-time in nature. Because Macintosh device drivers lend themselves to implementing real-time responses, you implement a CSAM as a Macintosh device driver.

A CSAM has two interfaces to Macintosh system software—one through the Device Manager and the other through the Catalog Manager. For the Device Manager interface, you must provide Open and Close driver subroutines. The Catalog Manager calls the Device Manager to open and close your driver. The Device Manager, in turn, calls your driver's Open and Close subroutines. You may provide the Prime, Status, and Control driver subroutines in accordance with the needs of your driver, but the Catalog Manager does not call these subroutines to communicate with your driver. The Open and Close driver subroutines are described in the section "Writing a Driver Resource for a CSAM" beginning on page 3-7. The Prime, Status, and Control driver subroutines are described in *Inside Macintosh: Devices*.

For the Catalog Manager interface, you provide a catalog service function and a parse function. When an application calls a Catalog Manager function, the Catalog Manager calls the CSAM's catalog service or parse function and passes it the application's catalog service request. A ***catalog service function*** accepts requests for catalog services from the Catalog Manager and calls CSAM-defined routines to implement those services. A ***parse function*** accepts requests to parse data about the CSAM's catalogs and their contents and calls CSAM-defined routines to implement those services.

Figure 3-3 illustrates who calls your driver subroutines and your catalog service and parse functions.

Figure 3-3 Who calls the CSAM driver subroutines and the catalog service and parse functions



The sections that follow describe the CSAM's driver resource and the CSAM catalog service and parse functions.

Writing a Driver Resource for a CSAM

This section provides information about the required resources that constitute your CSAM's device driver.

The driver resource that you must provide in your CSAM, like all resources, has a type, a resource ID, a resource name, and resource attributes. The resource type is `'DRVR'`. You may set your `'DRVR'` resource ID to any valid value. The Catalog Manager properly installs your driver. The `'DRVR'` resource name must be the same as the name of your driver. This point is illustrated later in this section.

For your driver to work properly with the Catalog Manager, you must configure your `'DRVR'` resource as follows:

- Set the `resSysHeap` resource attribute to guarantee that your driver is loaded into the system heap.
- Set the `resLocked` resource attribute so that your driver is always available and nonrelocatable in memory.

You may set other attributes needed for your CSAM. See *Inside Macintosh: Devices* for more detailed information about the 'DRVR' resource. See the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox* for information on resource attributes.

A resource of type `'DRVR'` contains header information and the driver's subroutines. The header information specifies certain settings for the driver and the offsets of the Open, Close, Status, Prime, and Control subroutines. The book *Inside Macintosh: Devices* provides information on setting up the header information. The header is followed by the driver subroutines themselves. Listing 3-1 illustrates the header of a sample CSAM's driver resource in Rez format.

Listing 3-1 A sample CSAM's driver resource header

```
#define DriverID          0x0b // unused, placeholder value

resource 'DRVR' (DriverID, ".SampleCSAM", sysheap, locked)
{
   /* driver flags */
   needLock, dontNeedTime, dontNeedGoodbye, noStatusEnable,
   ctlEnable, noWriteEnable, noReadEnable,

   0,             /* driver delay in ticks */
   0,             /* desk accessory mask */
   0,             /* desk accessory menu */
   ".SampleCSAM", /* driver name */
   /* the driver code follows the header fields */
};
```

Your Open subroutine handles initialization functions. It must do the following:

- Allocate and initialize any memory required. You need to allocate memory now because you cannot do so when the Catalog Manager calls your catalog service or parse function with an asynchronous request. Your CSAM must allocate its memory in the system heap and store the handle to the memory in the `dCtlstorage` field of the device control entry (`DCtlEntry`) structure.
- Call the `DirInstantiateDSAM` function to provide the Catalog Manager with pointers to your catalog service and parse functions. You can also provide a pointer to your private data, which the Catalog Manager passes back to you when it calls your catalog service and parse functions.
- Do any other preparation required to make the CSAM ready to receive and process service requests.

Your Open subroutine is always called synchronously.

In your Close subroutine, you should release any memory that you allocated in your Open subroutine. The Close subroutine is always called synchronously.

Depending on the needs of your driver, your Status, Prime, and Control subroutines may perform some work or simply return if called.

Note
The Device Manager interface requires you to use some assembly language. You can write your driver subroutines in a high-level language if you provide a dispatching mechanism, written in assembly language, between the Device Manager and the subroutines. See *Inside Macintosh: Devices* for instructions on writing subroutines in a high-level language and for detailed descriptions of all of the driver subroutines. ◆

When writing a device driver, you ordinarily write software that installs the driver in the Device Manager's unit table and opens the driver. For a CSAM, you do not need to provide software to install and open your driver directly. Instead, an AOCE setup template that you provide calls the `DirAddDSAM` function. This causes the Catalog Manager to install and open your driver. (Setup templates are discussed in the chapter "Service Access Module Setup" in this book.)

In addition to the `'DRVR'` resource, you must also provide a resource of type `'STR '` containing a single string that is both the name of your driver and the name of your `'DRVR'` resource. This string resource must have the resource name `DashName`. If you use another name for the string resource, the Catalog Manager will not be able to install your driver. Listing 3-2 illustrates the string resource. The name contained in this string resource must be the same as the name of the `'DRVR'` resource.

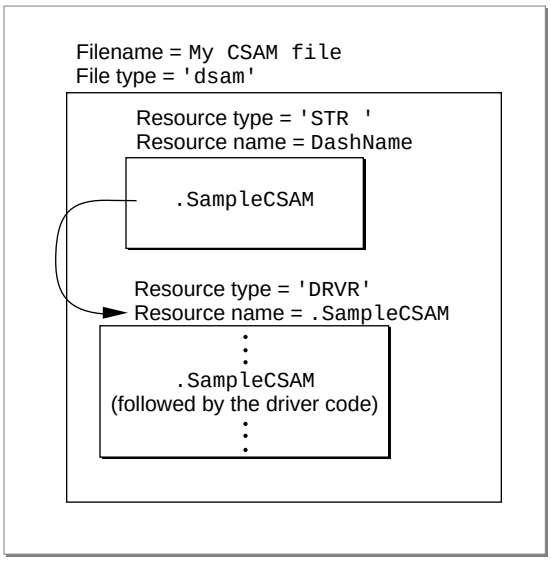
Listing 3-2 A CSAM's driver name string resource

```
/* The Driver's name must be in the resource named DashName." */
 resource 'STR ' (128, "DashName", purgeable ) {
   ".SampleCSAM"
};
```

Listing 3-1, Listing 3-2, and Figure 3-4 illustrate the following example. A file named My CSAM File contains a CSAM. The filename can be any string and is editable by a user. The file contains a `'STR '` resource named `DashName` that contains the string `.SampleCSAM`. The file also contains a `'DRVR'` resource whose resource name is `.SampleCSAM`. The driver itself is also named `.SampleCSAM`. The content of the string resource, the name of the `'DRVR'` resource, and the name of the driver are all the same.

Note that a driver name should always start with a period, followed by printable uppercase or lowercase characters, not to exceed a total of 31 characters.

Figure 3-4 Relationship of 'DRVR' and 'STR ' resources



Responding to the Catalog Manager

When an application makes a request for catalog services and specifies an external catalog for which your CSAM is responsible, the Catalog Manager calls your CSAM's catalog service or parse function. The catalog service and parse functions are essentially dispatching functions that receive all Catalog Manager requests. They in turn call other functions that you provide to service the request.

A CSAM does not need to support every function in the Catalog Manager API. The Catalog Manager itself handles calls to the `DirGetDirectoryInfo`, `DirGetExtendedDirectoriesInfo`, `DirEnumerateDirectoriesGet`, and `DirEnumerateDirectoriesParse` functions and, therefore, does not pass these requests to a CSAM. Other Catalog Manager functions that are not passed to a CSAM include

- `DirAddADAPDirectory`
- `DirNetSearchADAPDirectoriesGet`
- `DirNetSearchADAPDirectoriesParse`
- `DirFindADAPDirectoryByNetSearch`
- `DirCreatePersonalDirectory`
- `DirOpenPersonalDirectory`
- `DirClosePersonalDirectory`
- `DirMakePersonalDirectoryRLI`
- `DirGetOCESetupRefNum`

You must provide a dispatch function. You can provide both a catalog service function and a parse function for this purpose. However, because Catalog Manager request codes for catalog service and parse requests do not overlap, you can process all Catalog Manager requests through a single dispatch function. To do this, specify the same address for your catalog service function and your parse function when you call the `DirInstantiateDSAM` function.

The Catalog Service Function

The Catalog Manager calls your catalog service function when an application calls a Catalog Manager function (other than one of the parse functions) and specifies a catalog that you support. Your catalog service function must determine the type of request that the application is making and then service that request.

The catalog service function has the following declaration:

```
pascal OSErr MyDSAMDirProc (Ptr dsamData,
                            DirParamBlockPtr paramBlock,
                            Boolean async);
```

The `dsamData` parameter contains the private value that you provided to the `DirInstantiateDSAM` function in the `dsamData` field of that function's parameter block. You define this value for your own use. Typically, it is a pointer to your private data area. The `paramBlock` parameter contains a pointer to the `DirParamBlock` parameter block that the application provided to the Catalog Manager when the application made the service request. The `async` parameter is a Boolean value that specifies if the request must be processed synchronously or asynchronously. If this parameter is `true`, you must process the request asynchronously; otherwise, you process the request synchronously.

You determine the type of request by examining the `reqCode` field of the `DirParamBlock` parameter block. Requests for catalog services map one-to-one to functions in the Catalog Manager API. The method by which you service the request (that is, implement the Catalog Manager function) is up to you. See the section "Data Types and Constants" beginning on page 3-42 for a complete list of request codes for Catalog Manager requests. See the function descriptions in the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* for information on the type of service each function performs, the behavior of the function, and the information it returns.

When an application calls a Catalog Manager function synchronously, the Catalog Manager passes the request to your CSAM within the calling application's context. Therefore, the CSAM can allocate, move, or purge memory and can call any function. The CSAM must process a synchronous request immediately. (See *Inside Macintosh: Processes* for a discussion of application context.)

When an application calls a Catalog Manager function asynchronously, the Catalog Manager passes the request to your CSAM at interrupt time. You cannot allocate, move, or purge memory at interrupt time, nor can you call a function that allocates, moves, or purges memory. If you can service the asynchronous request immediately—that is, if you

can service the request without performing tasks that are likely to consume a relatively large amount of time, such as an I/O operation—do so. Otherwise, your catalog service function should place the request in a private queue that it maintains and return control to the Catalog Manager with a result code of `noErr`. The Catalog Manager will already have set the `ioResult` field of the `DirParamBlock` parameter block to 1 before passing the asynchronous request to your catalog service function. As your function receives time to execute, service the request.

The CSAM can defer processing an asynchronous request until it is convenient to complete the request. It can install a VBL task, a Time Manager task, a Deferred Task Manager task, or a Notification Manager task to ensure that it receives system time at some point in the future. See *Inside Macintosh: Processes* for more information on these topics.

Note
When you have insufficient memory to service an asynchronous request, you should return an error. However, before returning, you can attempt to acquire additional memory for future requests. Set the `dNeedTime` flag in the `dCtlflags` field in your driver's `DCtlEntry` structure. Later, after a process calls the `SystemTask` or `WaitNextEvent` function, the Device Manager calls your Control subroutine with the `accRun` control code. At this time, you can safely allocate memory.

Do not queue an asynchronous request for which you have insufficient memory in the hope that you can acquire the memory later and successfully complete the request. This may result in a system freeze condition. ◆

Your catalog service function returns both a function result and a value in the `ioResult` field of the `DirParamBlock` parameter block to indicate the outcome of its handling of the request. For each type of service request (function) that you process, you should return only those result codes that are defined by the Catalog Manager for the function. The description of each Catalog Manager function provides the result codes that a given function can return.

If your function was called synchronously, set the `ioResult` field and return the appropriate function result code when you finish servicing the request.

If your function was called asynchronously, do the following when you finish servicing the request: Set the `ioResult` field to the appropriate result code. If the application provided a completion routine (the value of the `ioCompletion` field of the `DirParamBlock` parameter block is not `nil`), restore the application's A5 register by setting register A5 to the value of the `saveA5` field of the `DirParamBlock` parameter block and call the application's completion routine; otherwise, return. When the completion routine returns control to your catalog service function, you may service another pending request or return.

Listing 3-3 is an example of a simple catalog service function, the `DoMyDSAMDirProc` function, that determines the type of request and then calls another function to service the request. `DoMyDSAMDirProc` passes the called function a pointer to the CSAM's global data area, `myGlobalInfoPtr`. This is the value the CSAM originally gave to the

Catalog Manager when it called the `DirInstantiateDSAM` function. The Catalog Manager passes the value back to the catalog service function to use in servicing the request. In this example, the functions that service a particular catalog service request, such as the `DoProcessDirGetDNodeMetaInfoReqt` function, set the `ioResult` field of the `DirParamBlock` parameter block. Before returning, the `DoMyDSAMDirProc` function calls the `DoProcessCallCompletion` function, which calls the completion routine if the calling application specified one. See Listing 3-6 on page 3-28 for an example of calling an application's completion routine.

Listing 3-3 A catalog service function

```
pascal OSErr DoMyDSAMDirProc(
   register Ptr               myGlobalInfoPtr,
   register DirParamBlockPtr  myParamBlock,
   Boolean                    async)
{
   switch (myParamBlock->header.reqCode) {   /* determine type of request */
   case kDirGetDirectoryIcon:
      DoProcessDirGetDirIconRequest(myGlobalInfoPtr, myParamBlock, async);
      break;
   case kDirGetDNodeMetaInfo:
      DoProcessDirGetDNodeMetaInfoReqt(myGlobalInfoPtr, myParamBlock, async);
      break;
   case kDirGetRecordMetaInfo:
      DoProcessDirGetRecrdMetaInfoReqt(myGlobalInfoPtr, myParamBlock, async);
      break;
   /* process other catalog service requests */
   }
   return (DoProcessCallCompletion(myParamBlock->header.ioResult, async));
}
```

The Catalog Manager defers calling your catalog service function until a time, sometimes called *deferred-task time*, when your function will work properly if the Macintosh is using virtual memory. See *Inside Macintosh: Memory* for information about memory management issues, including virtual memory.

The Parse Function

The Catalog Manager calls your parse function each time an application makes a parse request for a catalog that you support. A parse request corresponds to one of the Catalog Manager's parse functions, such as `DirLookupParse`, `DirEnumerateParse`, and so forth. Your parse function must determine the type of parse request that the application is making and then service that request.

The parse function has the following declaration:

```
pascal OSErr MyDSAMDirParseProc (Ptr dsamData,
                                 DirParamBlockPtr paramBlock,
                                 Boolean async);
```

The information in the section "The Catalog Service Function" beginning on page 3-11 also applies to the parse function. That information is not repeated here.

When you service a Catalog Manager parse request, you return information to the application by two methods. The first method, common to all Catalog Manager requests, consists of storing information in the appropriate fields of the `DirParamBlock` parameter block. The second, unique to parse requests, consists of passing data in predefined units to an application's callback routine.

It might be helpful to review here how Catalog Manager parse functions work. Each Catalog Manager parse function is paired with an associated get function. The `DirEnumerateDirectoriesGet`/`DirEnumerateDirectoriesParse` and `DirLookupGet`/`DirLookupParse` functions are examples of the get/parse function pairs in the Catalog Manager API. An application calls a Catalog Manager get function to obtain information about catalogs, records, attribute types, and so forth. If the target catalog is a catalog that you support, the Catalog Manager calls your CSAM's catalog service function to service the request. You place the requested data into a buffer provided by the application. You can use any format you wish for the data in this buffer; the data is therefore unreadable by the application. To retrieve the data from the buffer in a format that it understands, the application calls the corresponding Catalog Manager parse function, providing a pointer to a callback routine. The Catalog Manager, in turn, calls your CSAM's parse function. Your parse function passes data to the application by repeatedly calling the application's callback routine, each time passing it a defined chunk of data. The chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* provides descriptions of the application callback routines associated with different Catalog Manager parse functions and the type of data you need to return with each.

Note
Not all Catalog Manager get/parse function pairs work in exactly the same way. For example, most support starting or continuing an enumeration from a specified starting point, but some do not. Be sure to read the Catalog Manager function descriptions carefully to make sure your CSAM properly implements the Catalog Manager functions. ◆

You determine which Catalog Manager function the application has called by examining the `reqCode` field of the `DirParamBlock` parameter block. Then you process the request, just as you would when servicing a catalog service request. In addition, you call the application's callback routine as part of processing every parse request. You must set the A5 register to the value of the `saveA5` field of the `DirParamBlock` parameter block before calling the callback routine. You typically restore your own A5 register when you regain control.

Listing 3-4 illustrates how you call an application's callback routine. The `DoEnumerateParse` function is called by another CSAM function in the course of servicing the parse request that results from an application calling the `DirEnumerateParse` function. The `DoEnumerateParse` function gets a pointer to the `DirEnumerateParse` function's parameter block and a pointer to the buffer the CSAM previously filled in response to the `DirEnumerateGet` function. The application's callback routine expects to get a `DirEnumSpec` structure that provides information about one record, alias, pseudonym, or child dNode in a given dNode. Inside its main processing loop, the `DoEnumerateParse` function performs the following tasks:

- It initializes the `dataLength` fields inside the `DirEnumSpec` structure to the maximum size `RString` that the CSAM supports.
- It calls its `DoFillEnumSpec` function to extract data about one record, alias, pseudonym, or child dNode from the buffer the CSAM previously filled and stores the data in a `DirEnumSpec` structure. If this function does not return the `noErr` result code, `DoEnumerateParse` exits the loop immediately, knowing it has extracted all the data from the buffer or it has encountered an error.
- It sets register A5 to the application's register A5 so the callback routine can access the application's global variables and saves its own register A5 value.
- It calls the application's callback routine, passing it the value of the `clientData` field from the `DirEnumerateParse` parameter block and the enumeration specification just constructed.
- It restores register A5 to its own register A5 value.

The `DoEnumerateParse` function continues to execute the loop until it runs out of data to parse or encounters an error, or until the application's callback routine returns `true`.

Listing 3-4 Calling an application's callback routine

```
OSErr DoEnumerateParse (DirParamBlockPtr myParamBlock, Ptr buffer)
{
   DirEnumSpec    enumSpec;
   RString64      name, type;
   long           oldA5, saveSeq;
   Boolean        done = false;
   OSErr          myErr = 0;

   enumSpec.u.recordIdentifier.recordName = (RString*)&name;
   enumSpec.u.recordIdentifier.recordType = (RString*)&type;
   enumSpec.indexRatio = 0;

   while(!done) {
      name.dataLength = kRString64Size;
      type.dataLength = kRString64Size;
```

```
      /* extract data from the buffer and fill enumSpec appropriately */
      myErr = DoFillEnumSpec(buffer, &enumSpec);
      if (myErr != noErr) /* if no more data in the buffer, exit the loop */
         break;

      /* save my A5 register and call application's callback routine */
      oldA5 = SetA5(myParamBlock->enumerateParsePB.saveA5);
      done = (*myParamBlock->enumerateParsePB.eachEnumSpec)
                  (myParamBlock->enumerateParsePB.clientData, &enumSpec);

      /* restore my A5 register */
      (void) SetA5(oldA5);
   }
return myErr;
}
```

To avoid problems when virtual memory is in use, you must call an application's callback routine at deferred-task time. See the chapters "Virtual Memory Manager" in *Inside Macintosh: Memory* and "Deferred Task Manager" in *Inside Macintosh: Processes* for more information on the handling of virtual memory and deferred tasks.

Determining the Version of the Catalog Manager

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

Indicating the Features You Support

A catalog may not support all of the features of the Catalog Manager API. Therefore, you must identify to the Catalog Manager the features supported by each catalog to which your CSAM provides access. The Catalog Manager API defines the data type `DirGestalt` that consists of bits that specify the features supported by a given catalog.

This section defines those bits, sometimes referred to as *feature flags* or *capability flags*. The support or lack thereof for certain features affects the human interface of some components of PowerTalk. The impact of various feature settings on the human interface is discussed in "Human Interface Considerations" beginning on page 3-22.

The features represented by the bits can be grouped into six general categories (the corresponding bits are listed for each category):

- supplying identifying information
  - kSupportsDNodeNumberBit
  - kSupportsRecordCreationIDBit
  - kSupportsAttributeCreationIDBit
  - kSupportsPartialPathNamesBit
- pattern-matching for record names in an enumeration
  - kSupportsMatchAllBit
  - kSupportsBeginsWithBit
  - kSupportsExactMatchBit
  - kSupportsEndsWithBit
  - kSupportsContainsBit
- ordering the results of an enumeration
  - kSupportsOrderedEnumerationBit
  - kCanSupportNameOrderBit
  - kCanSupportTypeOrderBit
  - kSupportSortBackwardsBit
  - kSupportIndexRatioBit
- enumerating from a specified starting point
  - kSupportsEnumerationContinueBit
  - kSupportsLookupContinueBit
  - kSupportsEnumerateAttributeTypeContinueBit
  - kSupportsEnumeratePseudonymContinueBit
- other capabilities
  - kSupportsFindRecordBit
  - kSupportsAliasesBit
  - kSupportsPseudonymsBit
- reserved features
  - kSupportsAuthenticationBit
  - kSupportsProxiesBit

The bits in a variable of type `DirGestalt` are defined as follows:

```
enum {
   kSupportsDNodeNumberBit                  = 0,
   kSupportsRecordCreationIDBit             = 1,
   kSupportsAttributeCreationIDBit          = 2,
   kSupportsMatchAllBit                     = 3,
   kSupportsBeginsWithBit                   = 4,
   kSupportsExactMatchBit                   = 5,
```

```
   kSupportsEndsWithBit                        = 6,
   kSupportsContainsBit                        = 7,
   kSupportsOrderedEnumerationBit              = 8,
   kCanSupportNameOrderBit                     = 9,
   kCanSupportTypeOrderBit                     = 10,
   kSupportSortBackwardsBit                    = 11,
   kSupportIndexRatioBit                       = 12,
   kSupportsEnumerationContinueBit             = 13,
   kSupportsLookupContinueBit                  = 14,
   kSupportsEnumerateAttributeTypeContinueBit= 15,
   kSupportsEnumeratePseudonymContinueBit    = 16,
   kSupportsAliasesBit                         = 17,
   kSupportsPseudonymsBit                      = 18,
   kSupportsPartialPathNamesBit                = 19,
   kSupportsAuthenticationBit                  = 20,
   kSupportsProxiesBit                         = 21,
   kSupportsFindRecordBit                      = 22
};
```

Bit descriptions

`kSupportsDNodeNumberBit`
: Set this bit if the catalog can identify a dNode by a dNode number. All catalogs must be able to identify a dNode by its pathname.

`kSupportsRecordCreationIDBit`
: Set this bit if a catalog can identify a record by a record creation ID. If a catalog cannot identify a record by a record creation ID, you must set any record creation IDs that you return to 0. All catalogs must support identification of records by record name and record type. If a catalog does not additionally support record creation IDs, the record name and record type must be unique for each record. Note that to assure the proper behavior of aliases, a record creation ID must persist through system shutdown and startup.

`kSupportsAttributeCreationIDBit`
: Set this bit if a catalog can identify an attribute value by specifying its attribute creation ID and attribute type. All catalogs must be able to identify an attribute value by specifying the attribute value and attribute type.

`kSupportsMatchAllBit`
: Set this bit if the catalog supports browsing of record names and record types; that is, when an application calls the `DirEnumerateGet` or `DirFindRecordGet` function, the catalog can service a request to return information about all the records in a dNode or catalog.

`kSupportsBeginsWithBit`
: Set this bit if the catalog supports a search for record names and record types beginning with a certain string; that is, when an application calls the `DirEnumerateGet` or `DirFindRecordGet` function, the catalog can service a request to provide information about all records whose record name or record type begins with the string provided by the application.

`kSupportsExactMatchBit`
: Set this bit if the catalog supports a search for a record based on an exact match with the record name or record type; that is, when an application calls the `DirEnumerateGet` or `DirFindRecordGet` function, the catalog can service a request to provide information about the record whose record name or record type is provided by the application.

`kSupportsEndsWithBit`
: Set this bit if the catalog supports a search for record names and record types ending with a certain string; that is, when an application calls the `DirEnumerateGet` or `DirFindRecordGet` function, the catalog can service a request to provide information about all records whose record name or record type ends with the string provided by the application.

`kSupportsContainsBit`
: Set this bit if the catalog supports a search for record names and record types that contain a certain string; that is, when an application calls the `DirEnumerateGet` or `DirFindRecordGet` function, the catalog can service a request to provide information about all records whose record name or record type contains the string provided by the application.

`kSupportsOrderedEnumerationBit`
: Set this bit if the catalog provides requested information in some sorted order when an application calls the `DirEnumerateGet` function. The catalog may provide the information in an unspecified sorted order. If it returns the information sorted by name or by type, set one or both of the two following bits.

`kCanSupportNameOrderBit`
: Set this bit if the catalog supports the sorting by name option in the `DirEnumerateGet` function. If you set this bit, you must also set the `kSupportsOrderedEnumerationBit` bit.

`kCanSupportTypeOrderBit`
: Set this bit if the catalog supports the sorting by type option in the `DirEnumerateGet` function. If you set this bit, you must also set the `kSupportsOrderedEnumerationBit` bit.

`kSupportSortBackwardsBit`
: Set this bit if the catalog supports the backward sort direction option in the `DirEnumerateGet` function; that is, the catalog can provide entries preceding a certain point and sort those entries in reverse order.

`kSupportIndexRatioBit`
: Set this bit if the catalog supports the index ratio feature in the `DirEnumerateGet` function; that is, the catalog can provide the approximate position of a record among all records in a dNode as a percentile.

`kSupportsEnumerationContinueBit`
: Set this bit if the catalog supports the continue feature in the `DirEnumerateGet` function.

`kSupportsLookupContinueBit`
: Set this bit if the catalog supports the continue feature in the `DirLookupGet` function.

`kSupportsEnumerateAttributeTypeContinueBit`
: Set this bit if the catalog supports the continue feature in the `DirEnumerateAttributeTypesGet` function.

`kSupportsEnumeratePseudonymContinueBit`
: Set this bit if the catalog supports the continue feature in the `DirEnumeratePseudonymGet` function.

`kSupportsAliasesBit`
: Set this bit if the catalog supports adding an alias with the `DirAddAlias` function, deleting an alias with the `DirDeleteRecord` function, and enumerating aliases with the `DirEnumerateGet` function.

`kSupportsPseudonymsBit`
: Set this bit if the catalog supports the `DirAddPseudonym`, `DirDeletePseudonym`, and `DirEnumeratePseudonymGet` functions, and if it supports enumerating pseudonyms with the `DirEnumerateGet` function.

`kSupportsPartialPathNamesBit`
: Set this bit if a catalog can specify a catalog node by using the dNode number of an intermediate dNode and a partial pathname starting from the intermediate dNode to the target dNode.

`kSupportsAuthenticationBit`
: Reserved. Do not set this bit.

`kSupportsProxiesBit`
: Reserved. Do not set this bit.

`kSupportsFindRecordBit`
: Set this bit if the catalog supports the `DirFindRecordGet` and `DirFindRecordParse` functions, that is, it can provide information about records in the entire catalog, rather than in a given dNode. The `DirFindRecordGet` function requests information about records in an entire catalog; the `DirEnumerateGet` function requests information about records in a particular dNode.

These bits are also described from the application's perspective in the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces*.

You can use the following mask values to set the bits in a variable of type `DirGestalt`.

```
enum {
   kSupportsDNodeNumberMask         = 1L<<kSupportsDNodeNumberBit,
   kSupportsRecordCreationIDMask    = 1L<<kSupportsRecordCreationIDBit,
   kSupportsAttributeCreationIDMask = 1L<<kSupportsAttributeCreationIDBit,
   kSupportsMatchAllMask            = 1L<<kSupportsMatchAllBit,
   kSupportsBeginsWithMask          = 1L<<kSupportsBeginsWithBit,
   kSupportsExactMatchMask          = 1L<<kSupportsExactMatchBit,
   kSupportsEndsWithMask            = 1L<<kSupportsEndsWithBit,
   kSupportsContainsMask            = 1L<<kSupportsContainsBit,
   kSupportsOrderedEnumerationMask  = 1L<<kSupportsOrderedEnumerationBit,
   kCanSupportNameOrderMask         = 1L<<kCanSupportNameOrderBit,
   kCanSupportTypeOrderMask         = 1L<<kCanSupportTypeOrderBit,
   kSupportSortBackwardsMask        = 1L<<kSupportSortBackwardsBit,
   kSupportIndexRatioMask           = 1L<<kSupportIndexRatioBit,
   kSupportsEnumerationContinueMask = 1L<<kSupportsEnumerationContinueBit,
   kSupportsLookupContinueMask      = 1L<<kSupportsLookupContinueBit,
   kSupportsEnumerateAttributeTypeContinueMask =
                              1L<<kSupportsEnumerateAttributeTypeContinueBit,
   kSupportsEnumeratePseudonymContinueMask =
                              1L<<kSupportsEnumeratePseudonymContinueBit,
   kSupportsAliasesMask             = 1L<<kSupportsAliasesBit,
   kSupportsPseudonymsMask          = 1L<<kSupportsPseudonymsBit,
   kSupportsPartialPathNamesMask    = 1L<<kSupportsPartialPathNamesBit,
   kSupportsAuthenticationMask      = 1L<<kSupportsAuthenticationBit,
   kSupportsProxiesMask             = 1L<<kSupportsProxiesBit,
   kSupportsFindRecordMask          = 1L<<kSupportsFindRecordBit
};
```

You can define the features that a catalog supports by adding the values of the appropriate masks and storing the resulting value in the CSAM file, where it is available to both your CSAM and your setup template. Listing 3-5 provides an example of specifying the features that a given catalog supports.

Listing 3-5 Setting the feature flags for a catalog

```
#define kPDirFeatures(        \
      kSupportsRecordCreationIDMask\
      + kSupportsAttributeCreationIDMask \
      + kSupportsMatchAllMask \
      + kSupportsBeginsWithMask\
      + kSupportsExactMatchMask \
      + kSupportsOrderedEnumerationMask \
```

```
+ kCanSupportNameOrderMask \
+ kSupportSortBackwardsMask \
+ kSupportsEnumerationContinueMask \
+ kSupportsLookupContinueMask \
)
```

Once you define the features for a given catalog, your setup template passes that information to the Catalog Manager when it calls the `DirAddDSAMDirectory` function to add that catalog to the Setup catalog. The Catalog Manager, in turn, provides the feature flags for a given catalog to an application when the application calls the `DirGetDirectoryInfo` function for a given catalog.

Human Interface Considerations

Although a CSAM itself has no human interface, the features that its catalogs can support affect the human interface provided for those catalogs by certain components of PowerTalk system software. The following components of PowerTalk system software make information in a catalog available to the user:

- the Catalogs Extension (CE)
- the Catalog-Browsing panel in the mailer
- the Find panel in the mailer
- the Find in Catalog command in the Apple menu

The mailer is described in the chapter "Standard Mail Package" in *Inside Macintosh: AOCE Application Interfaces*. For a description of how these elements appear to the user, see the book *PowerTalk User's Guide*.

You need to understand how the settings of certain feature flags affect the user's ability to make use of the information in a catalog using the PowerTalk human interface components. This section notes the capabilities a catalog must support to provide a particular service to the user through the PowerTalk components and the implications of not supporting those capabilities. Here are some service guidelines:

- For catalogs that may contain multiple records with the same name and type, support record creation IDs.
- For catalogs that may contain more than one attribute value of a given attribute type, support attribute creation IDs.
- For a browsable catalog, support "match all" and "exact match" capabilities.
- For proper searching of a catalog, support the "exact match" and "begins with" capabilities and either the "match all" or "find record" capability.
- For efficient handling of large catalogs, support "ordered enumeration," "sort backward," and "enumeration continue" capabilities.
- For best scrolling with large catalogs, support index ratios.
- For efficient attribute lookups, support the "lookup continue" capability.

This information is based on release 1 of the PowerTalk components and is subject to change in future releases.

Supporting Records Having the Same Name and Type

If a catalog allows multiple records to have the same name and type, then it must support record creation IDs. Allowing more than one record with the same name and type without support for record creation IDs creates problems with the CE's user interface. For instance, if a user opens such a catalog, the CE displays the records having the same name and type. If the user then opens one of the records, it is indeterminate which record's attributes are shown to the user. Likewise, if the user makes an alias to such a record, it is not guaranteed that the alias will resolve to the correct record.

Supporting Multiple Attribute Values of the Same Type

If a record in a catalog can contain more than one attribute value of a given attribute type, then you need to support attribute creation IDs for that catalog. The CE requires an attribute creation ID. In the absence of attribute creation IDs, the only way to distinguish among attribute values of the same type is by specifying the attribute value itself. Since attribute values may be as large as 64 KB, this is not efficient, and the attribute creation ID is required for performance reasons. For instance, imagine a record that contains many attributes whose type is Lyric and whose value is the lyric of a popular song. If a user wants to view all of the lyrics, you might run out of buffer space while responding to the `DirLookupGet` function. When the CE calls `DirLookupGet` again to continue the enumeration, it needs a practical way to indicate from which point to continue the enumeration.

If your catalog is unable to support a genuine attribute creation ID that permanently and uniquely identifies an attribute value, then it must support for each attribute value a unique identifier that persists from the time the CSAM is opened at system startup until system shutdown. This unique identifier is called a ***pseudo-persistent attribute creation ID.*** The pseudo-persistent attribute creation ID for a given attribute value is not, by definition, consistent between one session and the next.

Because the CE requires an attribute creation ID when a catalog may contain more than one attribute value of a given attribute type, you must set the `kSupportsAttributeCreationIDBit` bit, regardless of whether the type of attribute creation ID your catalog supports is genuine or pseudo-persistent.

It is desirable that you not reuse a value for a pseudo-persistent attribute creation ID once a session has ended. One way of achieving this is to generate values that incorporate a number derived from the date and time of the session with an incrementing number. This guarantees uniqueness both within and between sessions.

Note
If a catalog's records contain only one attribute value per attribute type, the CE does not require you to support attribute creation IDs. ◆

Supporting Browsing and Finding

If the user can view all of the records in a catalog through the CE or the Catalog-Browsing panel in the mailer, the catalog is browsable. If the user cannot view a catalog's contents, the catalog is nonbrowsable.

A catalog is browsable when both the `kSupportsMatchAllBit` and `kSupportsExactMatchBit` bits are set. A catalog with the "match all" capability supports user browsing by servicing requests to return information on all the records in a dNode or catalog. A browsable catalog must also support an "exact match" capability because, while browsing, a user may make an alias for any object. The "exact match" capability is needed to resolve an alias.

Finding or searching a catalog differs from browsing in that the user specifies, in whole or in part, a particular record name as the target of interest. The Find panel in the mailer and the Find in Catalog command in the Apple menu do not search a catalog unless the following bits are set:

- either the `kSupportsMatchAllBit` or the `kSupportsFindRecordBit` bit
- the `kSupportsExactMatchBit` bit
- the `kSupportsBeginsWithBit` bit
- the `kSupportsEnumerationContinue` bit

Supporting Large Catalogs

The CE and the Catalog-Browsing panel in the mailer attempt to achieve efficiencies in memory requirements and response time when dealing with large catalogs containing many records. This behavior is called ***large-catalog mode.***

The CE and the Catalog-Browsing panel in the mailer can operate in large-catalog mode only if the catalog supports the following capabilities (the relevant bit that must be set is in parentheses):

- catalog can provide records in some sorted order (`kSupportsOrderedEnumerationBit`)
- catalog can provide, in reverse sorted order, the records preceding a specific point (`kSupportSortBackwardsBit`)
- catalog can continue an enumeration from a specified starting point (`kSupportsEnumerationContinueBit`)

If your CSAM provides access to a large catalog that does not provide records in some sorted and reverse sorted order and that cannot continue an enumeration from a specified starting point, you should make the catalog nonbrowsable. This avoids subjecting the user to heavy performance penalties and large memory requirements when working with that catalog. For example, when the CE is not operating in large-catalog mode, it attempts to enumerate all of the records in a given dNode of a catalog, bring the records into memory, and then sort them in the user's system script before displaying any records to the user. If the `DirEnumerateGet` function returns the `kOCEMoreData` result code, the CE calls the function again with a bigger buffer. It starts

the enumeration from the first record since the catalog does not support continuing the enumeration from the last record read. The CE continues to re-enumerate with a bigger buffer until the catalog dNode is completely enumerated or the Macintosh runs out of memory. It could take an unacceptable amount of memory and an unacceptably long time to open a catalog window for a large catalog that does not support large-catalog mode. (When the CE is operating in large-catalog mode, it enumerates either 60 records or three times the number of the records visible in the catalog window, whichever is greater.)

A user can still search for specific records in a large catalog that does not support large-catalog mode, although he or she is unable to view all of the records. The AppleLink address list is an example of a searchable but nonbrowsable catalog.

With large catalogs (those setting the `kSupportsOrderedEnumerationBit`, `kSupportSortBackwardsBit`, and `kSupportsEnumerationContinueBit` bits), the CE and the Catalog-Browsing panel use three different methods of managing scroll bars in a catalog window or panel:

- ratio-approximation
- letter-approximation
- three-position-thumb

The choice of method depends on the capabilities of the catalog being displayed and the script used in the catalog. If the catalog can provide the approximate position of a record within a catalog as a percentile value (an index ratio), it sets the `kSupportIndexRatioBit` bit. When this bit is set, the CE and the Catalog-Browsing panel always use the ratio-approximation method. The ratio-approximation method results in scroll bars that best indicate the true position of a record in a sorted catalog.

If a catalog cannot supply an index ratio, the scrolling method depends on whether the catalog can provide records sorted by record name (`kCanSupportNameOrderBit`) and whether the script used by the Macintosh system software matches the script used by the catalog.

If the catalog can return records in name order and the same script is used by both the catalog and the system software, the letter-approximation method is used. The letter-approximation method uses a table that maps each letter or range of letters in a given script to a number. After determining where the first visible record fits in the complete range of letters, the thumb is set accordingly.

If the scripts differ, the CE and the Catalog-Browsing panel have no idea where the record belongs within the range of letters in the catalog script. Therefore, they use the three-position-thumb method. They also use this method if a catalog cannot provide records sorted by record name. The three-position-thumb method is the least desirable method. It provides a scroll bar having only three positions–at the top of the scroll bar, at the bottom, and in the middle. These positions correspond to the first record in a catalog, the last record, and any other record. Thus, it gives no real information about the majority of records contained in a catalog. It is used as a last resort.

Table 3-1 summarizes the factors that determine the scrolling method.

Table 3-1 Determining the scrolling method for a catalog

Supports index ratio	Supports name order	Scripts	Scrolling method
Yes	Not applicable	Not applicable	Ratio approximation
No	Yes	Match	Letter-approximation
No	Yes	Do not match	Three-position-thumb
No	No	Not applicable	Three-position-thumb

Supporting Attribute Lookups

When the user is looking up attribute values through the CE, the efficiency of the operation depends a great deal on whether the catalog supports the continuation of the attribute lookup (indicated by the `kSupportsLookupContinueBit` bit). If a catalog does not support this feature and the `DirLookupGet` function returns the `kOCEMoreData` result code, the CE calls the function again with a bigger buffer instead of continuing the lookup from the last attribute. The CE continues to do this until all attribute values are completely enumerated or the Macintosh runs out of memory.

Providing Access Controls

You may want to provide access controls to safeguard the content of the catalogs that you support. If a catalog that you support already has its own system of controlling access, you can translate AOCE access controls into those of the external catalog, and vice versa. If a catalog has no access controls, you can implement them in your CSAM. You may provide access controls at the dNode, record, and attribute-type level to limit who may browse the contents of a dNode, record, or attribute type; who may modify the contents; and so forth. See the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* for a complete description of access controls.

To implement access controls, you must know who is making a particular service request. The `identity` field in the `DirParamBlock` parameter block indicates who is making the service request. It may contain the local identity, a specific identity, or 0.

The *local identity* is a reference value that identifies the principal user of the Macintosh computer on which your CSAM is installed. If your CSAM implements access controls, you should obtain the local identity by calling the `AuthGetLocalIdentity` function. When you receive requests for catalog service, compare the value in the `identity` field

in the `DirParamBlock` parameter block with the local identity. If the local identity is making the request, you can then determine if the access privileges of the local identity are sufficient to perform the requested operation.

If the `identity` field contains neither the local identity nor 0, it contains a specific identity. A *specific identity* is a reference value that identifies a user, other than the principal user, who has a PowerShare account. Your CSAM should take whatever action is appropriate, depending on how you choose to handle specific identities. One option, for example, is to treat a specific identity as a guest.

If the `identity` field contains 0, it indicates that a guest made the catalog service request. A guest is anyone other than the principal user and alternate users with PowerShare accounts. If the target catalog supports guest access, you can then determine if the access privileges for a guest are sufficient to perform the requested operation.

See the chapter "Authentication Manager" in *Inside Macintosh: AOCE Application Interfaces* for descriptions of the local identity, specific identities, and the `AuthGetLocalIdentity` function.

Handling Application Completion Routines

An application may provide a pointer to a completion routine when it makes an asynchronous Catalog Manager service request. The completion routine takes a single parameter—a pointer to the parameter block associated with the request.

Your CSAM must call the completion routine that an application provides. You need to

- push the pointer to the parameter block onto the stack (in case the completion routine was written in C or Pascal)
- store the pointer to the parameter block in register A0 (in case the completion routine was written in assembly language)
- store the result code for the function you just serviced in register D0 (in case the completion routine was written in assembly language)
- put the result code for the function in the `ioResult` field of the parameter block

After taking these steps, you set the A5 register to the value of the `saveA5` field of the `DirParamBlock` parameter block and call the completion routine.

You must call a completion routine at deferred-task time to avoid problems when virtual memory may be in use. See the chapters "Virtual Memory Manager" in *Inside Macintosh: Memory* and "Deferred Task Manager" in *Inside Macintosh: Processes* for more information on the handling of virtual memory and deferred tasks.

Listing 3-6 illustrates how you can call an application's completion routine.

Listing 3-6 Calling an application's completion routine

```
DirParamHeader record   0  ;  struct DirParamBlock {
qLink          ds.l     1  ;     Ptr      qLink;
reserved_H1    ds.l     1  ;     long     reserved_H1;
reserved_H2    ds.l     1  ;     long     reserved_H2;
ioCompletion   ds.l     1  ;     ProcPtr  ioCompletion;
ioResult       ds.w     1  ;     OSErr    ioResult;
saveA5         ds.l     1  ;     long     saveA5;
reqCode        ds.w     1  ;     short    reqCode;
               endr        ;  };

CallCompletion proc     export
               with     DirParamHeader
               move.l   4(sp),a0             ;A0 -> parameter block
               move.w   ioResult(a0),d0      ;D0 == ioResult
               move.l   ioCompletion(a0),d1  ;get application completion
               beq.s    @1                   ;exit if none
               move.l   a5,-(sp)             ;save my A5
               move.l   saveA5(a0),a5        ;restore application A5
               link     a6,#0                ;establish new stack frame
               move.l   a0,-(sp)             ;push param block on stack
               move.l   d1,a1                ;put completion routine in A1
               tst.w    d0                   ;set condition codes
               jsr      (a1)                 ;call appl completion routine
               unlk     a6                   ;clean out the stack
               move.l   (sp)+,a5             ;restore my A5
@1             rts                           ;exit from CallCompletion
               endwith
               endp
               end
```

Catalog Service Access Module Reference

This section describes the Catalog Manager functions that a CSAM or its setup template calls and the functions that a CSAM provides. The structures and data types used by these functions are described in the chapters "AOCE Utilities" and "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces*. The Catalog Manager functions that your CSAM supports are described in the chapter "Catalog Manager."

CSAM Functions

This section describes the Catalog Manager functions that you use to initialize a CSAM and to add and remove a CSAM and the external catalogs that it supports.

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

To call a Catalog Manager function from assembly language, push the address of the `DirParamBlock` parameter block and the `async` flag onto the stack using the Pascal calling convention, and place the selector value for the `_oceTBDispatch` trap macro in register D0. Each function description includes the selector value for that function. The function returns its result code in the `ioResult` field of the parameter block. (The `DirParamBlock` parameter block is described in the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces*.)

A CSAM must support asynchronous requests. See the sections "The Catalog Service Function" on page 3-11 and "The Parse Function" on page 3-13 for information on how to support an asynchronous request.

Initializing a CSAM

A CSAM must call the `DirInstantiateDSAM` function before it can receive catalog service requests.

DirInstantiateDSAM

The `DirInstantiateDSAM` function provides the Catalog Manager with the addresses of a CSAM's catalog service and parse functions.

```
pascal OSErr DirInstantiateDSAM (DirParamBlockPtr paramBlock);
```

`paramBlock` Pointer to a parameter block.

Parameter block

←	`ioResult`	`OSErr`	Result code
→	`dsamName`	`RStringPtr`	CSAM name
→	`dsamKind`	`OCEDirectoryKind`	CSAM kind
→	`dsamData`	`Ptr`	CSAM private data
→	`dsamDirProc`	`ProcPtr`	CSAM's catalog service function
→	`dsamDirParseProc`	`ProcPtr`	CSAM's parse function
→	`dsamAuthProc`	`ProcPtr`	Reserved; set to `nil`

Field descriptions

`ioResult`	The result of the function.
`dsamName`	A pointer to the name of the CSAM. You define the name of your CSAM. Use the same name that your setup template provides to the `DirAddDSAM` function.
`dsamKind`	You define this field to identify your CSAM further. Typically, you provide the signature of your CSAM. Use the same value that your setup template provides to the `DirAddDSAM` function.
`dsamData`	A pointer to data that is private to the CSAM. You provide this pointer. The Catalog Manager passes this pointer to the CSAM's catalog service or parse function when an application calls a Catalog Manager function and specifies a catalog that you support.
`dsamDirProc`	A pointer to the CSAM's catalog service function. The Catalog Manager calls the CSAM's catalog service function to process all application requests for catalog services except parse requests. You must provide this value.
`dsamDirParseProc`	A pointer to the CSAM's parse function. The Catalog Manager calls the CSAM's parse function to process an application's parse request. You must provide this value. You can pass the same pointer as you provided in the `dsamDirProc` field if you process all Catalog Manager requests through a single function.
`dsamAuthProc`	Reserved. Set this field to `nil`.

DESCRIPTION

Your CSAM's Open subroutine must call the `DirInstantiateDSAM` function to provide the Catalog Manager with the addresses of the CSAM's catalog service and parse functions. Until you do this, no application can use the services of the CSAM. Note that the addresses (or entry points) can be identical if you simply dispatch the incoming requests to other functions within your CSAM.

The `DirInstantiateDSAM` function is the only function in the Catalog Manager API that is called exclusively by a CSAM.

If the values that you provide in the `dsamName` and `dsamKind` fields do not match those provided by your setup template to the `DirAddDSAM` function, then the `DirInstantiateDSAM` function returns the `kOCEDSAMInstallErr` result code. If this occurs, the Catalog Manager never sends the CSAM any requests.

SPECIAL CONSIDERATIONS

This function is always executed synchronously.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0127

RESULT CODES

`noErr`	0	No error
`kOCELocalAuthenticationFail`	–1561	User hasn't entered Key Chain password
`kOCEDSAMInstallErr`	–1628	Mismatch on CSAM name and kind
`kOCEOCESetupRequired`	–1633	Local identity is not set up
`kOCEDSAMRecordNotFound`	–1634	CSAM record not in Setup catalog

SEE ALSO

The `DirAddDSAM` function, described next, causes the Catalog Manager to install and open a CSAM.

A CSAM's catalog service and parse functions are described in the section "Application-Defined Functions" beginning on page 3-37.

Application signatures are described in the chapter "Finder Interface" in *Inside Macintosh: Macintosh Toolbox Essentials*.

Adding a CSAM and Its Catalogs

The Catalog Manager provides the `DirAddDSAM` and `DirAddDSAMDirectory` functions so that your setup template can add a CSAM and the catalogs it supports to a user's Setup catalog.

DirAddDSAM

The `DirAddDSAM` function opens a CSAM that you specify and adds a record representing the new CSAM to the Setup catalog.

```
pascal OSErr DirAddDSAM (DirParamBlockPtr paramBlock);
```

`paramBlock` Pointer to a parameter block.

Parameter block

←	`ioResult`	`OSErr`	Result code
←	`dsamRecordCID`	`CreationID`	Creation ID of CSAM record
→	`dsamName`	`RStringPtr`	CSAM name
→	`dsamKind`	`OCEDirectoryKind`	CSAM kind
→	`fsSpec`	`FSSpecPtr`	CSAM file specification

Field descriptions

`ioResult` The result of the function.

`dsamRecordCID` The creation ID of the record that the function adds to the Setup catalog. This record represents the CSAM. You pass the CSAM record's creation ID to the `DirAddDSAMDirectory` function when you want to add a catalog that the CSAM supports.

`dsamName`	A pointer to the name of the CSAM. You define the name of your CSAM. Use the same name that your CSAM provides to the `DirInstantiateDSAM` function.
`dsamKind`	You define this field to further identify your CSAM. Typically, you provide the signature of your CSAM. Use the same value that your CSAM provides to the `DirInstantiateDSAM` function.
`fsSpec`	A pointer to the file system specification structure that identifies the file containing the CSAM.

DESCRIPTION

Your setup template calls the `DirAddDSAM` function to install a CSAM and make it available to the user. You call this function before calling the `DirAddDSAMDirectory` function.

The function installs the CSAM in the Device Manager's unit table and opens the driver. The function creates a record for the CSAM. The CSAM record name is the string that you provide in the `dsamName` field; its record type is `aoce DSAM`*xxxx*, where *xxxx* is the value you provide in the `dsamKind` field. The function then adds the new CSAM record to the Setup catalog and returns the record's creation ID.

The `dsamName` and `dsamKind` fields are provided to identify your CSAM. For example, the name of an AppleLink CSAM might be `AppleLink CSAM` whereas its kind might be `ALNK`. The combination of name and kind must be unique among CSAMs installed on the computer.

If the CSAM is already installed, the function provides you with the creation ID of the CSAM record and returns the `kOCEDSAMRecordExists` result code.

SPECIAL CONSIDERATIONS

If your CSAM is a component of a personal MSAM, your setup template calls the `DirAddDSAM` function as part of the combined access module initialization procedure, described in the chapter "Service Access Module Setup" in this book.

This function is always executed synchronously.

There is no registry to guarantee that your CSAM name and kind are unique. To ensure uniqueness, set your CSAM name to your company name or product name and set your CSAM kind to your CSAM's signature that is registered with Macintosh Developer Technical Services.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$011D

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCELocalAuthenticationFail`	–1561	User hasn't entered Key Chain password
`kOCEDSAMInstallErr`	–1628	CSAM could not be installed
`kOCEDSAMRecordExists`	–1636	CSAM record is already in Setup catalog

SEE ALSO

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

The `DirAddDSAMDirectory` function is described next.

To remove a CSAM that you added, use the `DirRemoveDSAM` function (page 3-35).

For more information about the Setup catalog and the CSAM record, see the chapter "Service Access Module Setup" in this book.

Application signatures are described in the chapter "Finder Interface" in *Inside Macintosh: Macintosh Toolbox Essentials*.

DirAddDSAMDirectory

The `DirAddDSAMDirectory` function adds a record for an external catalog to the Setup catalog.

```
pascal OSErr DirAddDSAMDirectory (DirParamBlockPtr paramBlock,
                                  Boolean async);
```

`paramBlock` Pointer to a parameter block.

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

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`clientData`	`long`	You define this field
→	`dsamRecordCID`	`CreationID`	Creation ID of CSAM record
→	`directoryName`	`DirectoryNamePtr`	Name of the catalog
→	`discriminator`	`DirDiscriminator`	Discriminator value
→	`features`	`DirGestalt`	Feature flags
→	`directoryRecordCID`	`CreationID`	Creation ID of Catalog record

Field descriptions

`ioCompletion` A pointer to a completion routine that you can provide. If you call this function asynchronously, it calls your completion routine when it completes execution. Set this field to `nil` if you don't provide a completion routine. The function ignores this field if you call it synchronously.

`ioResult` The result of the function. When you execute the function asynchronously, the function sets this field to 1 as soon as the function has been queued for execution. When the function completes execution, it sets this field to the actual function result code.

`clientData` Reserved for your use. If you call the `DirAddDSAMDirectory` function asynchronously, you can use this field to pass a private value to your completion routine.

`dsamRecordCID` The creation ID of the record representing the CSAM associated with the catalog you want to add. You can obtain the CSAM's record creation ID from the `DirAddDSAM` function.

`directoryName` A pointer to the name of the catalog that you want to add.

`discriminator` A value that distinguishes between two or more catalogs with the same name. You define this value for the catalog you want to add.

`features` The set of feature flags for the catalog you want to add. The flags are described in the section "Indicating the Features You Support" beginning on page 3-16.

`directoryRecordCID` The creation ID of the record for the catalog that you want to add. You obtain the creation ID by using the `CallBackDET` macro to call the `kDETcmdGetDSSpec` callback routine. This provides you with the Catalog record's complete record ID, from which you can extract the creation ID.

DESCRIPTION

Your setup template calls the `DirAddDSAMDirectory` function to add to the Setup catalog a Catalog record for an external catalog that you specify. Once the function successfully completes execution, the external catalog is accessible to the user.

When you add a record for an external catalog, the catalog becomes visible to the `DirEnumerateDirectoriesGet` function. The catalog remains visible and available for use with other Catalog Manager functions until its Catalog record is explicitly removed from the Setup catalog by the `DirRemoveDirectory` function.

(AOCE software creates the Catalog record whose creation ID you provide in the `directoryRecordCID` field. It does this when the user adds a catalog to his or her available catalog services.)

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0133

RESULT CODES

`noErr`	0	No error
`kOCEAlreadyExists`	–1510	Catalog with same name and kind already exists
`kOCELocalAuthenticationFail`	–1561	User hasn't entered Key Chain password
`kOCEDSAMInstallErr`	–1628	CSAM doesn't exist
`kOCEDSAMNotInstantiated`	–1635	CSAM is not instantiated

SEE ALSO

The `DirRemoveDirectory` function is described on page 3-37.

The `DirEnumerateDirectoriesGet` function is described in the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces.*

The `DirAddDSAM` function is described on page 3-31.

For more information about the Setup catalog and the Catalog record, see the chapter "Service Access Module Setup" in this book.

Catalog feature flags are described in the section "Indicating the Features You Support" beginning on page 3-16.

The `CallBackDET` macro and the `kDETcmdGetDSSpec` callback routine are described in the chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces.*

Removing a CSAM and Its Catalogs

The Catalog Manager provides the `DirRemoveDSAM` and `DirRemoveDirectory` functions. Your template uses these functions to remove records for CSAMs and external catalogs from the Setup catalog.

DirRemoveDSAM

The `DirRemoveDSAM` function removes a record for a specific CSAM from the Setup catalog.

```
pascal OSErr DirRemoveDSAM (DirParamBlockPtr paramBlock);
```

`paramBlock` Pointer to a parameter block.

Parameter block

←	`ioResult`	`OSErr`	Result code
→	`dsamRecordCID`	`CreationID`	Creation ID of CSAM record

Field descriptions

`ioResult`	The result of the function.
`dsamRecordCID`	The creation ID of the CSAM record in the Setup catalog for the CSAM that you want to remove. This creation ID is stored in the `kParentDSAMAttrTypeNum` attribute type in the template's record.

DESCRIPTION

Your setup template calls the `DirRemoveDSAM` function to remove a CSAM record from the Setup catalog. The function also closes the CSAM driver and removes from the Setup catalog all Catalog records for catalogs supported by the CSAM.

You can obtain the creation ID of the CSAM record by using the `CallBackDET` macro to call the `kDETcmdGetDSSpec` callback routine. Specify `kDETSelf` as the target to retrieve the `DSSpec` structure that identifies your template record. Then pass that `DSSpec` structure to the `DirLookupGet` function to read the `kParentDSAMAttrTypeNum` attribute type.

Once a CSAM's record is removed from the Setup catalog, the catalogs it serves are unavailable.

Ordinarily, you do not call this function. It is included to provide setup templates with flexibility in handling the CSAM record. For instance, if a user deletes all of the catalogs a CSAM supports, its setup template may remove the CSAM.

SPECIAL CONSIDERATIONS

This function is always executed synchronously.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0120

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEDSAMInstallErr`	–1628	CSAM doesn't exist

SEE ALSO

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

For more information about the Setup catalog, see the chapter "Service Access Module Setup" in this book.

You can add a CSAM to the Setup catalog by calling the `DirAddDSAM` function (page 3-31).

For information about the `CallBackDET` macro and the `kDETcmdGetDSSpec` callback routine, see the chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces*.

The `DirLookupGet` function is described in the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces*.

DirRemoveDirectory

The `DirRemoveDirectory` function removes from the Setup catalog a record that represents a catalog.

Because the function is not limited to removing external catalogs, it is described in the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces*.

Application-Defined Functions

You provide the catalog service and parse functions described in this section. You pass their addresses to the Catalog Manager when you call the `DirInstantiateDSAM` function. The Catalog Manager calls your functions when an application requests a service from an external catalog that you support. It is through these functions that you supply catalog services.

MyDSAMDirProc

The `MyDSAMDirProc` function accepts and processes Catalog Manager requests for catalog services. You must provide this function as part of your CSAM.

```
pascal OSErr MyDSAMDirProc (Ptr dsamData,
                            DirParamBlockPtr paramBlock,
                            Boolean async);
```

`dsamData` A pointer to the CSAM's private data. This is the value that you previously passed to the `DirInstantiateDSAM` function in the `dsamData` field of its `DirParamBlock` parameter block.

`paramBlock` A pointer to the parameter block that the application passed to the Catalog Manager when the application called a Catalog Manager function.

`async` A Boolean value that specifies if the request must be processed synchronously or asynchronously. If this field is set to `true`, you must process the request asynchronously.

DESCRIPTION

The Catalog Manager calls your catalog service function when an application requests a service, other than parse, from a catalog supported by your CSAM. You determine the type of request by examining the `reqCode` field in the `DirParamBlock` parameter block. Each possible value of the `reqCode` field corresponds to a Catalog Manager function. You then process the request and return the necessary information in the fields of the `paramBlock` parameter block.

RESULT CODES

Each type of service request that you may receive corresponds to a single Catalog Manager function. For each type of service request that you process, you should return only those result codes that are defined by the Catalog Manager for the corresponding function. See the description of each Catalog Manager function for the list of result codes you can return for that function.

SEE ALSO

The section "The Catalog Service Function" on page 3-11 provides general information on the actions that your `MyDSAMDirProc` function should take while servicing a request for catalog services. You decide how to implement a given Catalog Manager function for the catalog that you support.

The `DirInstantiateDSAM` function is described on page 3-29.

The chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* contains descriptions of each Catalog Manager function.

The request codes that may appear in the `reqCode` field in the `DirParamBlock` parameter block are listed in "Data Types and Constants" beginning on page 3-42.

MyDSAMDirParseProc

The `MyDSAMDirParseProc` function accepts and processes Catalog Manager parse requests. You may provide this function as part of your CSAM.

```
pascal OSErr MyDSAMDirParseProc (Ptr dsamData,
                                 DirParamBlockPtr paramBlock,
                                 Boolean async);
```

`dsamData` A pointer to the CSAM's private data. This is the value that you previously passed to the `DirInstantiateDSAM` function in the `dsamData` field of its `DirParamBlock` parameter block.

`paramBlock` A pointer to the parameter block that the application provided to the Catalog Manager when the application made a parse request.

`async` A Boolean value that specifies if the request must be processed synchronously or asynchronously. If this field is set to `true`, you must process the request asynchronously.

DESCRIPTION

The Catalog Manager calls your parse function when an application makes a parse request and specifies a catalog that your CSAM supports. You determine the specific type of parse request by examining the `reqCode` field in the `DirParamBlock` parameter block. Each possible value of the `reqCode` field corresponds to a Catalog Manager function. You then process the request by returning the necessary information in the fields of the parameter block and calling the application's callback routine.

SPECIAL CONSIDERATIONS

You can choose to dispatch all service requests through a single function. In that case, you don't provide a separate and distinct parse function. Instead, you pass the same address to the `DirInstantiateDSAM` function in both the `dsamDirProc` and `dsamDirParseProc` fields.

RESULT CODES

Each type of parse request that you may receive corresponds to a single Catalog Manager function. For each type of parse request that you process, you should return only those result codes that are defined by the Catalog Manager for the corresponding function. See the description of each Catalog Manager function for the list of result codes you can return for that function.

SEE ALSO

The sections "The Catalog Service Function" on page 3-11 and "The Parse Function" on page 3-13 provide general information on the actions that your `MyDSAMDirParseProc` function should take while servicing a parse request. You decide how to implement a given Catalog Manager parse function for the catalog that you support.

The chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* contains descriptions of each Catalog Manager function.

The request codes that may appear in the `reqCode` field in the `DirParamBlock` parameter block are listed in the section "Data Types and Constants" beginning on page 3-42.

Resources

This section describes the 'DRVR' resource type that you provide in a CSAM.

The Driver Resource

The driver resource contains the executable code that implements support for the Catalog Manager API. Listing 3-7 shows the Rez definition of the 'DRVR' resource type.

Listing 3-7 'DRVR' resource definition

```
type 'DRVR' {
   boolean = 0;                          /* unused */
   boolean dontNeedLock,   needLock;     /* lock driver in memory */
   boolean dontNeedTime,   needTime;     /* for periodic action */
   boolean dontNeedGoodbye,needGoodbye;  /* call before heap reinit */
   boolean noStatusEnable, statusEnable; /* responds to Status */
   boolean noCtlEnable,    ctlEnable;    /* responds to Control */
   boolean noWriteEnable,  writeEnable;  /* responds to Write */
   boolean noReadEnable,   readEnable;   /* responds to Read */
   byte = 0;                             /* unused */
   unsigned integer;                     /* driver delay (ticks) */
   integer;                              /* DA event mask */
   integer;                              /* driver menu ID */
   unsigned integer = 50;                /* offset to DRVRRuntime Open */
   unsigned integer = 54;                /* offset to DRVRRuntime Prime */
   unsigned integer = 58;                /* offset to DRVRRuntime Control */
   unsigned integer = 62;                /* offset to DRVRRuntime Status */
   unsigned integer = 66;                /* offset to DRVRRuntime Close */
   pstring[31];                          /* driver name */
   hex string;                           /* driver code */
};
```

The driver resource contains the following fields:

- An unused Boolean value.
- A Boolean value that indicates if your driver should be locked in memory. You must set this to needLock for a CSAM.
- A Boolean value that indicates if your driver should receive processor time periodically. Set this according to the needs of your CSAM.
- A Boolean value that indicates if your driver should be notified before the application heap is reinitialized. Because your CSAM driver must reside in the system heap, this Boolean value is irrelevant.

- A Boolean value that indicates if your driver responds to Status calls from the Device Manager.
- A Boolean value that indicates if your driver responds to Control calls from the Device Manager.
- A Boolean value that indicates if your driver responds to Write calls from the Device Manager.
- A Boolean value that indicates if your driver responds to Read calls from the Device Manager.
- An unused value.
- A value that indicates the number of ticks between your periodic time intervals. If you have already specified the Boolean value `needTime`, set this according to the needs of your CSAM.
- A value for desk accessories. It is irrelevant to a CSAM.
- A value for desk accessories. It is irrelevant to a CSAM.
- Five 4-byte values that specify the offsets to your Open, Prime, Control, Status, and Close driver subroutines, respectively.
- The name of your CSAM driver. You can use uppercase and lowercase letters when naming your driver, but the first character must be a period.
- The hexadecimal representation of your executable code. Your driver subroutines must be aligned on a word boundary.

Summary of Catalog Service Access Modules

C Summary

Data Types and Constants

```
enum {      /* feature flag bits */
   kSupportsDNodeNumberBit                  = 0,
   kSupportsRecordCreationIDBit             = 1,
   kSupportsAttributeCreationIDBit          = 2,
   kSupportsMatchAllBit                     = 3,
   kSupportsBeginsWithBit                   = 4,
   kSupportsExactMatchBit                   = 5,
   kSupportsEndsWithBit                     = 6,
   kSupportsContainsBit                     = 7,
   kSupportsOrderedEnumerationBit           = 8,
   kCanSupportNameOrderBit                  = 9,
   kCanSupportTypeOrderBit                  = 10,
   kSupportSortBackwardsBit                 = 11,
   kSupportIndexRatioBit                    = 12,
   kSupportsEnumerationContinueBit          = 13,
   kSupportsLookupContinueBit               = 14,
   kSupportsEnumerateAttributeTypeContinueBit= 15,
   kSupportsEnumeratePseudonymContinueBit   = 16,
   kSupportsAliasesBit                      = 17,
   kSupportsPseudonymsBit                   = 18,
   kSupportsPartialPathNamesBit             = 19,
   kSupportsAuthenticationBit               = 20,
   kSupportsProxiesBit                      = 21,
   kSupportsFindRecordBit                   = 22
};

enum {      /* feature flag masks */
   kSupportsDNodeNumberMask         = 1L<<kSupportsDNodeNumberBit,
   kSupportsRecordCreationIDMask    = 1L<<kSupportsRecordCreationIDBit,
   kSupportsAttributeCreationIDMask = 1L<<kSupportsAttributeCreationIDBit,
   kSupportsMatchAllMask            = 1L<<kSupportsMatchAllBit,
   kSupportsBeginsWithMask          = 1L<<kSupportsBeginsWithBit,
   kSupportsExactMatchMask          = 1L<<kSupportsExactMatchBit,
```

```
    kSupportsEndsWithMask            = 1L<<kSupportsEndsWithBit,
    kSupportsContainsMask            = 1L<<kSupportsContainsBit,
    kSupportsOrderedEnumerationMask  = 1L<<kSupportsOrderedEnumerationBit,
    kCanSupportNameOrderMask         = 1L<<kCanSupportNameOrderBit,
    kCanSupportTypeOrderMask         = 1L<<kCanSupportTypeOrderBit,
    kSupportSortBackwardsMask        = 1L<<kSupportSortBackwardsBit,
    kSupportIndexRatioMask           = 1L<<kSupportIndexRatioBit,
    kSupportsEnumerationContinueMask = 1L<<kSupportsEnumerationContinueBit,
    kSupportsLookupContinueMask      = 1L<<kSupportsLookupContinueBit,
    kSupportsEnumerateAttributeTypeContinueMask =
                              1L<<kSupportsEnumerateAttributeTypeContinueBit,
    kSupportsEnumeratePseudonymContinueMask =
                              1L<<kSupportsEnumeratePseudonymContinueBit,
    kSupportsAliasesMask             = 1L<<kSupportsAliasesBit,
    kSupportsPseudonymsMask          = 1L<<kSupportsPseudonymsBit,
    kSupportsPartialPathNamesMask    = 1L<<kSupportsPartialPathNamesBit,
    kSupportsAuthenticationMask      = 1L<<kSupportsAuthenticationBit,
    kSupportsProxiesMask             = 1L<<kSupportsProxiesBit,
    kSupportsFindRecordMask          = 1L<<kSupportsFindRecordBit
};

/* request codes for Catalog Manager functions */
#define kDirEnumerateParse                  0x101
#define kDirLookupParse                     0x102
#define kDirEnumerateAttributeTypesParse    0x103
#define kDirEnumeratePseudonymParse         0x104
#define kDirNetSearchADAPDirectoriesParse   0x105
#define kDirEnumerateDirectoriesParse       0x106
#define kDirFindADAPDirectoryByNetSearch    0x107
#define kDirNetSearchADAPDirectoriesGet     0x108
#define kDirAddRecord                       0x109
#define kDirDeleteRecord                    0x10A
#define kDirAddAttributeValue               0x10B
#define kDirDeleteAttributeValue            0x10C
#define kDirChangeAttributeValue            0x10D
#define kDirVerifyAttributeValue            0x10E
#define kDirAddPseudonym                    0x10F
#define kDirDeletePseudonym                 0x110
#define kDirEnumerateGet                    0x111
#define kDirEnumerateAttributeTypesGet      0x112
#define kDirEnumeratePseudonymGet           0x113
#define kDirGetNameAndType                  0x114
#define kDirSetNameAndType                  0x115
#define kDirGetRecordMetaInfo               0x116
```

```
#define kDirLookupGet                          0x117
#define kDirGetDNodeMetaInfo                   0x118
#define kDirGetDirectoryInfo                   0x119
#define kDirEnumerateDirectoriesGet            0x11A
#define kDirAbort                              0x11B
#define kDirAddAlias                           0x11C
#define kDirAddDSAM                            0x11D
#define kDirOpenPersonalDirectory              0x11E
#define kDirCreatePersonalDirectory            0x11F
#define kDirRemoveDSAM                         0x120
#define kDirGetDirectoryIcon                   0x121
#define kDirMapPathNameToDNodeNumber           0x122
#define kDirMapDNodeNumberToPathName           0x123
#define kDirGetLocalNetworkSpec                0x124
#define kDirGetDNodeInfo                       0x125
#define kDirFindValue                          0x126
#define kDirInstantiateDSAM                    0x127
#define kDirGetOCESetupRefNum                  0x128
#define kDirGetDNodeAccessControlGet           0x12A
#define kDirGetRecordAccessControlGet          0x12C
#define kDirGetAttributeAccessControlGet       0x12E
#define kDirGetDNodeAccessControlParse         0x12F
#define kDirDeleteAttributeType                0x130
#define kDirClosePersonalDirectory             0x131
#define kDirMakePersonalDirectoryRLI           0x132
#define kDirAddDSAMDirectory                   0x133
#define kDirGetRecordAccessControlParse        0x134
#define kDirRemoveDirectory                    0x135
#define kDirGetExtendedDirectoriesInfo         0x136
#define kDirAddADAPDirectory                   0x137
#define kDirGetAttributeAccessControlParse     0x138
#define kDirFindRecordGet                      0x140
#define kDirFindRecordParse                    0x141

struct DirInstantiateDSAMPB {
   AuthDirParamHeader                  /* parameter block header */
   RStringPtr        dsamName;         /* CSAM name */
   OCEDirectoryKind  dsamKind;         /* CSAM kind */
   Ptr               dsamData;         /* CSAM private data */
   ProcPtr           dsamDirProc;      /* catalog service function */
   ProcPtr           dsamDirParseProc; /* parse function */
   ProcPtr           dsamAuthProc;     /* reserved, set to nil */
};

typedef struct DirInstantiateDSAMPB DirInstantiateDSAMPB;
```

```
struct DirAddDSAMPB {
   AuthDirParamHeader              /* parameter block header */
   CreationID        dsamRecordCID; /* CSAM record creation ID */
   RStringPtr        dsamName;      /* CSAM name */
   OCEDirectoryKind  dsamKind;      /* CSAM kind */
   FSSpecPtr         fsSpec;        /* CSAM's file specification */
};

typedef struct DirAddDSAMPB DirAddDSAMPB;

struct DirAddDSAMDirectoryPB {
   AuthDirParamHeader              /* parameter block header */
   CreationID        dsamRecordCID; /* CSAM record creation ID */
   DirectoryNamePtr  directoryName; /* catalog name */
   DirDiscriminator  discriminator; /* catalog discriminator value */
   DirGestalt        features;      /* feature flags for the catalog */
   CreationID        directoryRecordCID;
                                    /* Catalog record creation ID */
};

typedef struct DirAddDSAMDirectoryPB DirAddDSAMDirectoryPB;

struct DirRemoveDSAMPB {
   AuthDirParamHeader            /* parameter block header */
   CreationID  dsamRecordCID;    /* CSAM record creation ID */
};

typedef struct DirRemoveDSAMPB DirRemoveDSAMPB;
```

CSAM Functions

Initializing a CSAM

```
pascal OSErr DirInstantiateDSAM
                           (DirParamBlockPtr paramBlock);
```

Adding a CSAM and Its Catalogs

```
pascal OSErr DirAddDSAM     (DirParamBlockPtr paramBlock);
pascal OSErr DirAddDSAMDirectory
                           (DirParamBlockPtr paramBlock, Boolean async);
```

Removing a CSAM and Its Catalogs

```
pascal OSErr DirRemoveDSAM  (DirParamBlockPtr paramBlock);
```

Application-Defined Functions

```
pascal OSErr MyDSAMDirProc  (Ptr dsamData, DirParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr MyDSAMDirParseProc
                            (Ptr dsamData, DirParamBlockPtr paramBlock,
                             Boolean async);
```

Pascal Summary

Data Types and Constants

```
CONST
   { feature flag bits }
   kSupportsDNodeNumberBit                    = 0;
   kSupportsRecordCreationIDBit               = 1;
   kSupportsAttributeCreationIDBit            = 2;
   kSupportsMatchAllBit                       = 3;
   kSupportsBeginsWithBit                     = 4;
   kSupportsExactMatchBit                     = 5;
   kSupportsEndsWithBit                       = 6;
   kSupportsContainsBit                       = 7;
   kSupportsOrderedEnumerationBit             = 8;
   kCanSupportNameOrderBit                    = 9;
   kCanSupportTypeOrderBit                    = 10;
   kSupportSortBackwardsBit                   = 11;
   kSupportIndexRatioBit                      = 12;
   kSupportsEnumerationContinueBit            = 13;
   kSupportsLookupContinueBit                 = 14;
   kSupportsEnumerateAttributeTypeContinueBit = 15;
   kSupportsEnumeratePseudonymContinueBit     = 16;
   kSupportsAliasesBit                        = 17;
   kSupportsPseudonymsBit                     = 18;
   kSupportsPartialPathNamesBit               = 19;
   kSupportsAuthenticationBit                 = 20;
   kSupportsProxiesBit                        = 21;
   kSupportsFindRecordBit                     = 22;

   { feature flag masks }
   kSupportsDNodeNumberMask                   = $00000001;
   kSupportsRecordCreationIDMask              = $00000002;
   kSupportsAttributeCreationIDMask           = $00000004;
```

```
kSupportsMatchAllMask                          = $00000008;
kSupportsBeginsWithMask                        = $00000010;
kSupportsExactMatchMask                        = $00000020;
kSupportsEndsWithMask                          = $00000040;
kSupportsContainsMask                          = $00000080;
kSupportsOrderedEnumerationMask                = $00000100;
kCanSupportNameOrderMask                       = $00000200;
kCanSupportTypeOrderMask                       = $00000400;
kSupportSortBackwardsMask                      = $00000800;
kSupportIndexRatioMask                         = $00001000;
kSupportsEnumerationContinueMask               = $00002000;
kSupportsLookupContinueMask                    = $00004000;
kSupportsEnumerateAttributeTypeContinueMask    = $00008000;
kSupportsEnumeratePseudonymContinueMask        = $00010000;
kSupportsAliasesMask                           = $00020000;
kSupportsPseudonymsMask                        = $00040000;
kSupportsPartialPathNamesMask                  = $00080000;
kSupportsAuthenticationMask                    = $00100000;
kSupportsProxiesMask                           = $00200000;
kSupportsFindRecordMask                        = $00400000;

{ request codes for Catalog Manager requests }
kDirEnumerateParse                             $101
kDirLookupParse                                $102
kDirEnumerateAttributeTypesParse               $103
kDirEnumeratePseudonymParse                    $104
kDirNetSearchADAPDirectoriesParse              $105
kDirEnumerateDirectoriesParse                  $106
kDirFindADAPDirectoryByNetSearch               $107
kDirNetSearchADAPDirectoriesGet                $108
kDirAddRecord                                  $109
kDirDeleteRecord                               $10A
kDirAddAttributeValue                          $10B
kDirDeleteAttributeValue                       $10C
kDirChangeAttributeValue                       $10D
kDirVerifyAttributeValue                       $10E
kDirAddPseudonym                               $10F
kDirDeletePseudonym                            $110
kDirEnumerateGet                               $111
kDirEnumerateAttributeTypesGet                 $112
kDirEnumeratePseudonymGet                      $113
kDirGetNameAndType                             $114
kDirSetNameAndType                             $115
kDirGetRecordMetaInfo                          $116
```

```
   kDirLookupGet                              $117
   kDirGetDNodeMetaInfo                       $118
   kDirGetDirectoryInfo                       $119
   kDirEnumerateDirectoriesGet                $11A
   kDirAbort                                  $11B
   kDirAddAlias                               $11C
   kDirAddDSAM                                $11D
   kDirOpenPersonalDirectory                  $11E
   kDirCreatePersonalDirectory                $11F
   kDirRemoveDSAM                             $120
   kDirGetDirectoryIcon                       $121
   kDirMapPathNameToDNodeNumber               $122
   kDirMapDNodeNumberToPathName               $123
   kDirGetLocalNetworkSpec                    $124
   kDirGetDNodeInfo                           $125
   kDirFindValue                              $126
   kDirInstantiateDSAM                        $127
   kDirGetOCESetupRefNum                      $128
   kDirGetDNodeAccessControlGet               $12A
   kDirGetRecordAccessControlGet              $12C
   kDirGetAttributeAccessControlGet           $12E
   kDirGetDNodeAccessControlParse             $12F
   kDirDeleteAttributeType                    $130
   kDirClosePersonalDirectory                 $131
   kDirMakePersonalDirectoryRLI               $132
   kDirAddDSAMDirectory                       $133
   kDirGetRecordAccessControlParse            $134
   kDirRemoveDirectory                        $135
   kDirGetExtendedDirectoriesInfo             $136
   kDirAddADAPDirectory                       $137
   kDirGetAttributeAccessControlParse         $138
   kDirFindRecordGet                          $140
   kDirFindRecordParse                        $141

DirInstantiateDSAMPB = RECORD
   qLink:         Ptr;           { reserved }
   reserved1:     LONGINT;       { reserved }
   reserved2:     LONGINT;       { reserved }
   ioCompletion:  ProcPtr;       { your completion routine }
   ioResult:      OSErr;         { result code }
   saveA5:        LONGINT;       { reserved }
   reqCode:       INTEGER;       { Catalog Manager function request code }
   reserved:      ARRAY[1..2] OF LONGINT;
                                 { reserved }
```

```
    serverHint:    AddrBlock;     { PowerShare server's AppleTalk address }
    dsRefNum:      INTEGER;       { personal catalog reference number }
    callID:        LONGINT;       { reserved }
    identity:      AuthIdentity;  { requestor's authentication identity }
    gReserved1:    LONGINT;       { reserved }
    gReserved2:    LONGINT;       { reserved }
    gReserved3:    LONGINT;       { reserved }
    clientData:    LONGINT;       { you define this field }
    dsamName:      RStringPtr;    { CSAM name }
    dsamKind:      OCEDirectoryKind;
                                  { CSAM kind }
    dsamData:      Ptr;           { CSAM private data }
    dsamDirProc:   ProcPtr;       { CSAM's catalog service routine }
    dsamDirParseProc: ProcPtr;    { CSAM's parse routine }
    dsamAuthProc:  ProcPtr;       { reserved }
    END;

DirAddDSAMPB = RECORD
    qLink:         Ptr;           { reserved }
    reserved1:     LONGINT;       { reserved }
    reserved2:     LONGINT;       { reserved }
    ioCompletion:  ProcPtr;       { your completion routine }
    ioResult:      OSErr;         { result code }
    saveA5:        LONGINT;       { reserved }
    reqCode:       INTEGER;       { Catalog Manager function request code }
    reserved:      ARRAY[1..2] OF LONGINT;
                                  { reserved }
    serverHint:    AddrBlock;     { PowerShare server's AppleTalk address }
    dsRefNum:      INTEGER;       { personal catalog reference number }
    callID:        LONGINT;       { reserved }
    identity:      AuthIdentity;  { requestor's authentication identity }
    gReserved1:    LONGINT;       { reserved }
    gReserved2:    LONGINT;       { reserved }
    gReserved3:    LONGINT;       { reserved }
    clientData:    LONGINT;       { you define this field }
    dsamRecordCID: CreationID;    { creation ID of CSAM record }
    dsamName:      RStringPtr;    { CSAM name }
    dsamKind:      OCEDirectoryKind;
                                  { CSAM kind }
    fsSpec:        FSSpecPtr;     { CSAM file specification }
    END;
```

```
DirAddDSAMDirectoryPB = RECORD
   qLink:          Ptr;           { reserved }
   reserved1:      LONGINT;       { reserved }
   reserved2:      LONGINT;       { reserved }
   ioCompletion:   ProcPtr;       { your completion routine }
   ioResult:       OSErr;         { result code }
   saveA5:         LONGINT;       { reserved }
   reqCode:        INTEGER;       { Catalog Manager function request code }
   reserved:       ARRAY[1..2] OF LONGINT;
                                  { reserved }
   serverHint:     AddrBlock;     { PowerShare server's AppleTalk address }
   dsRefNum:       INTEGER;       { personal catalog reference number }
   callID:         LONGINT;       { reserved }
   identity:       AuthIdentity;  { requestor's authentication identity }
   gReserved1:     LONGINT;       { reserved }
   gReserved2:     LONGINT;       { reserved }
   gReserved3:     LONGINT;       { reserved }
   clientData:     LONGINT;       { you define this field }
   dsamRecordCID: CreationID;     { creation ID of CSAM record }
   directoryName: DirectoryNamePtr;    { catalog name }
   discriminator: DirDiscriminator;    { discriminator value }
   features:       DirGestalt;          { feature flags for the catalog }
   directoryRecordCID: CreationID;     { creation ID of catalog record }
   END;

DirRemoveDSAMPB = RECORD
   qLink:          Ptr;           { reserved }
   reserved1:      LONGINT;       { reserved }
   reserved2:      LONGINT;       { reserved }
   ioCompletion:   ProcPtr;       { your completion routine }
   ioResult:       OSErr;         { result code }
   saveA5:         LONGINT;       { reserved }
   reqCode:        INTEGER;       { Catalog Manager function request code }
   reserved:       ARRAY[1..2] OF LONGINT;
                                  { reserved }
   serverHint:     AddrBlock;     { PowerShare server's AppleTalk address }
   dsRefNum:       INTEGER;       { personal catalog reference number }
   callID:         LONGINT;       { reserved }
   identity:       AuthIdentity;  { requestor's authentication identity }
   gReserved1:     LONGINT;       { reserved }
   gReserved2:     LONGINT;       { reserved }
   gReserved3:     LONGINT;       { reserved }
   clientData:     LONGINT;       { you define this field }
   dsamRecordCID: CreationID;     { creation ID of CSAM record }
   END;
```

CSAM Functions

Initializing a CSAM

```
FUNCTION DirInstantiateDSAM (paramBlock: DirParamBlockPtr): OSErr;
```

Adding a CSAM and Its Catalogs

```
FUNCTION DirAddDSAM          (paramBlock: DirParamBlockPtr): OSErr;
FUNCTION DirAddDSAMDirectory(paramBlock: DirParamBlockPtr;
                              async: BOOLEAN): OSErr;
```

Removing a CSAM and Its Catalogs

```
FUNCTION DirRemoveDSAM       (paramBlock: DirParamBlockPtr): OSErr;
```

Application-Defined Functions

```
FUNCTION MyDSAMDirFunc       (dsamData: Ptr; paramBlock: DirParamBlockPtr;
                              async: BOOLEAN): OSErr;
FUNCTION MyDSAMDirParseFunc  (dsamData: Ptr; paramBlock: DirParamBlockPtr;
                              async: BOOLEAN): OSErr;
```

Assembly-Language Summary

Trap Macros

Trap Macros Requiring Routine Selectors

```
_oceTBDispatch
```

Selector	Routine
0$127	DirInstantiateDSAM
0$11D	DirAddDSAM
0$133	DirAddDSAMDirectory
0$120	DirRemoveDSAM
0$135	DirRemoveDirectory

Result Codes

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEAlreadyExists`	–1510	Catalog with same name and kind already exists
`kOCELocalAuthenticationFail`	–1561	User hasn't entered Key Chain password
`kOCEDSAMInstallErr`	–1628	CSAM could not be installed or doesn't exist
`kOCEOCESetupRequired`	–1633	Local identity is not set up
`kOCEDSAMRecordNotFound`	–1634	CSAM record not in Setup catalog
`kOCEDSAMNotInstantiated`	–1635	CSAM is not instantiated
`kOCEDSAMRecordExists`	–1636	CSAM record is already in Setup catalog